IMPLEMENTATION OF AHB PROTOCOL USING FPGA

Mrs.Bhavana L. Mahajan[1], Dr.A.S.Hiwale[2], Mrs.Kshitija S.Patil[3], Prof.G.D.Salunke[4]

*1. Student (ME),E&TC,GSMCOE,Pune,Maharastra,India,**mlbhavana@gmail.com***
*2. Principal,E&TC,GSMCOE,Pune,Maharastra,Indi,**ashiwale@gmail.com***
3.** .Student (ME),E&TC,GSMCOE,Pune,Maharastra,India,**kspatil87@gmail.com
*4. Asst. Prof,E&TC,GSMCOE,Pune,Maharastra,Indi,**geetasalunke@g.mail.com.***

Abstract

Resolution is a big issue in SOC (System On Chip) while dealing with number of masters trying to sense a single data bus. The effectiveness of a system to resolve this priority resides in its ability to logical assignment of the chance to transmit data width of the data, response to the interrupts, etc. The purpose of this seminar report is to propose the scheme to implement such a system using the specification of AMBA bus protocol. The scheme involves the typical AMBA features of 'single clock edge transition', 'split transaction', 'several bus masters', 'burst transfer'. The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. Even though the arbitration protocol is fixed, any arbitration algorithm, such as highest priority or fair access can be implemented depending on the application requirements. The design architecture is written using VHDL (Very High Speed Integrated Circuits Hardware Description Language) code using Xilinx ISE Tools. This paper aims at covering the basics of buses, AMBA bus basics, overview of AHB Arbiter, various arbitration algorithms, their comparison, and finalize the best suitable algorithm for the above implementation.

Index Terms: *Amba bus, AHB, ASB, AHB, VHDL, round robin*

--***--

1: INTRODUCTION

1.1Buses

Buses are shared communication media used by devices to "talk to" each other both on-chip and off-chip. The communication actions which take place can carry both data and control structures.

The BUS Specification

AHB (AMBA High-performance Bus) is a bus protocol introduced in AMBA specification version 2 published by ARM limited Company. In addition to previous release, it has the following features:

- Single edge clock protocol
- Split transaction
- Several BUS Master
- Burst transfers
- Pipelined operations
- Single cycle bus master handover
- Non-tristate implementation
- Large bus-widths (64/128 bit)
- The AMBA specification describes an on-chip communications standard for designing High-performance 16 and 32-bit microcontrollers, signal processors and complex peripheral devices.
- AMBA has been proven in and is being designed into:
- • PDA microcontrollers, with a high number of integrated peripherals but also with very low power consumption
- Multi-media microcontrollers with floating-point co-processors, on-chip video controller and high memory bandwidth

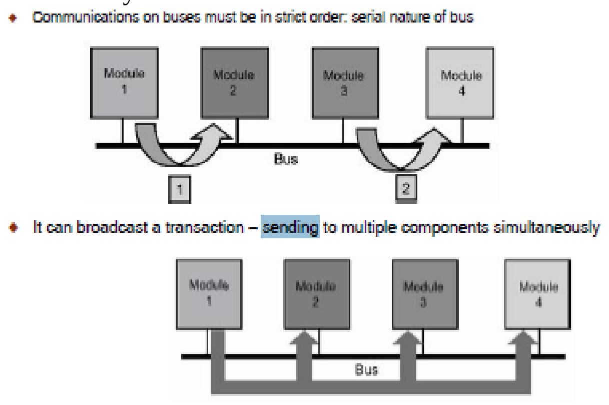
Communications on buses must be in strict order: serial nature of bus
Module 1
Module 2
Module 3
Module 4
Bus
1
2
It can broadcast a transaction – sending to multiple components simultaneously
Module 1
Module 2
Module 3
Module 4
Bus

Fig.1 Bus basics: order and broadcast properties

• Complex peripheral ASICs for consumer products
• Digital mobile communication devices integrating control and signal-processing functions
ARM's policy is to encourage the use of AMBA wherever possible. ARM partners have access to HDL models, development boards and other tools that support AMBA.

