Published in IET Computers & Digital Techniques
Received on 9th July 2008
Revised on 18th November 2008
doi: 10.1049/iet-cdt.2008.0097

ISSN 1751-8601

Analysis of shared-link AXI

N.Y.-C. Chang[1] *Y.-Z. Liao*[2] *T.-S. Chang*[1]
[1]*Department of Electronic Engineering, National Chiao Tung University, 1001 Ta-Hsueh Rd., Hsinchu 30010, Taiwan*
[2]*Global Unichip Corp., No. 10. Li-Hsin 6th Road, Hsinchu Science Park, Hsinchu City 300, Taiwan*
E-mail: ycchang@twins.ee.nctu.edu.tw

Abstract: Shared-link AXI provides decent communication performance and requires half the cost of its crossbar counterpart. The authors analysed the performance impact of the factors in a shared-link AXI system. The factors include interface buffer size, arbitration combination and task access setting (transfer mode mapping). A hybrid data locked transfer mode was also proposed to improve the performance due to AXI's extra transition cycle. The analysis is carried out by simulating a multi-core platform with a shared-link AXI backbone running a video phone application. The performance is evaluated in terms of bandwidth utilisation, average transaction latency and system task completion time. The analysis showed that channel-independent arbitration could contribute up to 23.2% of bandwidth utilisation and completion time difference. Moreover, the analysis suggests that the proposed hybrid data locked mode should be used only by long access latency devices. Such setting resulted in up to 21.1% completion time reduction compared with the setting without the hybrid data locked mode. The design options in shared-link AXI bus are also discussed.

1 Introduction

With the rapid progress of system-on-a-chip (SOC) and massive data movement requirement, on-chip system bus becomes the central role in determining the performance of a SOC. Two types of on-chip bus have been widely used in current designs: pipelined-based bus and packet-based bus.

For pipelined-based buses, such as ARM's AMBA 2.0 AHB [1], IBM's CoreConnect [2] and OpenCore's WishBone [3], the cost and complexity to bridge the communications among on-chip designs are low. However, pipeline-based bus suffers from bus contention and inherent blocking characteristics due to the protocol. The contention issue can be alleviated by adopting multi-layer bus structure [4] or using proper arbitration policies [5, 6]. However, the blocking characteristic, which allows a transfer to complete only if the previous transfer has completed, cannot be altered without changing the bus protocol. This blocking characteristic reduces the bus bandwidth utilisation when accessing long latency devices, such as an external memory controller.

To cope with the issues of pipelined-based buses, packet-based buses such as ARM's AMBA 3.0 AXI [7], OCP-IP's Open Core Protocol (OCP) [8], and STMicroelectronics' STBus [9] have been proposed to support outstanding transfer and out-of-order transfer completion. We will focus on AXI here because of its popularity. AXI bus possesses multiple independent channels to support multiple simultaneous address and data streams. Besides, AXI also supports improved burst operation, register slicing with registered input and secured transfer.

Despite the above features, AXI requires high cost and possesses long transaction handshaking latency. However, a shared-link AXI interconnect can provide good performance while requiring less than half of the hardware required by a crossbar AXI implementation. This work focused on the performance analysis of a shared-link AXI. The handshaking latency is at least two cycles if the interface or interconnect is designed with registered input. This would limit the bandwidth utilisation to less than 50%. To reduce the handshaking latency, we proposed a hybrid data locked transfer mode. Unlike the lock transfer in [10] which requires arbitration lock over transactions, our data locked mode is based on a transfer-level arbitration scheme and allows bus ownership to change between transactions. This gives more flexibility to arbitration policy selection.

With the additional features of AXI, new factors that affect the bus performance are also introduced. The first factor is the arbitration combination. The multi-channel architecture allows different and independent arbitration policies to be adopted by each channel. However, existing AXI-related works often assumed a unified arbitration policy where each channel adopts the same arbitration policy [10–12]. Another key factor is the interface buffer size. A larger interface buffer usually implies that more out-of-order transactions can be handled. The third factor is the task access setting, which defines how the transfer modes should be used by the devices within a system. Proper task access settings can yield better performance. However, the proper setting may be different under different circumstances, such as different buffer sizes.

Being aware of the performance factors mentioned above, we conducted a detailed simulation-based analysis on the performance impact of the factors. The analysis is carried out by simulating a multi-core platform with a shared-link AXI backbone running a video phone application. The performance is evaluated in terms of bandwidth utilisation, average transaction latency and system task completion time. In addition to the analysis on the performance impact of the aforementioned factors, the performance of a corresponding five-layer AHB-lite bus, which has a cost comparable to a 5-channel shared-link AXI, is also included for comparison.

The rest of the paper is organised as follows. Section 2 presents the related works on AXI bus. Section 3 presents the proposed transfer modes and Section 4 explains the corresponding arbitration framework. Section 5 presents the simulation platform and evaluation metrics for performance comparison. The comparison of the simulation result is available in Section 6. Finally, Section 7 concludes this work.

2 Related works

Many works have been conducted on the communication architecture of pipelined-based bus. Earlier works used formal analytic approach [13, 14] to explore the design space of communication architecture and evaluate the performance of a pipeline-based bus system. Although formal analytic approach can provide the average or best/worst case overall bus performance, such an approach can hardly account for instantaneous changes of bus behaviour. This limitation gave rise to high-level simulation-based approach which is capable of capturing the detailed instantaneous bus behaviour with cycle accuracy [15]. Pasricha *et al.* [16] used the cycle count accurate transaction boundaries model in the architecture exploration of an MPEG AHB system. Later, Pasricha *et al.* [17] also conducted experiment on bus architecture synthesis under different given constraints. Their synthesis method yielded cost-efficient bus matrices much faster and reliable than manual optimisation.

