OVERVIEW OF BUS-BASED SYSTEM-ON-CHIP INTERCONNECTIONS

Erno Salminen, Vesa Lahtinen, Kimmo Kuusilinna, and Timo Hämäläinen
Tampere University of Technology, Institute of Digital and Computer Systems
P.O. Box 553, FIN-33101 Tampere, Finland
Tel: +358-3-3115 4540, Fax: +358-3-3115 4575
E-mail: erno.salminen@tut.fi

ABSTRACT

This paper introduces the basic properties, such as structure, transfer properties and arbitration of bus-based interconnections for System-on-Chip (SoC) designs. The overview shows that contemporary SoC buses differ only in minor details. As a result, practically every studied interconnection method could rather easily conform to a common interface. Such interface would enhance design re-use and make system design easier. However, due to their similarity, the choice between buses is not a straightforward task.

1. INTRODUCTION

Interconnecting functional units is a complex task in designing a System-on-Chip (SoC). This is due to two facts. Firstly, despite the standardization work, there is no general solution for interconnection that is suitable for arbitrary applications. Secondly, communication is becoming more and more important factor in determing the overall performance of the system. This paper introduces the basic properties of bus-based interconnections and how they are used in available and proposed on-chip buses.

Virtual Socket Interface Alliance (VSIA) aims to develop standards for integration and re-use of IP blocks [4]. As a part of the stardardization work it has developed a common interface, called Virtual Component Interface (VCI) [13] that all blocks can utilize. It does not make statements of the actual interconnection at physical level. Separating computation from communication gives the possibility to design the system without the knowledge of the underlying communication architecture as long as it uses a standard interface to the interconnection. When the bus manufacturers start to support a common interface it is easier to re-use existing functional blocks.

The goal for this paper is to show the common features found in several contemporary on-chip buses. The buses considered here are AMBA [5], Core Connect [6][7], Core Frame [8], HIBI [9][10], Lotterybus [11], Marble [12], VCI [13], PI bus [14], Silicon Backplane [15] and Wishbone [17]. Sonics Inc., designer of Silicon Backplane, has extended the VCI definition to Open Core Protocol (OCP) [16] which is used in Silicon Backplane. Lotterybus and VCI do not actually specify a bus but an arbitration method and a common bus interface, respectively.

The paper is organized as follows. First, structure and clocking of the buses are compared. Second, the transfer types and arbitration methods are listed. After that the reconfiguration and performance metrics are discussed shortly. Finally, Section 3. concludes the overview.

2. CLASSIFYING BUS-BASED INTERCONNECTIONS

To classify buses, we consider three main features that are structure, transfer properties and arbitration. These are refined to features explained in the following subsections and summarized in Table 1. The supported properties are marked with '*x*', unsupported with '-', and when there is no information available they are marked with '*n/a*'. The exceptions are listed below the table.

The properties were collected from public specifications and publications without carrying out practical experimentations. For this reason, comparison between the performances of the buses cannot be done. However, it is likely, that due to license and other legal agreements, it is not possible to publish any performance comparison even if test cases had been implemented.

0-7803-7448-7/02/$17.00 ©2002 IEEE

Table 1. Summary of Bus Properties

	Structure						Transfers				Arbitration			
Name	Hierarchical	Uni-/ Bidirectional	Shared / Point-to-point connections	Synchronous / Asynchronous	Multiple clock domains	Test structures	Handshaking	Split transfer	Pipelined transfer	Broadcast	App. specific /1- / 2-level arbitration	Pipelined arbitration	Centralized / Distributed arbitration	Dynamic configuration
AMBA	x	1)	S	S	n/a	x	x	2)	x	n/a	as	x	C	n/a
Core Connect	x	U	3)	S	n/a	x	x	n/a	x	n/a	1	x	C	x
Core Frame	x	U	P	S	x	n/a	4)	n/a	n/a	n/a	as	x	C	n/a
HIBI	-	B	S	S	x	-	-	x	-	-	2	x	D	x
Lotterybus	x	n/a	S	S	n/a	n/a	n/a	n/a	n/a	n/a	1	x	C	x
Marble	n/a	B	S	A	x	x	x	x	x	x	1	x	C	n/a
PCI	n/a	U	n/a	S	x	-	x	x	x	-	as	-	C	n/a
PI Bus	n/a	B	S	S	n/a	n/a	x	-	n/a	n/a	as	n/a	C	n/a
Silicon Backplane	n/a	n/a	5)	S	x	x	x	n/a	x	x	2	x	D	x
Wishbone	n/a	6)	S	S	x	-	x	n/a	-	n/a	as	n/a	C	n/a

