SoC IP Interfaces and Infrastructure
A Hybrid Approach

Cary Robins, Shannon Hill
ChipWrights, Inc.

ABSTRACT

System-On-Chip (SoC) designs incorporate more and more Intellectual Property (IP) with each year. In the early years of IP integration there were no standard interfaces and the task of integrating the IP into a given design had a large potential for introducing errors. Later, the AHB bus emerged as one of the most popular IP interface standards. As complexity of SoCs continued to increase and performance requirements grew the AHB bus architecture alone became less and less sufficient for the job.

This paper presents three generations of SoC designs beginning with a flat single AHB Bus based interconnect, followed by a multi-tier AHB/APB segmented communication infrastructure and finally our hybrid approach using both the AHB bus for control path operations and point-to-point BVCI connections through an internal crossbar for data flow. This architecture eliminates many of the dataflow bottlenecks common to SoCs and leaves the device constrained only by processing power and DRAM bandwidth. The power benefits of the architecture are also discussed throughout.

The paper also presents a discussion about options and tradeoffs in the various industry standard interfaces and justifies the selections made. Finally, various options for crossbar arbitration mechanisms are presented, tradeoffs discussed, and a solution is shown.

1. INTRODUCTION

One of the supposedly solved problems in IP integration is the use of standard interfaces. Even though today's standard interfaces have made a great deal possible in the design of SoCs, the way we often use them does not always lead to an optimal solution. Every type of standard interface protocol, including the numerous variants within each specification and the subset implementations, has its own set of advantages and disadvantages. In this paper we review some of the common SoC structures we have seen and used in recent years and then present our current SoC infrastructure and discuss its benefits.

It is a universal truth that every design team finds their optimal solution based on the specifics of their design, the specifics of their market requirements and as a function of what is available to them at the time. Even though these parameters are never identical, we believe we have identified some universal points that will benefit most SoC designs. Along the way, we will make some requests of IP vendors and recommendations on the design architecture of their complex digital IP.

2. RECENT HISTORY

Figure 1 shows the classic SoC architecture based on the Advanced High-speed Bus[1] (AHB) which is described in the Advanced Microcontroller Bus Architecture (AMBA) specification from ARM, Ltd.
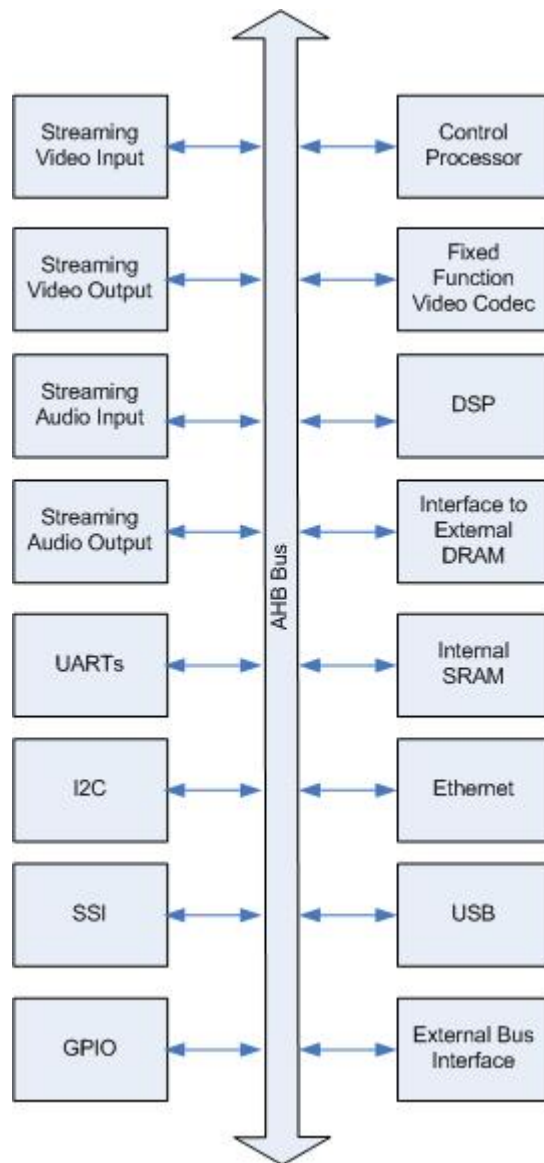

Figure 1: Classic AHB-Based SoC Architecture

Some common elements that we use in our devices are shown, but Figure 1 is fairly representative of a large number of SoC designs. Even in cases where the elements do not look familiar, the actual contents of the IP blocks are not the point, they are a context for discussing the system interconnect.