Most of the techniques developed in the abovementioned works can be extended for the analysis of packet-based bus. Pasricha *et al.* [12] extended their communication architecture synthesis framework to AXI. Their work automatically generates the best bus topology, arbitration policy and parameter settings driven by throughput requirements. Besides bus topology exploration, comparison between packet-based bus and pipelined-based bus has also drawn attention. Pasricha *et al.* [18] compared the performance of a shared-link AXI and a single-layered AHB. Their comparison showed that up to 30% of bandwidth utilisation improvement can be achieved by AXI compared with AHB. They also investigated the impact of the transaction reordering buffer size in the memory controller. Lee *et al.* [19] built a crossbar AXI platform and a single-layered shared-link AHB platform to quantify the performance difference. They reported 40% communication efficiency improvement between AXI and AHB. Ruggiero *et al.* [20] studied the scalability of AHB, AXI and STBus under shared bus topology. Their result showed that AXI is far more scalable to the number of master devices than AHB. When the number of processor reaches eight, AXI can achieve 60% bandwidth utilisation improvement over AHB.

Comparison of bus connectivity configuration, such as shared-link, multi-layer (partial crossbar) and full matrix (crossbar), has also been interested as well. Lahiri *et al.* [21] proposed a design space exploration methodology and compared the performance between single-layer and multi-layer shared-link buses. Recently, Murali *et al.* [22] presented a bus communication architecture exploration method that finds the most power-efficient crossbar interconnect for a packet-based bus. They also briefly compared the performance and normalised cost ratio among shared-link, multi-layer and crossbar configurations.

Although the aforementioned works conducted analyses on communication architecture, the register slicing impact and multi-channel arbitration issues that arise with the features of packet-based bus have been overlooked. In addition, previous performance comparison of multi-channel AXI and single-layer share-link AHB may not be fair since AXI requires much more hardware cost.

3 Transfer modes

3.1 Normal

This mode is the basic transfer mode in an AXI bus with registered interface. In the first cycle of a transfer using normal mode, the initiator sets the valid signal high and sends it to the target. In the second cycle, the target receives the high valid signal and sets the ready signal high for one cycle in response. Once the initiator receives the high ready signal, the initiator resets the valid signal low and this transfer is completed. As a result, at least two cycles are needed to complete a transfer in an AXI bus with registered interface. Fig. 1 illustrates the transfer of

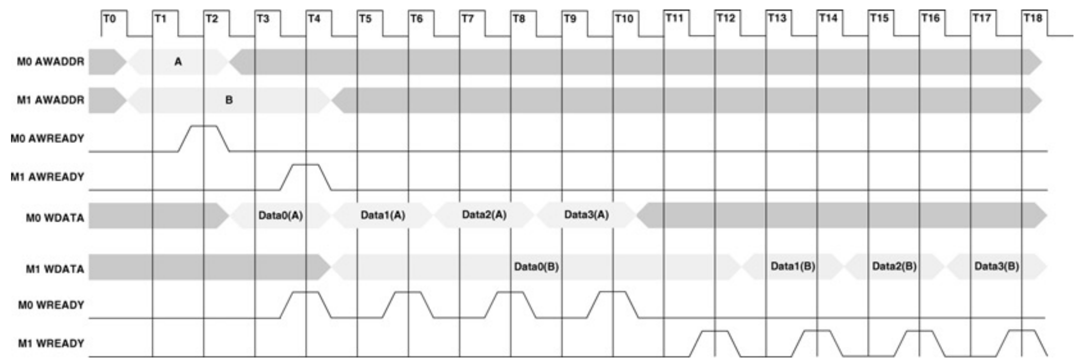

Figure 1 *Normal mode transfer example*

two normal transactions with a data burst length of four. It takes 16 bus cycles to complete the eight data transfer in the two transactions. This means 50% of the bus available bandwidth is wasted.

3.2 Interleaved mode

The interleaved mode [10, 20] hides transfer latency by allowing two transactions from different initiators to be transferred in an interleaved manner. Fig. 2 illustrates the transfer of the two transactions mentioned earlier using interleaved transfer mode. The one cycle latency introduced in the normal mode for request B is hidden by the transfer of request A. Similarly, the interleaved transfer mode can also be applied to data channels. As a result, transferring the data of the two transactions only takes nine cycles.

To support the interleaved mode, only the bus interconnect needs additional hardware. No additional hardware in device interface or modification on bus protocol is required. Hence, an AXI interconnect that supports the interleaved mode can be used with standard AXI device.

3.3 Proposed data locked mode

Although the interleaved mode can increase bandwidth utilisation when more than one initiator is using the bus, the interleaved mode cannot be enabled when only one standalone initiator is using the bus. To handle this, we proposed the data locked mode. In contrast to the locked transfer implemented in [11] that can only perform when the bus ownership is locked across consecutive transactions, the proposed data locked mode locks the ownership of the bus only within the period of burst data transfers. During the burst data transfer period, the ready signal is tied high and hence the handshaking process is bypassed. Unlike the interleaved mode, which can be applied to both request and data channels, the proposed data locked mode supports only burst data transfer.

Fig. 3 illustrates an example of two transactions using data locked mode to transfer data. Device M0 sends a data locked request A and device M1 sends a data locked request B. Once the bus interconnect accepts request A, the bus interconnect records the transaction ID of request A. When a data transfer with the matched ID appears in the data channel, the bus interconnect uses data locked mode to transfer the data continuously. For a transaction with a data burst of n, the data transfer latency is $n + 1$ cycles.

There are two approaches to signal the bus interconnect to use the data locked mode for a transaction. One uses ARLOCK/AWLOCK signal in the address channels to signal the bus of an incoming transaction using data locked transfer. However, doing so requires modifying the protocol definition of these signals and the bus interface. To avoid modifying the protocol, the other approach assigns the devices that can use the data locked mode in advance. The overhead of this approach is that the bus interconnect must provide mechanisms to configure the device transfer mode