Table exceptions
1) AHB and APB unidirectional, ASB bidirectional
2) AHB and ASB allow split transfers
3) Data lines shared, control lines point-to-point ring
4) Palmbus uses handshaking, Mbus does not
5) Data lines shared, control lines point-to-point
6) Both possible, unidirectional recommended

2.1. Structure

In hierachical structures, two or more buses are connected via bridges. Depending on implementation the buses can transfer data independently as long as they do not refer resources residing on other buses. Another possibility is that one bus acts as a slave device. In this scenario, the slave bus remains in idle state until it is accessed. AMBA uses this latter method to decrease power consumption while Core Connect and Core Frame have two independent buses connected with a bridge. In addition, Lotterybus defines an arbitration method that does not presume any fixed communication topology.

Another design choice is whether to use uni- or bidirectional signal lines. Bidirectional lines save routing resources but require tri-state buffers that are not usually recommended in ASIC design due to difficulties with their control and testing. It is also argued that unidirectional buses are faster than bidirectional. Most studied buses use unidirectional buses except HIBI, Marble and PI bus. AMBA and Wishbone support both possibilities.

Traditionally, the basic definition for bus is that it uses shared interconnection lines. This is true for all except Core Frame that has a shared central memory. In Core Frame, all units have a point-to-point connection to a memory controller. Signals used for arbitration and decoding are not taken into account, because *request* and *grant* used in centrally arbitrated systems are commonly implemented as unidirectional and point-to-point signals.

2.2. Clocking

If all the data transfers are clocked with a single clock the communication system is *synchronous*. In such system, it is crucial to keep the clock skew in acceptable limits. *Asynchronous* systems overcome this problem because every single data transfer uses handshaking. It should be noted that handshaking is also used in many synchronous systems for a data transaction consisting of several data transfers. Locally synchronous, globally asynchronous approach allows integration of agents that use different clock frequencies. This way the power consumption can be reduced by clocking agents with the lowest possible frequency irrespective of each other.

One key factor to usability of an interconnection is its ability to be suited for different applications. Unfortunately, such properties are almost impossible to measure. Only guidelines may be stated, such as versatile support for configuration, either dynaminally or during the system design. A support for agents with different clock frequencies and data widths are the two most basic properties. In addition, structures for manufacturing test are becoming more important.

Nearly all on-chip buses use a global clock to synchronize transfers. Earlier, when devices were constructed from multiple chips or boards the delays could not be predicted. This resulted in asynchronous bus protocols allowing slow devices to be used in conjunction with faster ones. Marble is the only one using asynchronous timing. However, some synchronous buses also allow the integration of blocks using different clock domains as shown in Table 1.

2.3. Transfers

Support for multiple clocks is provided with some sort of handshaking between master and slave devices. In asynchronous bus, such as Marble, all transfers use handshaking. In synchronous systems, the handshaking does not add delay into transfers when the active units are fast enough. When slower devices take part in transfers, they can use wait signals to stretch the transfer for several clock cycles. Nearly all studied buses use handshaking. HIBI is the only one that explicitly denies any handshaking in bus transfers.

In read operations, it is very likely that the slave device cannot provide the data immediately. This situation can be handled with handshaking but more sophisticated method is to use *split transfer*. In split transfer, the read operation is split into two write operations and the bus is released in the middle. Thus the bus is not reserved unintentionally, i.e. while waiting the slave device to respond. Four buses out of ten studied support split transfers.