AMBA Specification

The AMBA specification defines:
• A high-speed, high-bandwidth bus, the Advanced System Bus (ASB)
• A simple, low-power peripheral bus, the Advanced Peripheral Bus (ASP)

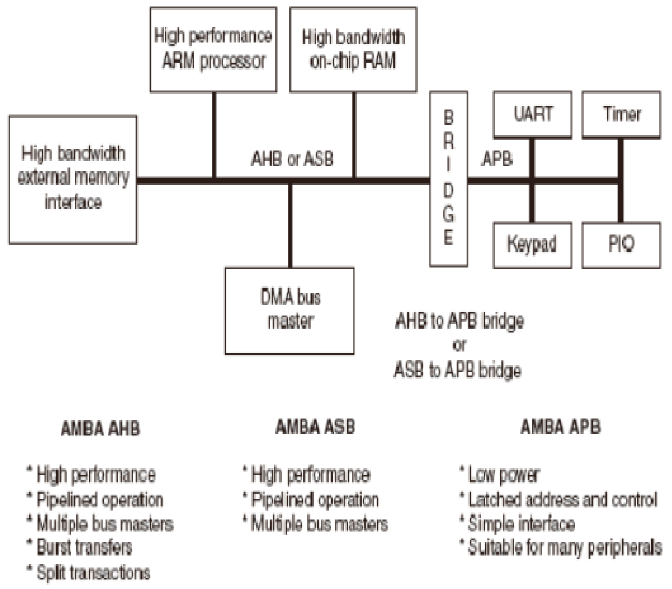

Fig.2 AMBA Bus

• Access for an external tester to permit modular testing and fast test of cache RAM
• Essential housekeeping operations (reset/power-up, initialization and power-down)

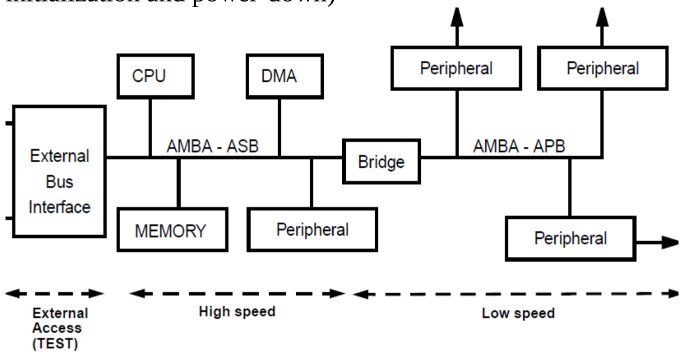

Fig.3 A typical AMBA-based microcontroller

A typical AMBA-based microcontroller shows:

1.1.1 The Advanced System Bus (ASB)

The ASB is designed for high-performance, high-bandwidth usage:
• Non-multiplexed (i.e. separate) address and data buses
• Support for pipelined operation (including arbitration)
• Support for multiple bus masters, with low silicon overhead
• Support for multiple slave devices, including a bridge to the peripheral bus (APB)
• Centralized decoder and arbiter

Absolute transfer rates depend on many design factors, but, for comparison purposes,if a 32-bit data path and a 100MHz clock are assumed, 200Mbytes/sec rate can be achieved. These figures are not limited by the specification but are simply provided for clarification. Multiple bus masters are supported through the use of bus request, bus grant and bus lock signals. Use of these signals is optional; if you have a single bus master, you do not have the penalty of implementing these bus control lines.The high-performance bus which is the main system 'backbone'. This bus is also able to sustain the data rates required by the external us interface. The CPU and other bus masters (such as a DMA controller), and high-speed local memory are normally connected to this bus. (The ASB is connected by a bridge to the simpler APB)

1.1.2 The Advanced Peripheral Bus (APB)

The APB is designed to be a secondary bus to ASB, connected by a bridge (which limits the ASB loading). APB is a much simpler bus and has a low power focus:
• Data access is controlled by select and strobe only (i.e. no clock, and thereby reducing power)
• Almost zero-power consumption when bus is not in use
• Simple unpiplined interface, typical of that required by many simple peripheral microcells.

Data transfer rates are dependent on the speed of the peripherals. A single read or write cycle takes 5 clocks, so assuming a 32-bit data path and 100MHz clock, the data rate is 80Mbytes/sec. These figures are not limited by the specification but are simply provided for clarification. The data bus of the APB can be more readily optimized to suit the peripherals connected. Many peripherals have narrow data path needs, and one mechanism may be to connect the 32-bit peripherals next to the bridge and 8-bit peripherals furthest away, reducing the die area needed for the bus. Although the clocking strategy is not specified in AMBA, the partitioning provided by the bridge and APB does suggest a good starting point for minimizing power consumption. Many peripherals (timers, baud rate generators, pwm units) require a divideddown system clock, and locating a single programmable divider adjacent to the bridge is convenient and power-efficient.