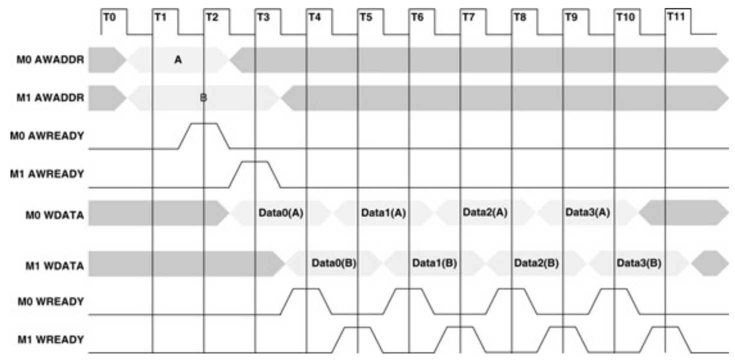

Figure 2 *Interleaved mode transfer example*

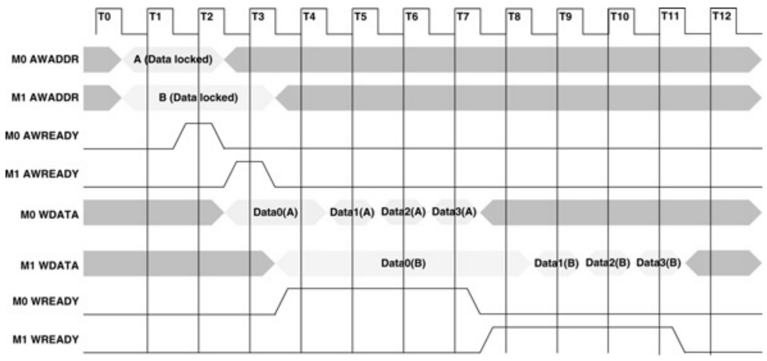

Figure 3 *Data locked mode transfer example*

mapping. Note that these two approaches can be used together without conflict.

To support the proposed data locked mode, the bus interconnect needs an additional buffer, called data locked mode buffer, to keep record of the transactions using the data locked mode. Each entry in the buffer stores one transaction ID. If all the entries in the data locked mode buffer are in use, no more transactions can be transferred using the data locked mode.

3.4 Proposed hybrid data locked mode

The hybrid data locked mode is proposed to allow additional data locked mode transaction requests to be transferred using the normal or interleaved mode when the data locked mode buffer is full. This allows more transactions to be available to the scheduler of the devices that support transaction scheduling. With the additional transactions, the scheduler of such devices may achieve better scheduling result.

However, only a limited number of additional transactions using the data locked mode can be transferred using the normal or interleaved mode. This avoids bandwidth-hungry devices from occupying the bus with too many transactions. A hybrid mode counter is included to count the number of additional transactions transferred. If the counter value reaches the preset threshold, no more data locked mode transactions can be transferred using the normal or interleaved mode until the data locked mode buffer becomes not full again. Once the data locked mode buffer is not full, the hybrid mode counter is reset.

4 Channel-independent arbitration framework

With the introduction of the multi-channel architecture and the proposed transfer modes, traditional arbitration framework that was based on single-channel architecture must be revamped. Therefore, we propose an arbitration framework that supports different arbitration flows for address channels and data channels. In contrast to existing works that used unified arbitration, each independent channel in the proposed arbitration framework is allowed to have its own arbitration policy. This framework allows different arbitration policies to be combined together in a simple plug-and-play manner.

Fig. 4*a* illustrates the arbitration flow for address channels. Upon receiving multiple data locked mode transaction requests from different initiators, the arbitration flow first checks whether the data locked mode buffer is full or not. If there is an available empty entry in the data locked mode buffer, the data locked mode transaction requests are arbitrated according to the arbitration policy adopted for the address channel. If the data locked mode buffer is full and the hybrid mode counter has not reached the threshold, all data locked mode transaction requests are treated as normal and interleaved mode transaction requests. As a result, all transaction requests are arbitrated together according to the adopted arbitration policy. On the other hand, if the hybrid mode counter has already reached the threshold, only the original normal and interleaved mode transaction requests are arbitrated. This arbitration flow gives higher priority to data locked mode transactions than normal or interleaved mode transactions.

Fig. 4*b* illustrates the arbitration flow for data channels. The arbitration flow first checks if there is already a transaction transferring data using the data locked mode. If there is already a transaction transferring data using the data locked mode, no other transaction would be granted. If no transaction is transferring, data locked mode transactions would be arbitrated according to the arbitration policy adopted by the data channel. If there is no data locked mode transaction requesting to transfer data, normal and interleaved mode transactions are arbitrated.

The reason for giving higher priority to the transactions using the data locked mode is that these transactions are often latency sensitive. To minimise the latency of these transactions, the transactions must be transferred using the data locked mode and given the highest arbitration priority.

5 Performance analysis setup

To properly evaluate the performance of the proposed transfer modes and arbitration framework on a shared-link AXI bus, we built a high-level model of a simplified multi-core

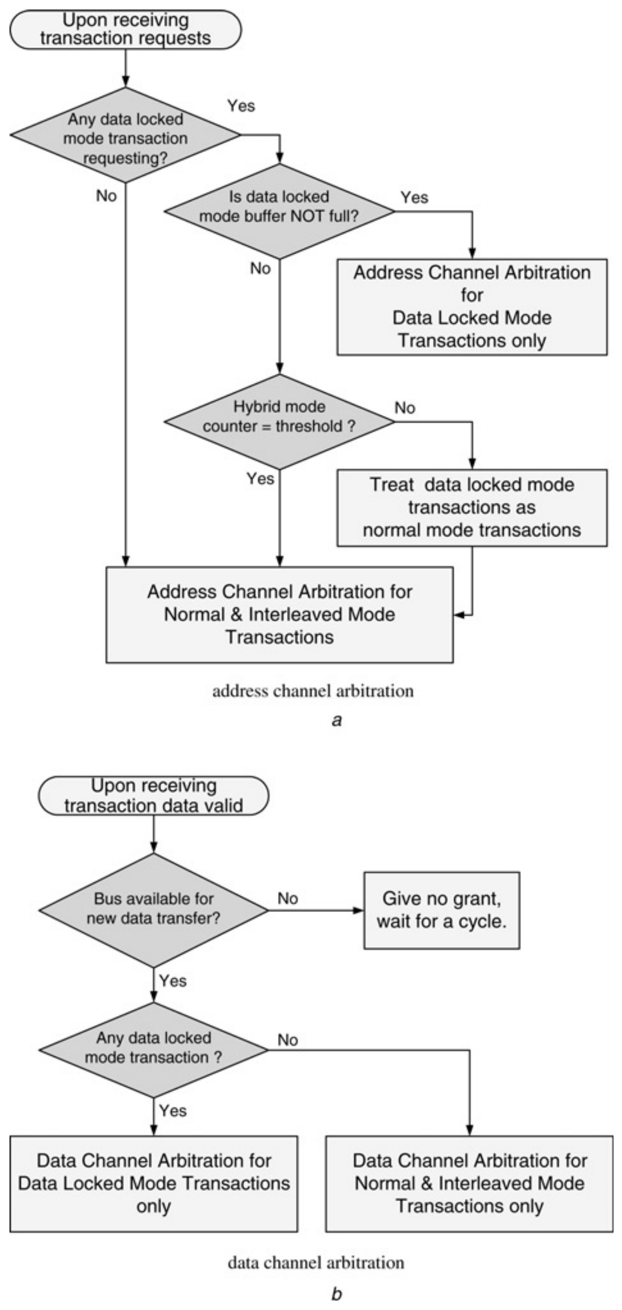
Upon receiving transaction requests
Any data locked mode transaction requesting?
Yes
No
Is data locked mode buffer NOT full?
Yes
No
Address Channel Arbitration for Data Locked Mode Transactions only
Hybrid mode counter = threshold ?
No
Yes
Treat data locked mode transactions as normal mode transactions
Address Channel Arbitration for Normal & Interleaved Mode Transactions
address channel arbitration
a
Upon receiving transaction data valid
Bus available for new data transfer?
No
Give no grant, wait for a cycle.
Yes
Any data locked mode transaction ?
No
Yes
Data Channel Arbitration for Data Locked Mode Transactions only
Data Channel Arbitration for Normal & Interleaved Mode Transactions only
data channel arbitration
b