This interconnect architecture has some distinct advantages:

It is a well known, well supported, and a universally available solution. This single point cannot be overstated. It is so compelling that if this architecture enables you to meet your goals (performance, area and power) then you may just want to stop reading now!

It requires relatively little logic overhead.

It provides an easy-to-understand structure and a flat address space.

Unfortunately, the simple single AHB architecture for today's complex SoCs also has many drawbacks:

The problem of **orthogonality**. There is a single path from/to all devices. This means that any access from one device to another is blocking all other accesses even if the other access involves a different master and slave. Thus, it becomes an artificial bandwidth bottleneck for your system.

Another problem that results from all devices "waiting their turn" in a single queue for a single communication resource, is that the worst case **latency** for any device to get access to the bus can get very high. This will affect FIFO sizing and ultimately **performance**.

The AHB bus requires that a 32-bit address bus and a 32-bit data bus be routed throughout the device. In order to achieve the required performance it must run as fast as possible. These two requirements are in opposition. It is difficult to scale the speed of a common bus structure over large areas in order to keep up with the scaling frequencies of the localized processors.

Another consequence of routing a bus throughout the chip is **power** consumption. Every AHB access from Master X to Slave Y drives the full capacitance of this bus to every other Master and Slave device. When the bus is moving large amounts of streaming data, this power draw can become significant.

Because the AHB bus is a synchronous design and its performance is critical to the SoC, the maximum operating frequency that can be obtained on this bus is propagated throughout the design to subsystems that can and should run at slower clock frequencies.

The real driving problem faced by all original AHB designs was performance. Some fundamental changes in function and technology made the architecture less suitable:

Applications involved multi-media streaming data more than small individual accesses.

On-chip metal delays began to dominate performance. While processor performance had always been gated by silicon, bus delay was gated by metal. Thus the two could not scale together.

These changes led SoC architects to look at splitting the AHB bus into multiple segments and possibly putting some of the low-speed devices on the ARM Advanced Peripheral Bus (APB)[1]. Figure 2 shows a typical split bus configuration.
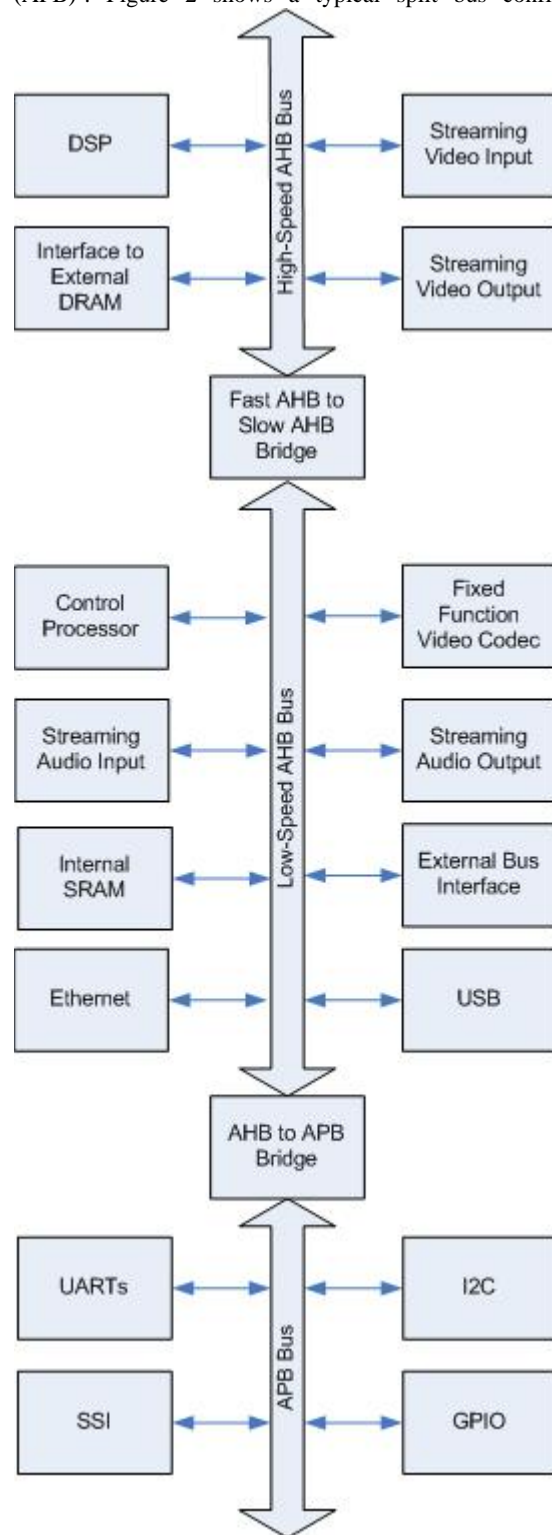