The simple, low-speed, low-power peripheral bus. This is often, but not always, a narrower bus and is designed to be simple (i.e.unpipelined) for connecting many common peripherals such as timers, parallel I/O ports, UARTs, etc. (By placing these infrequently accessed peripherals on the APB, and partitioning them away from the ASB, loading on the ASB is reduced and allows maximum performance on the ASB to be more readily achieved.)

1.1.3 Advanced High-performance Bus (AHB)

AHB is a bus protocol introduced in Advanced Microcontroller Bus Architecture version 2 published by ARM Ltd company. In addition to previous release, it has the following features:

- single edge clock protocol
- split transactions
- several bus masters
- burst transfers
- pipelined operations
- single-cycle bus master handover
- non-tristate implementation
- large bus-widths (64/128 bit).
- 2nd-generation AMBA system bus
- Synchronous, no multiplexed bus
- Separate read, data buses
- Multimaster, arbitrated bus
- 32-, 64-, 128-, 256-bit data paths
- 32-bit address bus
- Pipelined, split transactions
- Supports bursts (4-, 8-, 16-beat)
- Non-tristate, multiplexer implementation

The AHB takes on many characteristics of a standard plug-in bus. It's a multimaster with arbitration, putting the address on the bus, followed by the data. It also supports wait-state insertion and has a data-valid signal (HREADY). This bus differs in that it has separate read (HRDATA) and write (HWDATA) buses. These bus connections are multiplexed, rather than making use of a tristate multiple connections.

AHB supports bursts, with 4-, 8-, and 16-beat bursts, as well as undefined-length bursts and single transfers. Bursts can be address wrapped, i.e., staying within a fixed address range. Bursts can't cross a 1-kB address boundary, though. Slaves can insert wait states to adjust its response (up to 16).

All bus operations are initiated by bus masters, which also can serve as a slave. The master-generated address is decoded by a central address decoder that provides a select signal to the addressed bus slave unit. The bus master can "lock" the bus, reserving it with the central arbiter for a series of locked transfers.

The slave unit has the option to terminate a transaction as an error, signal the master to retry, or split the transaction for later completion. Split transactions enable the slave to defer the operation until it's able to accomplish it, thereby releasing the bus for other accesses. The slave signals a split transaction and saves the master number (HMASTER\[]). When ready to complete the transaction, the slave signals the arbiter with the master number. When the arbiter grants bus access to the master, it restarts the transaction. No master can have more than 1 pending split transaction.

AHB supports 32, 64, and 128-bit data-bus implementations with a fixed 32-bit address bus. It is a synchronous bus that supports bursts and pipelining of accesses to improve throughput. The AHB system bus and APB peripheral bus are linked through a 'bridge' that acts as the master to the peripheral bus slave devices. The peripheral bus (APB) is a simpler, lower-speed, low-power bus for slower devices. It is typically used for connecting peripherals such as UARTS, rather than for SRAM, Flash etc. as these will be on the AHB, requiring the additional bandwidth. The AHB and APB can run at different clock rates. AHB supports multiple masters (either through a central arbiter, or through slave level arbiters in the case of a multi-layer AHB-lite system). The arbiter has the task of determining which master gets to do an access. Every transfer has an address/control phase and a separate data phase. They're both pipelined (able to start the next transfer's arbitration and address phase while finishing the current transfer).The address transfer is always followed by the data phase. A slave (memory or peripheral device which accepts a read or write request from a master) can prolong the transfer (add wait states) using the HREADY signal. Separate uni-directional buses for read (HRDATA) and write (HWDATA) are used.

Burst Support

AHB supports bursts, which can either be of undefined-length or fixed length (4, 8 or 16 beats). There is also, of course, the possibility to do a single transfer (one read or write). Bursts may be performed to a fixed address (eg for FIFO access), increment addresses (in steps of a single increment equal to the size of the access) or wrap (where a critical word within a cache line is accessed first). Bursts may not cross a 1kB boundary, to simplify slave handling of bursts and address decoder design. The address from a master is decoded by a central address decoder that provides a select signal to one of the slaves.

Support for bus locking, error signaling and split accesses

The bus master can lock the bus, allowing it to perform a sequence of atomic, locked transfers, with guarantees that other masters cannot perform intervening accesses. This is typically used to implement mutexes or semaphores between masters. Slaves may respond to accesses by the master by signaling OK, or by reporting an error. In the full AHB system (but not AHB-lite), slaves may also give a retry response, or the less commonly used split response. Split transactions let the slave to delay completion of the access until ready but to free the bus for other accesses by a different master. The slave records the number of the master and signals the arbiter when the split transfer can complete. When the arbiter re-grants the bus to that master, it restarts the transaction. A master can have only one pending split transaction.