Figure 4 *Arbitration framework for a share-link AXI bus*

a Address channel arbitration
b Data channel arbitration

platform system using SystemC [23]. The simulation accuracy of this model depends on modelling methodology, platform architecture authenticity and application traffics accuracy. The bus and components in the platform were modelled using transaction-level and behaviour-level modelling method, respectively. Transaction-level modelling uses a transaction instead of a cycle as the basic simulation unit. Since a transaction takes a fixed number of cycles to complete in each channel, transaction-level modelling ensures bus cycle accuracy in our simulations. More details on transaction-level modelling can be found in [15]. To pursue platform architecture authenticity, the multi-core platform model was built based on a real multi-core platform [24]. The real platform has been verified with a portable media player and smartphone applications. This ensures the simulation result from our platform model to be practical. The application traffics were derived based on the behaviour and algorithm of the platform components to ensure traffics accuracy. The details of the platform architecture and bus traffics are provided in the following subsections.

5.1 Multimedia platform architecture

Fig. 5 illustrates the target platform from the system bus point of view. Note that when the platform is used for AHB simulation, the bus interconnect is replaced with a five-layer AHB-lite interconnect with each master port having one dedicated AHB-lite bus. Since we only focus on the transaction behaviour on the bus, the devices are modelled to only exhibit transaction behaviour and pattern. However, the CPU does generate transactions related to interrupt service routines (ISR) upon receiving an interrupt request (IRQ). In addition, the DMA controller is also programmed to carry out different data moving tasks to mimic the behaviour of its real

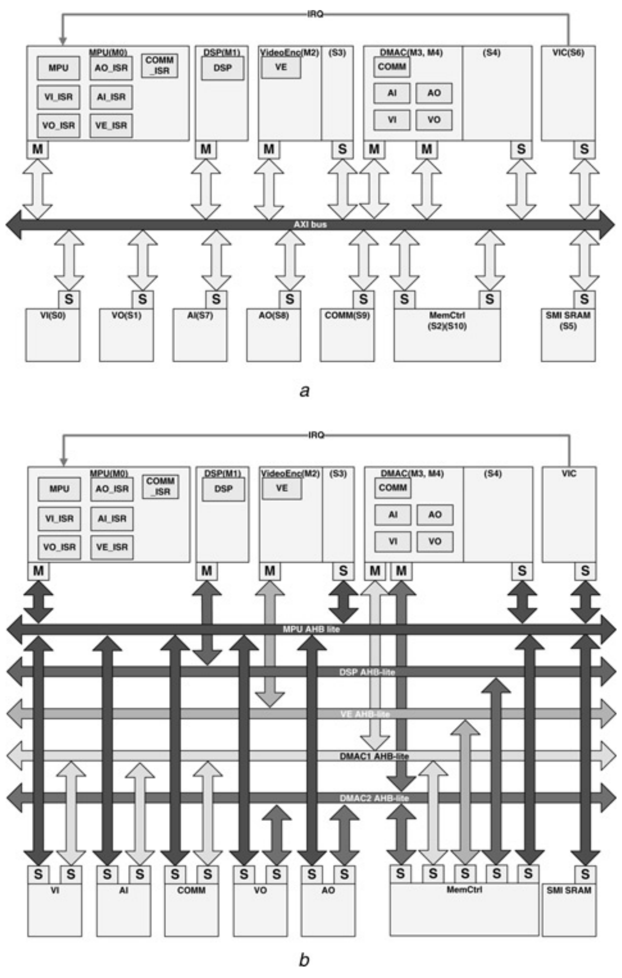

Figure 5 *Block diagram of the target platform*

a Using AXI
b Using AHB-lite

counterpart. Including such more detailed behaviour enables us to include the inter-task dependency between devices. Note that the memory controller has two slave ports to allow more transactions to be seen by the scheduler of the memory controller. Among all the devices, the memory controller is the only one with access latency ranging from 0 to 16 cycles.

The AXI bus is clocked at 40 MHz with both the address and data widths being 32-bit wide. This would yield an ideal total bandwidth of 320 MB/s with the read and write bandwidths being 160 MB/s each.

5.2 Video phone scenario

We have selected the video phone application for analysis because it covers a variety of devices and traffics that are common in most multimedia consumer electronic products. The bandwidth requirement of the video phone application is heavier than other applications such as portable media player, video recording, MP3 player and regular phone service. This heavy bandwidth requirement also makes the video phone application a perfect application to test the performance limit of a bus.