In message passing systems and during dynamic reconfiguration, it is beneficial to transfer the same data to every agent. By using *broadcast* the data is transferred simultaneously to all target agents. However, such operation is found only in Marble and Silicon Backplane. HIBI has a broadcast operation for dynamic reconfiguration but not for data.

Some buses use pipelined transfers to obtain higher clock frequency. Pipelined transfers send the address on the first cycle to leave enough time for address decoding and transfer data on following cycles. It is possible that the address of the next transfer is interleaved with the last data of the previous transfer.

2.4. Arbitration

A shared resource, such as a bus or a shared memory, suffers from contention when multiple agents want to access it simultaneously. *Arbitration* is a way to define the current master, i.e. the owner. Traditionally, the arbitration is implemented in centralized fashion where every agent requests a central arbiter for a permission to access the shared resource. The arbiter uses some algorithm to grant the ownership to one of the requesting agents. Three often used algorithms are *round-robin*, *fixed priority*, and *time division multiple access* (TDMA) [1][2].

It is also possible to implement the arbitration in a distributed fashion where every agent is aware of the state of the shared resource and can decide the correct time to access the bus by itself. This method does not need dedicated request and grant signals. It is shown to minimize the latencies in arbitration, but requires more hardware [9]. All studied buses utilize run-time arbitration. However, in large networks compile time, i.e. static, scheduling may result in smaller implementation compared to the dynamic scheduling [3].

Arbitration can be pipelined with data transfers. That way, the next owner is determined beforehand and it can access the shared resource without a delay when it is released. However, some implementations require a clock cycle or two when the bus master is changed. In addition, some parts of the arbitration can be determined at compile time. In TDMA, the time is divided into time frames which are repeated. Inside the frame, some agents are granted time slots, i.e. the ownership is granted to them during certain clock cycles. Initial time slot allocation is done at compile time. This way real-time requirements can be met by guaranteeing bandwidth (throughput), i.e. an agreed amount of bus time to an agent. If the owner does not have anything to send during its time slot it can release the shared resource. A 2-level hybrid arbitration has two layers. The first is the TDMA and all the time that is left over is arbitrated using the second level dynamic arbitration, e.g. round-robin.

There is not much variation in arbitration methods. All buses but HIBI and Silicon Backplane use centralized arbitration. Both HIBI and Silicon Backplane use a hybrid 2-level arbitration which is distributed among agents, i.e. there is no central arbiter unit. Also the address decoding is distributed in them.

Usually, the algorithm for 1-level arbitration is either fixed priority of round-robin. AMBA, Core Frame, VCI, PI bus, and Wishbone do not specify the arbitration algorithm, only its interface. By implementing a new, more complicated arbitration algorithm the system can be tuned to better suite the appplication at the cost of increased design time. In Lotterybus, the master is chosen probabilistically according to 'lottery tickets' assigned for them. The central arbiter generates a pseudo random number that matches one ticket number. Agent having most tickets is most likely the next bus master.

2.5. Reconfiguration

Although the major part of the communication of an embedded system can be usually estimated at compile time, the communication varies dynamically and the static arbitration cannot fulfill the communication needs.

Therefore, it is reasonable to make run-time configuration possible. The configuration defines the priorities, TDMA parameters, and longest reservation time of the shared resource. This way the communication can be tuned to better meet the current requirements of the application. Usually the start times of the transfers creep away from the estimated times in TDMA-based systems. Fast synchronization is an efficient method for correcting

the timing before the error gets too large and the benefit of the TDMA arbitration is lost. However, the dynamic reconfiguration is rather complex to design and requires more hardware resources. There is, again, a trade-off between silicon area and performance. Four buses support dynamic reconfiguration.

2.6. Performance Metrics

The most often used metric for performance is *throughput* which defines how much data is transmitted in a time unit. The throughput values are usually given in megabytes per second (MB/s). Unfortunately, the announced values might be theoretical peak values, which are extremely hard to reach in real applications.