AHB (Advanced High-performance Bus) X as a later generation of AMBA bus is intended for high performance high-clock synthesizable designs. It provides high-bandwidth communication channel between embedded processor (ARM, MIPS, AVR, DSP 320xx, 8051, etc.) and high performance peripherals/ hardware accelerators (ASICs MPEG, color LCD, etc), on-chip SRAM, on-chip external memory interface, and APB bridge. AHB supports a multiple bus master’s peration, peripheral and a burst transfer, split transactions, wide data bus configurations, and non tristate implementations. Constituents of AHB are: AHB-master, slave-, arbiter-, and Xdecoder.A simple transaction on the AHB consists of an address phase and a subsequent data phase (without wait states: only two bus-cycles). Access to the target device is controlled through a MUX (non-tristate), thereby admitting bus-access to one bus-master at a time. **AHB-Lite** is a subset of AHB which is formally defined in the AMBA 3 standard. This subset simplifies the design for a bus with a single master.

1.2 AHB Components

AHB master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.(max. 16)

AHB slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.

AHB arbiter ensures that only one bus master at a time is allowed to initiate data transfers.

AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer. A single centralized decoder is required in all AHB implementations.

2: AMBA BUS ARBITRATION

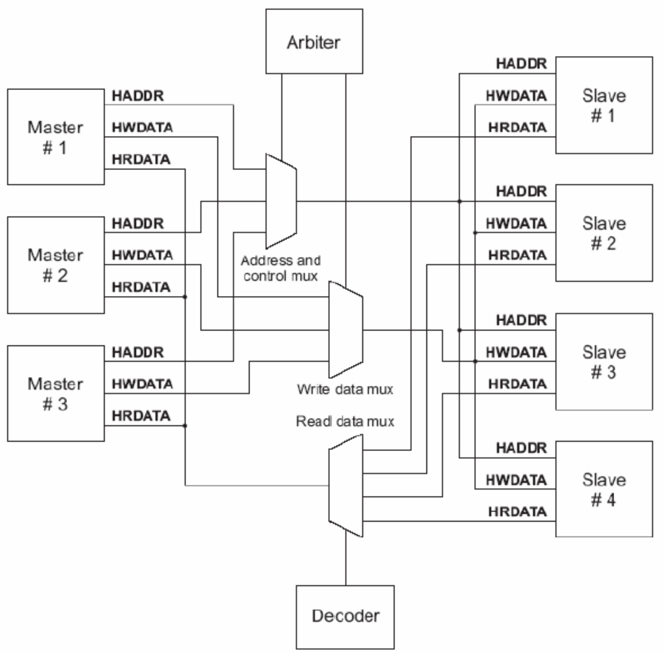

Fig.4 AMBA bus arbitration

As with other AHB blocks, the arbiter may be very simple, or quite complex. There may be up to 16 masters in the system. Each one of them has a HBUSREQ bus request output which goes to the arbiter and a corresponding HGRANT input which the arbiter uses to indicate which master has been selected. The AHB specification does not provide any specific guidance on how the arbiter should decide which master gets the bus. Schemes in common use are either priority based or cyclic. In the priority case, one master is more important than the other and if that master requests the bus, it will always be granted. A low priority master is granted only if no higher priority requests are present. In the cyclic case, each master is given a turn at controlling the bus for a certain number of cycles, then the next master gets it for some cycles and so on. It is possible (but rare) for HADDR to be used by the arbiter - for example, it may be designed to recognize that when a master is accessing a particular slave, that access may have higher (or lower) priority than normal.

It is often a good idea to design the arbiter such that it tries to avoid changing ownership of the bus until the end of the burst, as this maximizes available bandwidth. It is not possible to do this in all cases, though. Obviously, if the slave returns ERROR, RETRY or SPLIT, the master may choose to end the burst. For an INCR burst of undefined length, the arbiter cannot know when the burst will end and therefore cannot predict when it is safe to handover to another master. Even for a fixed length burst (where HBURST indicates the transfer will have 4, 8 or 16 bits), while the arbiter can be designed to recognize this case and count the number of transfers, it is not possible to guarantee that the burst completes. The difficulty is that HBURST is sampled on the first rising HCLK edge of the burst, but this could co-incide with a cycle where the arbiter has already to change HGRANT. So, the arbiter would change control of the bus on the first cycle of the burst. To avoid this problem would need HBURST to be factored combinatorially into the HGRANT generation logic and the specification does not allow that.