In the video phone application, the system must deliver both audio and video communication at the same time. The system supports 44.1 kHz stereo audio capture/output and audio compression/decompression. As to video, the system provides VGA-sized video capture, compression, decompression and display with a target frame rate of 30 fps. Table 1 lists the task description, bandwidth requirement and task completion time constraint of each master device in the video phone application. Although more devices may be included in a system, the bus traffic is usually dominated by the master devices listed in Table 1. The total bus bandwidth requirement is 247.8 MB/s, which occupies 77.5% of the 320 MB/s available bus bandwidth. If the bus can achieve a bandwidth utilisation higher than 77.5%, all the system tasks are more likely to complete within the specified timing constraints.

6 Simulation result

6.1 Evaluation metrics

The definition and physical meaning of the evaluation metrics are explained as follows:

(A) Bandwidth utilisation (BWU)

The BWU is defined as the percentage of available ideal bus bandwidth being used to actually transfer data, that is,

$$\mathrm{BWU} = \frac{B_{\mathrm{used}}}{B_{\mathrm{ideal}}} \times 100\% \tag{1}$$

Table 1 Port task description and bandwidth requirement

Master port	Task	Required read BW (MB/s)	Required write BW (MB/s)	Total required BW (MB/s)
MPU	audio codec	1.467	1.467	2.934
	OS routine	0.001	0.001	0.002
	Total ISR	0.172	0.493	2.935
	total bandwidth requirement	1.640	1.961	3.601
DSP	video decode	14.836	42.473	57.309
Video encoder	video encode	59.927	14.255	74.182
DMAC0	video in to MEM	27.927	27.927	55.855
	audio in to MEM	0.176	0.176	0.353
	3G communication	0.132	0.132	0.265
	total bandwidth requirement	28.236	28.236	56.472
DMAC1	MEM to video out	27.927	27.927	55.855
	MEM to audio out	0.176	0.176	0.353
	total bandwidth requirement	28.104	28.104	56.208
system total bandwidth requirement		132.743	115.028	247.771

where B_{used} and B_{ideal} are the actually used bandwidth and available ideal bandwidth, respectively. A higher BWU implies that more data can be transferred within a period of time. It also implies shorter effective transaction latency from the system's point of view.

(B) Transaction latency

The transaction latency we used is defined as the average of read and write transaction latencies. The latency of a read or write transaction is measured from the time a transaction request is sent from a master till the time the read data or write response is returned to the master. The average transaction latency, denoted as TL, can be defined as

$$\text{TL} = \frac{\sum \text{TL}_{\text{read}} + \sum \text{TL}_{\text{write}}}{N_{\text{read}} + N_{\text{write}}} \tag{2}$$

where $\sum \text{TL}_{\text{read}}$ and $\sum \text{TL}_{\text{write}}$ are the sums of all read and write transaction latencies, respectively. N_{read} and N_{write} are the total number of read and write transactions, respectively. In contrast to the bandwidth, which increases as more data can be transferred, the transaction latency may remain the same even if the BWU has been increased.

(C) System task completion time

The system task completion time is defined as the time when all tasks in the video phone application have been completed. We believe it is crucial to minimise the system task completion time so that the task-level timing constraint can be met. In the video phone application, all tasks must be done within 33 ms, otherwise we say the system violates the real-time constraint.

6.2 AXI interface buffer size and bus arbitration impact

The effect of the bus interface buffer size and the combination of arbitration policies are investigated first. The investigated buffer sizes are 1, 2, 4, 8 and 16. Each entry keeps the record of a transaction. Table 2 lists the abbreviations of the investigated arbitration policy combinations. The weighting parameter, which is slots in the TDMA and tickets in the Lottery, are tuned to match the bandwidth requirement of the video phone application. Table 3 lists the detailed weight parameter of each channel. Since write response channel does not require high bandwidth, round-robin arbitration is selected for write response channel due to its fairness. Note that in this experiment, only the normal and the interleaved modes are used.

Fig. 6 shows the BWU, average transaction latency and completion time, respectively. In general, the BWU increased as the interface buffer size increased. However, the BWU stopped increasing when the buffer size is greater than 8 because of the required bandwidth limit. The average transaction latency is also proportional to the interface buffer size. This is because with a larger buffer, a transaction would spend more time pending in the interface buffer before this transaction finishes transferring the data. In contrast, the completion time decreases as the buffer size increases.

It is interesting that the transaction latency did not reflect the result in BWU. This is because the outstanding and out-of-order transfer capabilities, which are related to the buffer size, allow multiple transactions to be transferred on the bus in an overlapped manner. As a result, the latency of a transaction becomes longer, but the transfers on the bus can be arranged in a more compact way. This is one of the characteristics in a packet-based bus that is different from a traditional pipeline-based bus.

Table 2 Combinations of arbitration policies

Arbitration policy of channels			
Combination name	Address channel	Data channel	Write response channel
FF	fixed priority	fixed priority	round-Robin
FT	fixed priority	TDMA	round-Robin
FR	fixed priority	round-Robin	round-Robin
FL	fixed priority	lottery	round-Robin
TF	TDMA	fixed priority	round-Robin
TT	TDMA	TDMA	round-Robin
TR	TDMA	round-Robin	round-Robin
TL	TDMA	lottery	round-Robin
RF	round-Robin	fixed priority	round-Robin
RT	round-Robin	TDMA	round-Robin
RR	round-Robin	round-Robin	round-Robin
RL	round-Robin	lottery	round-Robin
LF	lottery	fixed priority	round-Robin
LT	lottery	TDMA	round-Robin
LR	lottery	round-Robin	round-Robin
LL	lottery	lottery	round-Robin

Table 3 Weight allocation of TDMA and Lottery arbitration schemes

Channel	Slots/tickets of each initiator port				
read address	CPU	DSP	video enc.	DMAC0	DMAC1
	4	8	24	24	24
write address	CPU	DSP	video enc.	DMAC0	DMAC1
	4	24	8	24	24
read data	video in	mem. ctrl. 0	mem. ctrl. 1	others	
	8	24	16	4	
write data	CPU	DSP	video enc.	DMAC0	DMAC1
	4	24	8	24	24

For the arbitration policy, the impact on the performance was not significant when the buffer size is 1 and 2. After the buffer size becomes larger than 2, the combinations with data channels using fixed-priority, such as FF, TF, RF and LF, usually achieve lower BWU than other combinations. On the other hand, the combinations with data channels using TDMA, such as FT, TT, RT and LT, usually achieve the highest BWU. Similar trend is also observed in the execution time. However, this trend is weak in the transaction latency, which doest not reflect the result of the BWU.