Figure 2: Split Bus SoC Architecture

Note: It is possible to run one high-speed AHB bus in a localized area for the high speed devices and then use another for lower speed devices. Benefits of this approach are:

The power and routing ramifications of the bus architecture are divided by the number of segments.

The smaller bus segments (with less metal) make it possible to scale the frequency higher.

If the fastest bus has few devices, it can be implemented as an AHB crossbar. A solution that is prohibitive with too many devices.

However, many disadvantages remain and some new ones are introduced:

The simplicity and low logic overhead of the AHB bus is compromised. Special high-speed to low-speed and low-speed to high-speed translators are required. To get reasonable performance, queues must be implemented to "store and forward" transactions between the different speed buses.

Whenever a device on the low-speed AHB must access a device on the high-speed AHB, (or vice versa), devices on both buses are stalled. The result is often that the performance gains of the high-speed bus are lost. (This problem can be mitigated somewhat by using the split transaction protocol that the AHB bus supports, and adding hardware to store and forward transactions between the two bus structures. However, the AHB protocol is not a true split transaction model. The master which receives a split transfer response is then stalled until the transaction completes. Thus the round trip access time still limits performance instead of just latency. We have also found that many available IP devices do not support split transactions.

The blocking problem for orthogonal transactions is cut in half, but not truly solved. This holds true for the latency issue as well.

3. OUR SOLUTION

Let's compare the classic bus structure in Figure 1 with the changes made in Figure 2. One benefit of the bus structure in Figure 1 is that it allows ultimate flexibility. Any device can access any other device anywhere in the system. Any device can be a master, a slave, or both. Our observation is that this generality is not really required. For example, the Ethernet and USB DMA engines do not need access to APB devices like UARTs, and SSI type interfaces. We can re-sort the devices in these diagrams into three target devices being arbitrated for, regardless of the number of processing elements or DMA Engines. These three devices are:

1. External DRAM
2. Internal SRAM
3. Control and Status Registers (CSRs) everywhere

It is a sufficient optimization to allow concurrent access to the SRAM, DRAM and the register space as a single unit. Thus, we can solve the problem of orthogonality (stop transactions between distinct masters and slaves from blocking one another) by architecting a system with three connection fabrics.
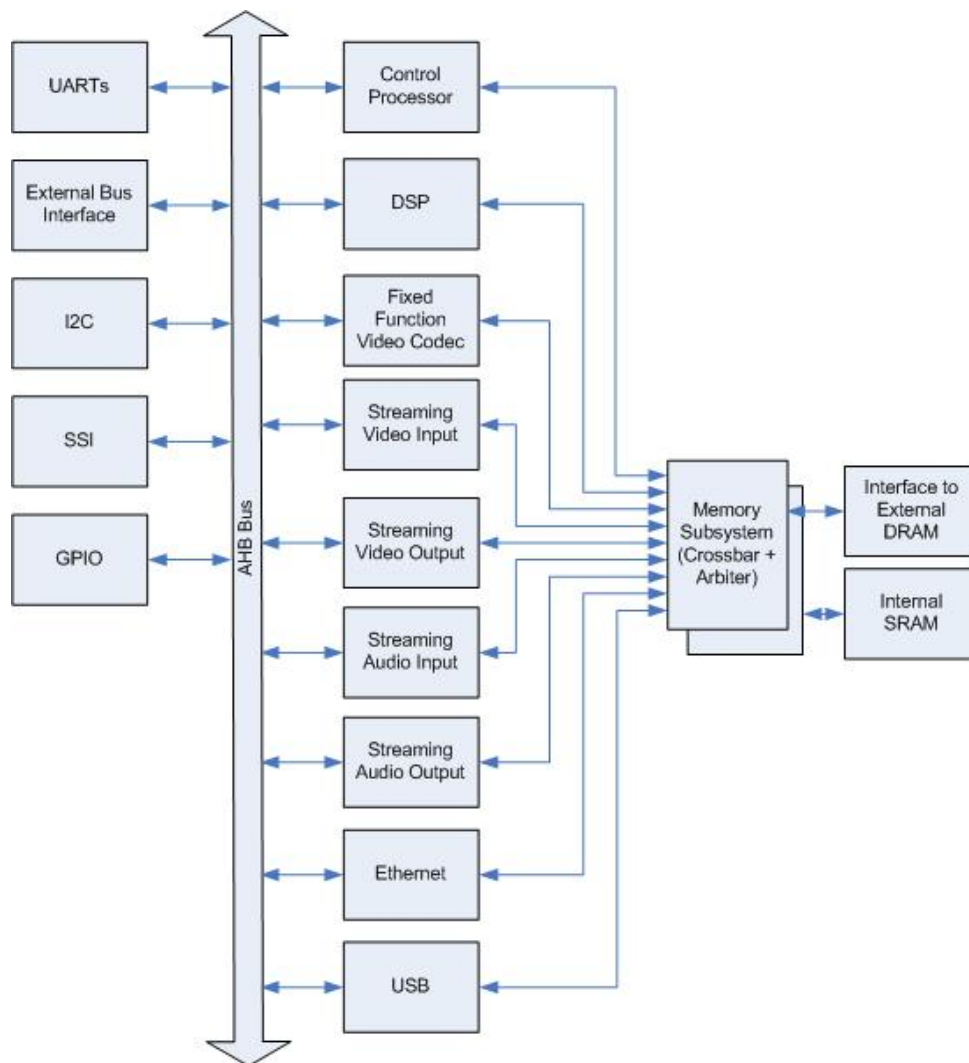
We agree that the CSRs are not one device, they are many. And it would be nice if any master could load or query the CSRs of any device independent of the others, but this is an unnecessary optimization. These are very low-speed devices that move little or no data, and in many cases, access is only performed at startup. The total bandwidth required for CSR access is easily handled by even the slowest bus implementation.

One way to solve the orthogonality problem would be to propose three AHB bus segments for our application: one AHB bus tied to the DRAM, one to the SRAM, and one for the CSR interfaces to each block. Each master device would have a distinct AHB master controller to access each bus. This does allow simultaneous access to the two memory subsystems and the CSR address space, but is clearly a disaster in terms of implementation. It multiplies all of the problems described in Figure 1 by three instead of dividing them as we did in Figure 2. Instead we have chosen a hybrid approach as shown in Figure 3.

Given the low speed requirement for CSR accesses and our requirement to maintain our multi-master generality (i.e. all masters have access to all registers) we implemented a standard bus for this part of the problem. Since the multi-master feature was required, the AHB is used as opposed to the APB. The clock frequency on this AHB implementation can be very slow, thus the metal delay (timing closure) and power problems are no longer an issue. The worst case latency problems associated with bus architectures are also not an issue because we are not moving any high-speed data on this bus.