2.1 AHB Arbitration

Features

- Round robin priority
- Scalable (Up to 16 masters)
- AMBA® 2.0 AHB interface
- HWDATA, HADDR and AHB control steering
- HBUSREQ and HGRANT arbitration

The Ahb Arbiter is used in AMBA® 2.0 AHB multi-master systems to arbitrate the access to the AHB bus. The Ahb Arbiter is basically a “traffic controller” which allows the AHB bus to be shared between multiple bus masters such as processors, DMA controllers, and peripheral core master interfaces.

The Ahb Arbiter uses a round robin priority scheme with Master0 having the default priority. This priority scheme assures that each master equally has it’s turn at acquiring and completing an AHB bus transaction. Each inactive master is locked out (HLOCK) while the active master has access to the bus to prevent contention.

The Ahb Arbiter steers all the AHB HWDATA, HADDR, HTRANS, HWRITE, HSIZE and HBURST signaling from each master to the AHB system bus.

The Ahb Arbiter is delivered as a three master arbiter but can easily be configured to allow up to sixteen AHB bus masters.
IP Package

The Ahb Arbiter package includes fully tested and verified Verilog source. The Ahb Arbiter can also be delivered as an FPGA Netlist for Xilinx, and Altera FPGAs.

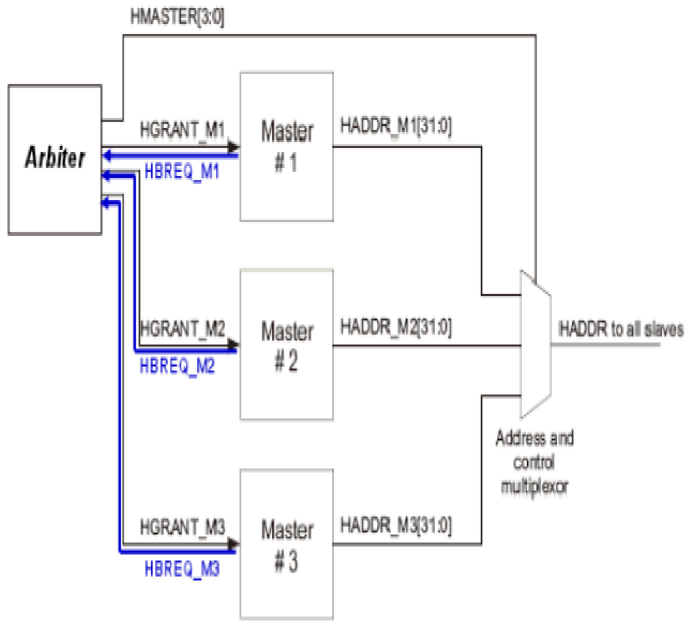

Fig.5 AHB Arbitration

2.2 Limitation of AHB

There are literally a billion or more shipped devices containing systems built around the AMBA AHB bus architecture. It is relatively easy to understand and design around. It's relatively synthesis & EDA tool friendly and is widely supported not just by ARM and its licensees but also by many other semiconductor IP and EDA companies. There are some things that AHB doesn't do well, though. On this page, we'll briefly discuss some of those things.

Lack of parallelism

One issue in higher-performance systems is that the original protocol lacks support for parallelism. Although multi-layer AHB allows multiple masters to talk to multiple slaves at the same time, this is still done through a matrix of point-to-point AHB-lite systems and there is consequently always in-order completion to a particular master. It means that slave wait states cannot be hidden to the master and there is no ability to have multiple outstanding requests.

Arbitration overhead

High arbitration overhead can also be an issue with AHB systems. If the full AHB bus request/grant protocol is used (rather than multi-layer AHB-lite), then there is a two cycle arbitration overhead. Although this can be hidden to some extent by the use of bursts, it does mean that designers have to think carefully about arbitration and the use of fixed priority or round-robin schemes. AHB-lite removes this problem, but there is still the issue of slave-level arbitration and consequent variable latency when two masters access the same slave.