The completion time comparison shows that when TDMA is used for data channel, the 33 ms timing constraint can be satisfied in buffer sizes 8 and 16. On the other hand, no arbitration policy combination satisfied the 33 ms timing constraint with the buffer size smaller than 8.

In summary, bandwidth and completion time improved sub-linearly as the interface buffer size increased. However, the buffer size increase also increased the transaction latency near linearly. When a smaller interface buffer is used, the arbitration combination had less impact on performance. On the other hand, for a larger buffer size, the best arbitration combination could yield up to 23.3% performance gain over the worst arbitration combination. The result suggests that using a fair arbitration policy on data channels should be more promising. For address channels, a simpler arbitration policy is good enough because address channel arbitration had less impact on bus performance.

6.3 Task access setting impact

This subsection finds out how the hybrid data locked mode should be used and shows the performance impact delivered by using the hybrid data locked mode. A task access setting defines how the transfer modes should be used by the devices in a system. Table 4 lists the four settings investigated here. The memory device is singled out because it is the only device with non-zero access latency. All other devices have zero access latency and hence are treated the same. Note that the results here are the average over all 16 arbitration policy combinations. The buffer size of data locked mode and hybrid counterthreshold are both set to one.

Fig. 7 shows the average BWUs, average transaction latency and completion time of different task access settings. In general, HN setting achieved the highest BWU among all the task access settings except in buffer size 16. This is because when the buffer size is 16, the BWU of NN, HN and NH settings is already high enough to handle all the data transfer. If the buffer size is small, the use of the hybrid data locked mode could reduce the completion time by up to 26.8% compared with NN. Although HN achieved the highest performance in most cases, HH achieved the highest BWU in buffer size 1 because no out-of-order transfer can be carried out with the interface buffer having only one entry. Consequently, HH took shorter time to transfer a transaction and less bandwidth would be wasted. In contrast to HH's highest BWU in buffer size 1, HH setting achieved the lowest bandwidth utilisation when the buffer size is larger than 2. This is because HH had less opportunity to enable interleaved transfer mode on normal transactions when the interface buffer size increases. Unlike the result in the BWU, HH setting achieved the shortest average transaction latency among the four settings. The transaction latency of HH did not increase significantly as the buffer size increased. On the other hand, NN had the expected longest average transaction latency. The trend in completion time matches the trend in BWU in general. HN setting achieved the shortest completion time and met the 33 ms timing constraint with a buffer size larger than 4.

In summary, HN was the best task access setting in most cases in terms of BWU and completion time. This suggests that the hybrid data locked mode would best be used by long access latency devices, but not by zero access latency devices. From the transaction latency perspective, HH achieved the shortest transaction latency. This suggests that processors or devices that require short latency should use the hybrid mode.

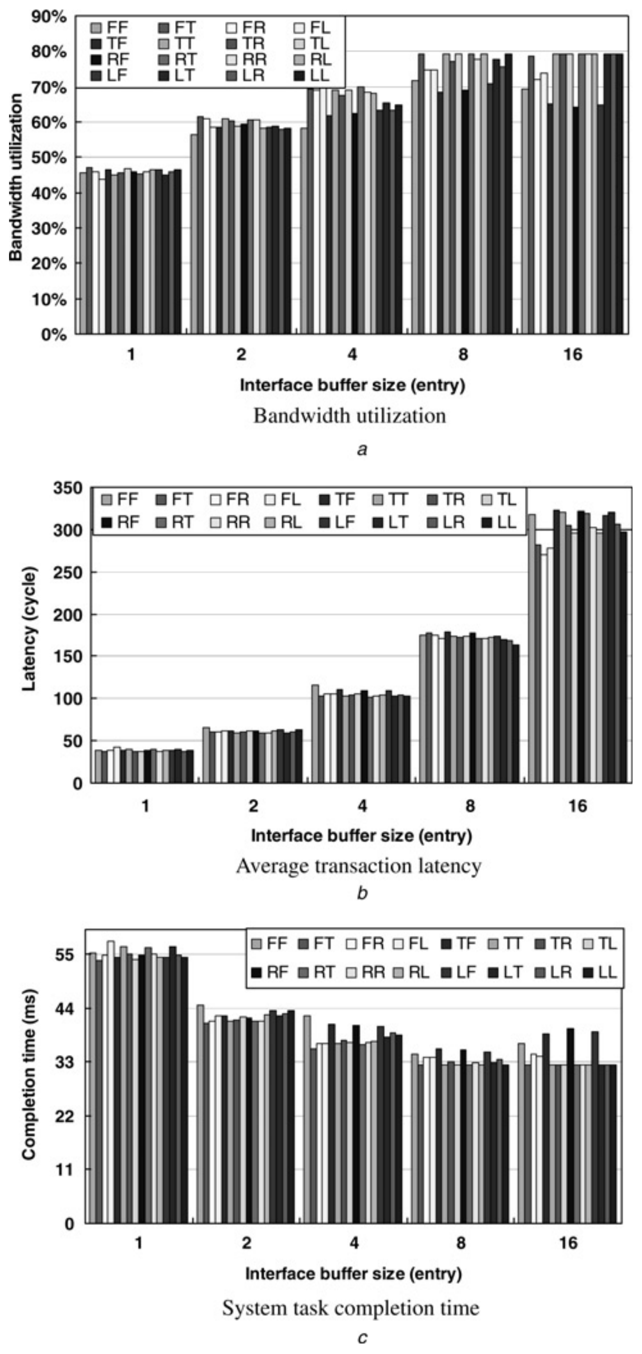

Figure 6 *Performance of different interface buffer size and arbitration policy combinations*

a Bandwidth utilisation
b Average transaction latency
c System task completion time

6.4 Single-layer shared-link AXI against five-layer AHB-lite

This subsection compares the performance between a share-link five-channel AXI interconnect and a cost equivalent five-layer AHB-lite interconnect. The five-layer AHB-lite interconnect is capable of providing a maximum bandwidth of 800 MB/s. We used two task access settings for the AXI case, one is NN setting and the other is HN setting.