For each of the two memory elements, we implemented a crossbar/arbiter with low overhead point-to-point links from each master device (processor or DMA) to the crossbar. The point-to-point interface protocol chosen is the Basic Virtual Component Interface (BVCI)[2] as defined by the VSIA industry consortium.

Each memory requester (master device) has a direct connection to each Memory Sub-System (MSS). A synchronization boundary is implemented within the MSS for all paths into the crossbar. Each access to the memory charges up wires of the minimum length required to get from one device to the other, each connection switches only when it is moving data, and the clock frequency of each connection is exactly what the device requires and no more. In order to allow full bandwidth access to each memory, the MSS switch and arbitration logic runs at the frequency of the memory element it connects. This high-speed logic is contained within a small area allowing for lower power and more importantly, the ability to scale in the same ways as the processors. In power saving modes, when the full memory bandwidth is not required, the clock frequency of the memory and its associated MSS can be scaled to meet the application demands.

Figure 3: Hybrid Solution

are well known and understood. For this reason, we have limited our discussion of bus architectures to AMBA protocols.

APB and AHB-lite are not suitable for our application because we require multi-master functionality. ASB is perfect for our needs but are not frequently seen in the industry. AHB interfaces are by far the most widely available and are a natural choice for us despite the added complexity over ASB and APB. The emerging AXI standard is overkill and is intended more for the data connection portion our system.

However, we have one problem with any of these bus protocol options. The bus protocols are synchronous and run at a clock frequency that must be defined by the overall system design. It cannot be specified differently by each IP block. At the same time, many IP blocks have a required clock frequency associated with their physical interface. Others have a clock frequency defined by their performance requirements/capabilities. In order for this to work, a clock boundary must be defined within the IP to connect the AHB bus interface to the native clock domain.

If at all possible, we would prefer that our IP vendors not solve the asynchronous clock boundary problem within their core. In the best case, an IP block has only one clock domain for the

We would like to have IP available that has two distinct internal interfaces; a bus interface for Control and Status Registers only, and a point-to-point link for Burst DMA data.

The second interface is a master, if the DMA engine is included, and a slave, if the DMA engine is external to the IP. It is important to note that these two interfaces *not* require use of the same clock.

The logical choice for the control interface is the industry standard AHB Bus. For streaming data applications the best data interface is a point-to-point link. We think that the BVCI protocol is the best one defined for this task.

4. THE BUS INTERFACE

We are not necessarily partial to ARM as the supplier for our bus standards, but they have done a good job in defining a variety of bus protocols (APB, ASB. AHB and AXI) and they

entire design. This makes the back end design process (synthesis, place and route, test insertion, gate level simulations) much less complicated. If multiple domains are necessary, they should be delivered as distinct hierarchies. This requires that the AHB Slave Interface be delivered as a separate module that interfaces to the IP (preferably) with a simple asynchronous request/ack, address/data protocol. The boundary synchronization would be done in the AHB interface module. The native clock of the IP should be used to drive the CSRs, the core IP and the BVCI link to the crossbar.

Over the years, we have also found many of our system bugs to be caused by incompatibilities between AHB implementations from different vendors. For this reason, our most preferred solution is to use our own AHB slave module that has clock boundary crossings included, and interface to the IP with the simple asynchronous handshake.

5. THE POINT-TO-POINT INTERFACE

The Virtual Socket Interface Alliance (VSIA) defined three point-to-point protocols, called Virtual Component Interfaces (VCI), for use as on-chip inter-IP connections. These are Peripheral (PVCI), Basic (BVCI), and Advanced (AVCI). BVCI is probably the most popular of the three and we think the right compromise between functionality and complexity.