Re-usability of components

Another issue with the bus protocol itself is that any system will have two components that are specific to that system. Although masters and slaves can (in principle) be re-used from one design to the next, the same is not generally true of arbiters and decoders. These components are unique to a particular system configuration. Further difficulties (not really the fault of the AHB protocol itself) are that although masters and slaves should conform fully to the specification, often what happens is that designers will take short-cuts and omit support for those features they don't plan to use. This can cause problems when these slaves are re-used elsewhere (or indeed in the original design if incorrect assumptions have been made about the subset of the full specification used by the master). Some examples seen in multiple customer designs have been slaves which do not support the master issuing the BUSY cycle type and memory slaves which have been unable to cope with WRAP write bursts.

Timing closure

A common problem related to AHB system design is that of timing closure. After completing RTL design and starting netlist generation (or even as late as STA on post-layout netlists), it may be discovered that a timing path is too long for the bus clock cycle length. To resolve this typically means inserting wait states into the slave design, or (even worse), adding cycles to the arbiter or decoder. This needs RTL changes and may dramatically decrease system performance.

3: TOPOLOGIES

In respect to topology on-chip communication architectures can be classified as:

Shared bus: The system bus is the simplest example of a shared communication architecture topology and is commonly found in many commercial SoCs .Several masters and slaves can be connected to a shared bus. A block, bus arbiter, periodically examines accumulated requests from the multiple master interfaces and grants access to a master using arbitration mechanisms specified by the bus protocol. Increased load on a global bus lines limits the bus bandwidth. The advantages of shared-bus architecture include simple topology, extensibility, low area cost, easy to build, efficient to implement. The disadvantages of shared bus architecture are larger load per data bus line, longer delay for data transfer, larger energy consumption, and lower bandwidth. Fortunately, the above disadvantages with the exception of the lower bandwidth may be overcome by using a low-voltage swing signaling technique.

Hierarchical bus**:** this architecture consists of several shared busses interconnected by bridges to form a hierarchy. SoC components are placed at the appropriate level in the hierarchy according to the performance level they require. Low performance SoC components are placed on lower performance buses, which are bridged to the higher performance buses so as not to burden the higher performance SoC components. Commercial examples of such architectures include the AMBA bus, Core Connect, etc. Transactions across the bridge involve additional overhead, and during the transfer both buses remain inaccessible to other SoC components. Hierarchical buses offer large throughput improvements over the shared busses due to: (1) decreased load per bus; (2) the potential for transactions to proceed in parallel on different buses; and multiple ward communications can be preceded across the bridge in a pipelined manner
Ring**:** in numerous applications, ring based applications are widely used, such as network processors, ATM switches. In a ring, each node component (master/slave) communicates using a ring interface, are usually implemented by a token pass protocol.

3.1 On-chip communication protocols

Communication protocols deal with different types of resource management algorithms used for determining access right to shared communication channels. From this point of view, in the rest of this section, we will give a brief comment related to the main feature of the existing communication protocols.

Static-priority**:** employs an arbitration technique. This protocol is used in shared-bus communication architectures. A centralized arbiter examines accumulated requests from each master and grants access to the requesting master that is of the highest priority. Transactions may be of non-preemptive or preemptive type.AMBA, Core Connect... uses this protocol.

Time Division Multiple Access **(TDMA):** the arbitration mechanism is based on a timing wheel with each slot statically reserved for unique master. Special techniques are used to alleviate the problem of wasted slots. Sonics uses this protocol.

Lottery**:** a centralized lottery manager accumulates request for ownership of shared communication resources from one more masters, each of which has, statically or dynamically, assigned a number of X lottery tickets.

Token passing**:** this protocol is used in ring based architectures. A special data word, called token, circulates on the ring. An interface that receives a token is allowed to initiate a transaction. When the transaction completes, the interface releases the token and sends it to the neighboring interface.

***Code Division Multiple Access* (CDMA):** this protocol has been proposed for sharing on-chip communication channel. In a sharing medium, it provides better resilience to noise/interference and has an ability to support simultaneously transfer of data streams. But this protocol requires implementation of complex special direct sequence spread spectrum coding schemes, and energy/battery inefficient systems such as pseudorandom code generators, modulation and demodulation circuits at the component bus interfaces, and differential signaling.

4: ARBITRATION ALGORITHMS

In this section we briefly present and discuss the key features of the arbitration

4.1. Round-Robin

A round-robin arbitration policy is a token passing scheme wherein fairness among masters is guaranteed, and no starvation can take place. In each cycle, one of the masters (in round-robin order) has the highest priority for access to a shared resource. If the token-holding master does not need the bus in this cycle, the master with the next highest priority who sends a request can be granted the resource. The advantages of round-robin are twofold:

Unused time slots are immediately re-allocated to masters which are ready to issue a request, regardless to their access order. This reduces bus under-utilization in comparison with a statically fixed slot allocation that might grant the bus to a master which is not going to carry out any communication. The worst-case waiting time for the bus access request of a master is reliably predictable (being proportional to the number of instantaneous requests minus one), even though the actual waiting time is not. The uncertainty on the actual bandwidth that can be granted to a master is the major drawback of this scheme.