Table 4 Task access settings

Settings	Memory access tasks	Other tasks
NN	normal and interleaved	normal and interleaved
HN	hybrid	normal and interleaved
NH	normal and interleaved	hybrid
HH	hybrid	hybrid

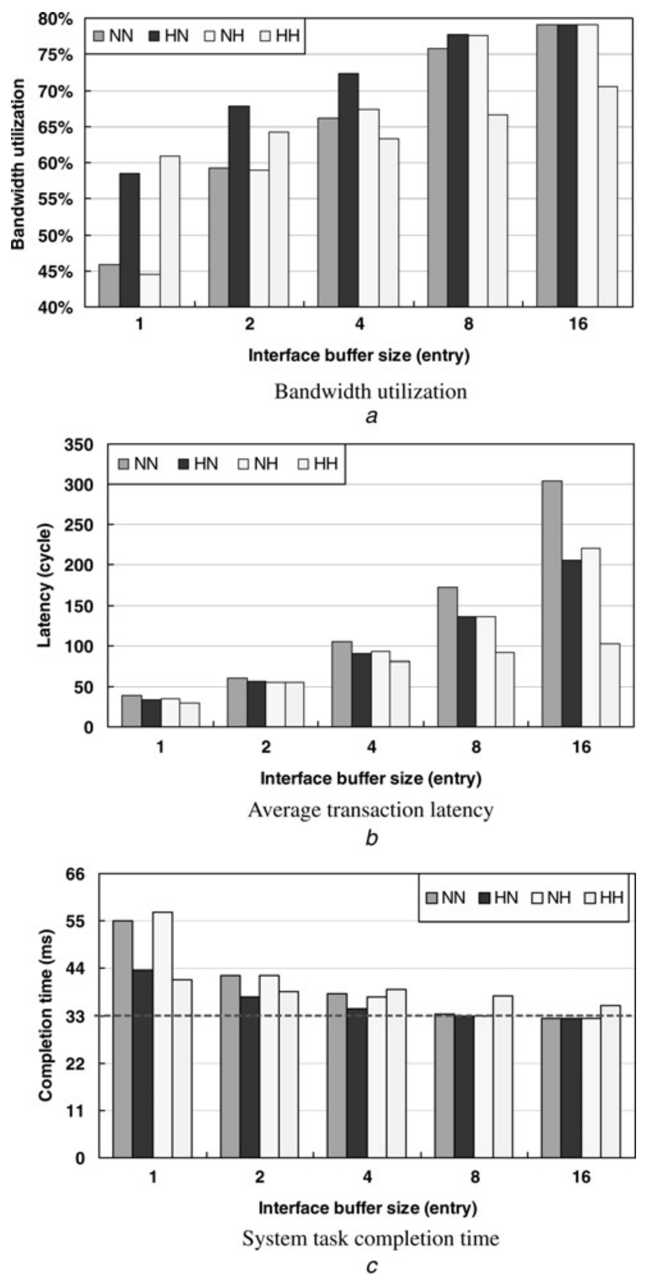

Figure 7 *Performance of different task access settings and interface buffer size*

a Bandwidth utilisation
b Average transaction latency
c System task completion time

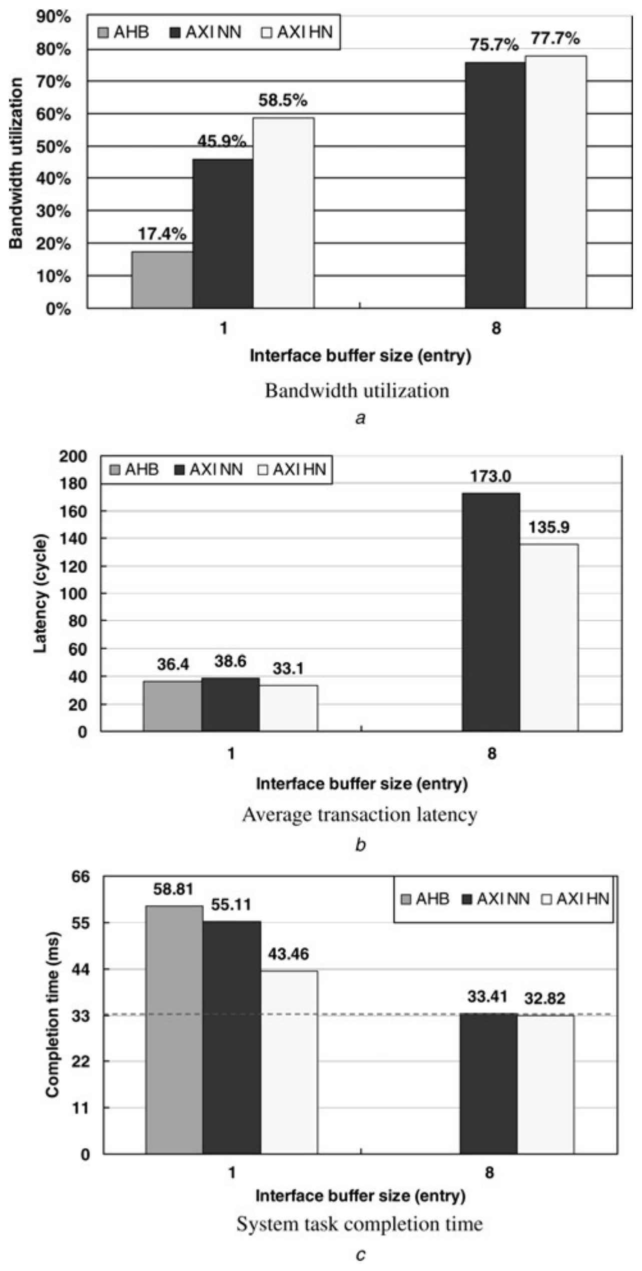

Figure 8 *Performance of five-layer AHB-lite and single-layer shared-link AXI*

a Bandwidth utilisation
b Average transaction latency
c System task completion time

The interface buffer sizes we investigated are 1 and 8. Since the interface buffer size has no effect in AHB, only the result of buffer size 1 is available for AHB.