In the years since we started using the protocol, two new standards have become available. AXI from ARM Ltd. addresses many of the same issues as BVCI, but is not really suitable for our needs. The Open Core Protocol International Partnership (OCP-IP) is another industry consortium that is the follow on to VSIA. This specification is really a superset of BVCI to encompass out of band signals and test features. Both of these additions are beyond the scope of this paper but OCP-IP could easily be substituted for BVCI in all cases.

The most important point to make is that the exact choice of standard is not the critical issue. It is relatively easy to perform translation between these interfaces but harder to do so between a point-to-point and a bus type structure.

A major advantage of BVCI is it's "fire and forget" capability. The outgoing read request transaction is decoupled from the return data transaction. Outgoing Read transactions are released in one clock cycle and (since they only contain an address) they pass through the switch in one cycle. Outgoing Write transactions take one cycle for address and one cycle for each data word in the burst and are released with a local handshake acknowledgment, but without an acknowledge extending back from the memory write operation. The outgoing read and write transactions can be interleaved and pipelined as deeply as necessary. Read data is returned on a distinct return path in the order the Read operations went out. The pipelining will affect read latency, but not system performance.

The PVCI protocol does not support "fire and forget" and the interface stalls until the read data returns. The AVCI does not require that read data return in order, but supports tagging to match return data with read requests. For our needs the former is too limiting and the latter is unnecessarily complex.

The FIFO nature of this interface makes clock boundary crossings very easy to implement. Using a FIFO on the interface helps buffer bursts of request and thus reduces the incidence of back pressure from the switch allowing the IP to operate without stalls.

It is worth noting also that for our switch implementation, we chose to multiplex address/commands with write data. This reduces the size of the switch with no real performance penalty.

6. CROSSBAR ARBITRATION

Arbitration for access to the SoC interconnect fabric is really a separate issue from the fabric itself. Arbitration is necessary for any of the architectures shown, and the arguments for the benefits of various schemes are nearly the same in all cases. In our view, the fabric and the arbitration combine to make what we have been calling the infrastructure. The arbitration mechanism will have a significant effect on the success of the fabric in meeting performance goals.

If the system interconnect is underutilized, say 50% - 60% of its potential bandwidth, then it is likely that the arbitration mechanism doesn't matter at all. It is simply intuitive that if there is very little traffic, then you don't need traffic lights. Likewise, in a very busy system, requiring say greater than 90% of its potential bandwidth, it is likely that no arbitration mechanism exists that will prevent overflow or underflows in some of the critical real-time queues. The bandwidth performance difference between these two extremes is where the argument for arbitration lies. The better conceived your arbitration mechanism, the closer to full utilization you can achieve. In today's power-sensitive systems it is often more desirable to throw some logical complexity at the issue rather than beef up the bus by increasing width or clock frequency.

Static Fixed Priority. In an underutilized system, the simplest mechanism of all is fixed priority that is defined in the logic and is not changeable, i.e., Requester 1 has highest priority, Requester 2 has second highest priority, and so on. If this gets the job done, then it also has the benefit of very simple logic. One caution is to be very careful in your selection of priorities as they cannot be tweaked. Some system simulations before tape out are vital.

Programmable Fixed Priority. This allows for the software to configure the priorities at run-time. This is not hard to implement and the biggest benefit is that you don't have to get it right before you have silicon to experiment with. It also allows for some tweaking depending on the application. But again, in an underutilized system, none of this may be a problem.

Round-Robin Priority. With this approach, each requester takes successive turns accessing the memory. The arbiter cycles around a loop of Requesters and grants access to the next Requester in the loop with its request signal asserted. If only one requester is active, it can use the entire bandwidth of the bus. If two are active, then they will alternate bursts. This has the distinct advantage of eliminating the problem of starvation of one requester due to the large bandwidth of a higher priority requester. It has the disadvantage of possibly under serving some of the critical real-time requesters.

Weighted Round-Robin (WRR) Priority. This is a round-robins solution that allows software to tweak the relative importance (weight) of the various requesters and to defer to some more often.

There are several ways to implement these more complex arbitration mechanisms. We chose a simple one that provides options for all four of these common mechanisms. We used a programmed table based mechanism as shown in Table 1. The approach is loosely based on the one specified for SPI-4 by the Optical Internetworking Forum[3].