4.2. TDMA

A time division multiple access scheme is based on the fixed allocation of a slot to each master, so that each of them is guaranteed fixed and predictable bandwidth. Unfortunately, high priority communications in a TDMA-based architecture may incur significant latencies, because the performance provided by this scheme strongly depends on the time-alignment of communication requests and slot allocation and therefore on the probability of dynamic variations of the request patterns.

4.3. Slot Reservation

This arbitration policy can be seen as a limit case of TDMA, in that only one master is periodically allocated a slot for the contention-free access to the bus. For the inter-slot time, we decided to manage the contention among the remaining masters in a round-robin fashion. Although this is not a conventional scheme for SoC communication architectures, we propose this policy to combine the advantages of the above mentioned schemes: one master is given priority in the competition for bus access (in terms of guaranteed fixed bandwidth), while all other masters can contend for the shared communication resource avoiding the risk of starvation.

4.4 Performance analysis of arbitration algorithms

Our objective was to stress the distinctive features of the considered arbitration algorithms so to come up with selection guidelines under different system workloads. To this purpose, we identified three scenarios at the application level, corresponding to three different communication patterns: mutually dependent tasks, independent tasks, and pipelined tasks.

4.4.1. Mutually Dependent Tasks

Let us assume a workload wherein one task is running on each processor and that the correct execution of each task involves synchronization with the other ones. In particular, let us assume that all tasks have to synchronize with each other at predefined points of the multiprocessor benchmark. In this case, system performance optimization translates to avoiding that some tasks reach the synchronization point much earlier or much later than the others, because this would generate idle waiting time for the unsynchronized task. An example thereof is represented by the bootstrap stage of RTEMS on the multiprocessor system. RTEMS selects one processor to act as a master and all other ones are considered as slaves, and they play a slightly different role in the booting operation. Each processor (master and slaves) at first independently initializes its private memory and hardware devices, and then synchronization has to take place at the shared memory. In fact, the master processor is in charge of initializing the shared memory and of allocating the structures for inter-processor communication. Then it starts polling the status variables of the slave processors, until they are all set-to ``ACTIVE," indicating that the slave processors have defined their own data structures in the shared memory. When this synchronization condition occurs, the master processor sets those variables to ``FINISHED," notifying the slaves that the initialization of the shared memory is over and that each processor can independently complete its bootstrap stage and load task

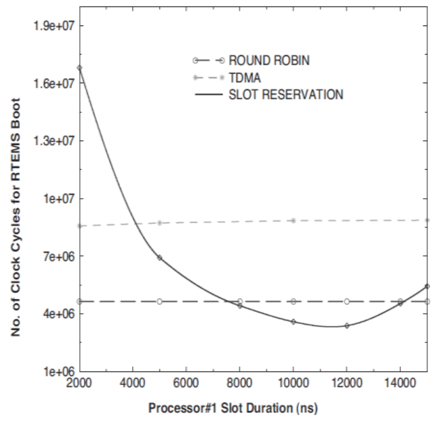

Fig.6 Execution time for bootstrap routine of RTEMS on the microprocessor platform