Fig. 8 compares the BWU, average transaction latency and completion time of the AHB and AXI platforms. The BWU of shared-link AXI is significantly higher than that of AHB. If the interface buffer size is 8, the BWU of AXI outperformed AHB by at least 58.3%. However, AXI's transaction latency can reach up to 4.7 times of AHB's in buffer size 8. The completion time comparison shows that despite the long latency in AXI, the completion time in AXI reduced up to 44.2% when compared with AHB.

The result shows that a single share-link AXI outperforms a five-layer AHB-lite interconnect in the videophone case study. Given that the hardware cost of a five-layer AHB-lite interconnect is comparable to a shared-link AXI interconnect, using a shared-link AXI interconnect may be more efficient than using a multi-layer AHB interconnect.

7 Discussion and summary

The analysis in this work provides some insights for multimedia system design involving a shared-link AXI interconnect. If the buffer cost and transaction latency are not the primary concerns, system designers can consider using larger interface buffer to take the full advantage of out-of-order and outstanding transfer capabilities. However, some care must be taken in selecting a proper arbitration combination when using a channel-independent arbitration framework, especially when the interface buffer is large. The analysis showed that the arbitration combination could affect bus performance by up to 23.2%. Moreover, the arbitration combination that yields the best system performance may vary depending on the interface buffer size, task access setting and application traffic characteristic. In general, using a fair arbitration policy such as the TDMA is preferred for data channels. As to address channels, system designers can select a simpler arbitration policy to reduce the cost since the address channel arbitration has less impact on system performance. On the other hand, if the buffer cost and transaction latency do matter, system designers can use the hybrid data locked mode to achieve a performance similar to the case that uses only the interleaved mode, but with only half the interface buffer size. When the hybrid data locked mode is adopted for only long access latency devices, the simulation showed up to 21.1% completion time and 14.3% transaction latency reduction with respect to the setting without the hybrid data locked mode. With the short transaction latency, system designers can also consider adopting the hybrid data locked mode for latency-sensitive devices, such as CPUs, to reduce transaction latency. Although the analysis was conducted using AXI, we believe that the experience can be extended and applied to other shared-link packet-based bus as well.

8 References

[1] ARM Limited: 'AMBA 2 specification', Available at http://www.arm.com/products/solutions/amba2overview.html

[2] IBM Microelectronics: 'CoreConnect bus architecture', Available at http://www-306.ibm.com/chips/products/coreconnect/

[3] OpenCore: 'SoC interconnect: wishbone', Available at http://www.opencores.org/projects.cgi/web/wishbone/wishbone

[4] ARM Limited: 'Multi-layer AHB overview' May 2004

[5] POLETTI F., BERTOZZI D., BENINI L., BOGLIOLO A.: 'Performance snalysis of arbitration polices for SoC communication architectures', *Des. Autom. Embedded Syst.*, 2003, **8**, (2–3), pp. 189–210

[6] LAHIR K., RAGHUNATHAN A., LAKSHMINARAYANA G.: 'LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs'. In Proc. 38th Design Automation Conf., June 2001, pp. 15–20

[7] ARM Limited: 'AXI protocol', Available at http://www.arm.com/products/solutions/AMBA3AXI.html

[8] Open Core Protocol International Partnership (OCP-IP): 'Open core protocol', Available at http://www.ocpip.org/

[9] ST Microelectronics: 'STBus interconnect,' Available at http://www.st.com/stonline/products/technologies/soc/stbus.htm

[10] Arm Limited: 'PrimeCell AXI interconnect (PL300) technical reference manual', October 2005

[11] Synopsis Inc.: 'DesignWare IP solutions for AMBA interconnect'at Available at http://www.synopsys.com/products/designware/amba_solutions.html

[12] PASRICHA S., DUTT N., BEN-ROMDHANE M.: 'Automated throughput driven synthesis of bus-based communication architectures'. In Proc. ASPDAC, February 2005

[13] RICHTER K., JERSAK M., ERNST R.: 'A formal approach to MpSoC performance verification', *IEEE Comput.*, 2003, **36**, pp. 60–67

[14] MADL G., PASRICHA S., ZHU Q., BATHEN L., DUTT N.: 'Formal performance evaluation of AMBA-based system-on-chip designs'. In Proc. 6th ACM and IEEE Int. Conf. Embedded Software, 2006, pp. 311–320

[15] CAI L., GAJSKI D.: 'Transaction level modeling: an overview'. In Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign, October 2003, pp. 19–24

[16] PASRICHA S., DUTT N., BEN-ROMDHANCE M.: 'Fast exploration of bus-based on-chip communication architectures'. In Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and System Synthesis, September 2004, pp. 242–247

[17] PASRICHA S., DUTT N., BEN-ROMDHANE M.: 'Constraint-driven bus matrix synthesis for MPSoC'. In Proc. ASPDAC, January 2006, pp. 30–35

[18] PASRICHA S., DUTT N., BEN-ROMDHANE M.: 'High level design space exploration of shared bus communication architectures', CECS Technical Report 04-06, 2004

[19] LEE S., LEE C., LEE H.-J.: 'A new multi-channel on-chip-bus architecture for system-on-chips'. In Proc. IEEE Int. SOC Conf., September 2004, pp. 305–308

[20] RUGGIERO M., ANGIOLINI F., POLETTI F., BERTOZZI D.: 'Scalability analysis of evolving SoC interconnect protocols'. In Proc. Int. Symp. System-on-Chip, November 2004, pp. 169–172

[21] LAHIRI K., RAGHUNATHAN A., DEY S.: 'Design space exploration for optimizing on-chip communication architectures,', *IEEE Trans. Comput.r-aided Des. Integr. Circuits Syst.*, 2004, **23**, (6), pp. 952–961

[22] MURALI S., BENINI L., DE MICHELI G.: 'An application-specific design methodology for on-chip crossbar generation,', *IEEE Trans. Comput.-aided Des. Inegr. Circuits syst.*, 2007, **26**, (7), pp. 1283–1296

[23] Open SystemC Initiative (OSCI): 'SystemC', Available at http://systemc.org

[24] LIN T.-J., LIU C.-N., TSENG S.-Y., CHU Y.-H., WU A.-Y.: 'Overview of ITRI PAC project–from VLIW DSP processor to multicore computing platform'. In Proc. IEEE Int. Symp. VLSI Des., Automation, and Test, April 2008, pp. 188–191