	Address	Data Entry
	0x00	Requester 0
	0x01	Requester 1
	0x02	Requester 2
	0x03	Requester 3
	0x04	Requester 4
Last Grant Ptr →	0x05	Requester 5
	0x06	Requester 6
	0x07	Requester 7
	0x08	Requester 8
	0x09	Requester 9
	0x0A	Requester 10
	0x0B	Requester 11
	0x0C	Requester 12
	0x0D	Requester 13
	0x0E	Requester 14
	0x0F	Requester 15
	0x10	Requester 0
	0x11	Requester 1
	0x12	Requester 2
	0x13	Requester 3
	0x14	Requester 4
	…	…
	0x3D	Requester 13
	0x3E	Requester 14
	0x3F	Requester 15

Table 1: Table Based WRR Arbiter Example

A small programmable table of N times the number of requesters is used. The larger values of N will provide greater granularity and flexibility at the cost of size. For our system we chose N=4.

The table is initialized at power on reset with the numbers of the requesters in order as shown in Table 1. On each arbitration cycle, the arbiter searches for the next Requester in the table with its Request signal asserted. After each grant, it holds its place in the loop and continues from there. This is an implementation of Round-Robin arbitration. If WRR is necessary, the table can be reconfigured by the system software to reorder the Requesters, and more importantly to give some Requesters more slots than others. Be aware that if a Requester has no entry at all in the table, it will never get access.

A simple option to the table based arbiter is to reset the Last Grant pointer to the top of the table after each arbitration cycle. This creates a Fixed Priority mechanism. Note that the priority is inherently programmable in this implementation.

Multi-class Requesters. The ultimate arbitration solution is to provide multiple classes of requesters each with their own arbitration scheme amongst themselves and another higher-level arbitration scheme between the classes. One can imagine a WRR to select the class first and then another WRR to select the Requester. The useful aspect of this approach is to recognize two classes of Requesters. One class includes those requesters that have real-time requirements (audio and video queues) and the other contains all of the rest. Use a static fixed priority between the classes, i.e., always serve any requester in the real-time class, and then look at requesters in the standard class. Within each class an RR or WRR could be used.

As always, tradeoffs are made between the complexity implemented in the arbiter and the requirements of the application.

7. CONCLUSION AND SUMMARY

We have presented an SoC Infrastructure solution based on two different classes of standard interfaces. In our case we chose AHB for the bus structure and BVCI for the point-to-point link. This solution solves the following problems:

Orthogonality. The system can allow simultaneous access to all of the memory devices without interfering with one another.
Performance. Aside from the simultaneous access issue, we have eliminated the inherent difficulty in getting the bus structure to run fast enough and limiting the system bandwidth.
Power. This is a problem that must be solved on many fronts, but we have reduced the burden of driving large capacitances on each data transaction and we have facilitated the easy scaling of memory subsystem frequencies. We have also reduced the interface clock speeds on many devices.
Stalls. Except in very busy systems, we have virtually eliminated the effects of back pressure on the IP subsystems.
Memory Bandwidth. Each memory subsystem can potentially achieve its maximum bandwidth capacity.

The architecture removes all artificial internal bandwidth bottlenecks and limits performance only by the processing and inherent memory bandwidth limitations.

Our recommendations and requests for IP vendors and suggestions for the community at large are the following:

1. Separate the CSR interface to your IP from the Streaming Data interface.
2. Implement the CSR interface as a bus structure for its generality. Offer both AHB and a simple asynchronous option.
3. Implement the Streaming Data interface with the BVCI point-to-point protocol.

8. REFERENCES

1. ARM, Ltd.: "AMBA Specification," Rev. 2.0, May 1999.
2. Virtual Socket Interface Alliance (VSIA): Virtual Component Interface Standard 2.0, April 2001
3. OIF-SPI-4-02.1 System Packet Interface Level 4 (SPI-4) Phase 2 from the Optical Interworking Forum (OIF), October 15, 2003