Another important metric is the *latency* of a transfer. It measures how long it takes to reach the destination. It should be noted that even if the latency may be several clock cycles, it is possible to keep throughput high by pipelining the transfers. In shared interconnections, the contention may increase the latency. Thus it is good to measure the average latency as well as the worst case latency. To make real-time systems possible the worst case timing has to be determined.

No simulation nor benchmarking was carried out to extract the performance of the buses. As the performance depends on application type, it would be beneficial for SoC designers to have a thorough, common benchmark suite for measuring the interconnection performance.

3. CONCLUSIONS

This paper describes shortly some available bus interconnections for SoC designs. As most properties are similar with only few exceptions, a standard interface would be a great improvement for design re-use. Then all IP blocks could be mixed without paying attention to underlying communication architecture. The best implementation could be found with design space exploration, which takes the bus-specific features into account.

An average version of the considered buses would use synchronous, shared, unidirectional signal lines. The data transfers would use pipelining, split transaction and handshaking, but no broadcast operation. The arbitration would be dynamic, centralized, and pipelined with data transfers. The actual algorithm would be application specific. Support for multiple clock domains, hierarchical structures, dynamic reconfiguration, and test sturctures may be included if they can be implemented with reasonable hardware overhead. All these buses have their pros and cons and it would require thorough benchmarking to make optimal choice for certain application.

4. REFERENCES

[1] K.A. Kettler, J.P. Lehoczky, and J.K. Strosnider, "Modeling Bus Scheduling Policies for Real-time Systems", *16th IEEE Real-Time Systems Symposium*, 1995, pp. 242-253.

[2] K. Lahiri, A. Raghunathan, and S. Dey, "Performance Analysis of Systems With Multi-Channel Communication Architectures", *International Conference on VLSI Design*, Jan. 2000, pp. 530-537.

[3] J. Liang, S. Swaminathan, and R. Tessier, " ASOC: A Scalable, Single-Chip Communication Architecture", *International Conference on Parallel Architectures and Compilation Techniques*, 2000, pp. 37-46

[4] M. Birnbaum, H. Sachs, "How VSIA Answers the SOC Dilemma", *Computer, Volume: 32 Issue: 6* , June 1999, pp. 42-50.

[5] AMBA Specification Rev 2.0, ARM Ltd., 1999.

[6] 32-bit Processor Local Bus Architecture Specification, Version 2.9, SA-14-2531-01, IBM Corporation, 2001.

[7] Processor Local Bus Arbiter Core User's Manual, Version 2.0, IBM Corporation, 2000.

[8] B. Cordan, "An Efficient Bus Architecture for System-on-Chip Design", *1999 IEEE Integrated Circuits Conference*, 1999, pp. 623-626.

[9] K. Kuusilinna, T. Hämäläinen, P. Liimatainen, and J. Saarinen, "Low-Latency Interconnection for IP-Block Based Multimedia Chips", *The 2nd IASTED International Conference on Parallel and Distributed Computing and Networks*, 1998, pp. 411-416.

[10] K. Kuusilinna, P. Liimatainen, T. Hämäläinen, and J. Saarinen, "Reconfiguration Mechanism for an IP Block Based Interconnection", *The 25th EUROMICRO99 Conference*, 1999, pp. 42-45.

[11] K. Lahiri, A. Raghunathan, G. Lakshminarayana, "LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs", *Design Automation Conference*, 2001, pp. 15-20.

[12] W.J. Bainbridge and S.B. Furber, "Asynchronous Macrocell Interconnect Using MARBLE", *1998 Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 122-132.

[13] Virtual Component Interface Specification (OCB 2 1.0), VSI Alliance, 1999.

[14] PI Bus VHDL Toolkit, Version 3.1, Open Microprocessor Initiative, 1997.

[15] Sonics μNetworks Techical Overview, A21-1, Sonics Inc., 2000.

[16] Open Core Protocol Specification 1.0, Version 1.2, Sonics Inc., 2000.

[17] Wishbone System-On-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision B.1, Silicore Corporation, 2001.