4.4.2. Independent Tasks

The second scenario we investigated makes use of a benchmark consisting of independent tasks, each running on a specific processor. This system workload does not have any synchronization point, nor does it involve inter-processor communication. The above scenario has been implemented on our simulation platform by executing the same matrix multiplications on each processing element. Matrices are initially stored in each processor's private memory, and the traffic generated on the bus is associated with read operations of matrix elements and to write transactions storing the results back in the memory. Tasks execution and consequent measurements are triggered once RTEMS has booted on all of the processors. The performance metric we select for this class of benchmarks is the average task execution time, given the independent nature of the tasks themselves. Our experiments have been carried out ranging the number of processors from 2 to 10, analyzing the scaling properties of the performance metric. Results relative to the tasks execution times are reported in Figure 7, for the cases of 4 and 8 active processors. When four tasks are running, we observe that round robin outperforms the other schemes. In fact, if we randomly select one processor and periodically grant it a slot for contention-free access to the bus, the improvement of its execution time translates to a relevant degradation of the performance for the other processors, and the average task execution time of the system increases. Though it is interesting to observe that a slot allocation of 9000 ns manages to balance the execution times of all processors, so that on average all tasks complete within the same time, similarly to what happens with round robin or TDMA, and this is the most efficient approach for this scenario. The relevant difference between the three arbitration algorithms is in the average execution time that can be obtained by each of them under the hypothesis of balanced task execution times. The balancing effect for slot reservation (achieved by properly tuning the slot duration) occurs at an average execution time which lies between that provided by round robin (the optimal one) and that provided by TDMA (worst case).The same effect can be observed with 8 processors, even though the average values increase and the gap between round robin and slot reservation decreases. One might guess that the performance of TDMA is likely to increase for smaller values of TDMA slot respect to those reported in Figure 7, so to reduce bus idleness. The answer to this question is reported in Figure 8, where the average execution time

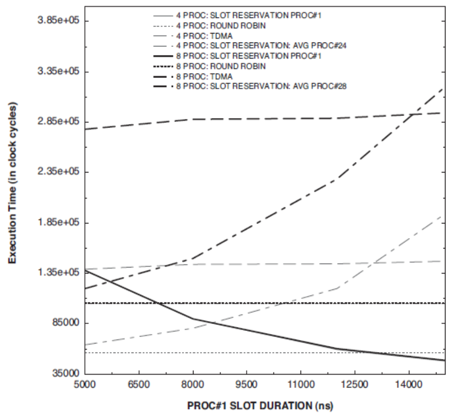

Fig.7 Execution time for a benchmark consisting of independent task

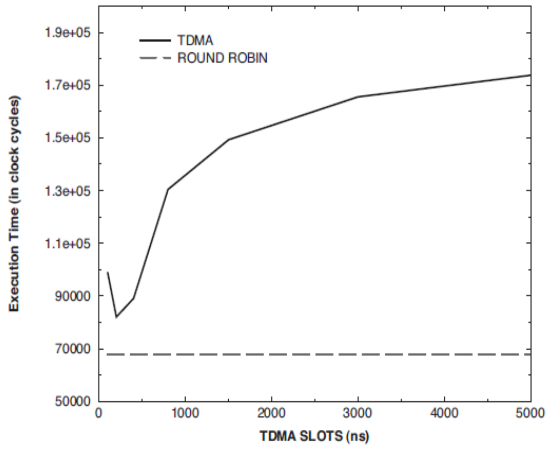

Fig.8 Comparison between the performance of Round robin and TDMA for small values of TDMA slot

4.4.3. Pipelined Tasks

A last scenario that is worth investigating is the one wherein the impact of arbitration policies on the throughput of a distributed signal processing application can be assessed. While in the previous subsection we analyzed a system workload wherein the traffic across the bus did not depend on inter-processor communication at all, but was only related to computation, now we want most of bus transactions to be related to communication among processors. We want to relate the performance of such a system to the way communication related traffic is accommodated on the bus by the different arbitration policies. To this purpose, we set up a multiprocessor system wherein different tasks execute in a pipelined fashion; with balanced computation workloads for all of the processors (they execute matrix multiplications). On top of the first processor, a task generates matrices that are handed over to the second processor of the pipeline. At each stage, the computation is carried out and the result transmitted to the next stage. In other words, the pipeline consists of couples of producer consumer tasks, and the communication occurs, at a high level of abstraction, by means of FIFO queues. The performance metric for this system is the throughput, denned as the number of matrices per second produced by the last processor of the pipeline (i.e., frame rate).Figure 10 shows the frame rate provided by the arbitration policies, changing the value of the slot duration for slot reservation and TDMA. While the performance of slot reservation is highly sensitive to the slot time, the performance of TDMA is almost independent of it. Surprisingly enough, although both the workload and the communication needs of the pipelined processors are perfectly balanced, slot reservation performs better than TDMA for a wide range of slot durations. This can be explained by looking at the performance of round robin, that is always much better than TDMA. Since our slot reservation algorithm implements a round-robin arbitration policy during inter-slot times, as long as the slot duration is much shorter than the inter slot time, the performance of slot reservation is dominated by the performance of round robin, while it becomes much worse when larger slots are used. Therefore, in Figure 10 only two experiments for slot reservation have been carried out, because they are sufficient to clarify the dependence of execution time as a function of the slot duration. Since the frame rate provided by slot reservation is always smaller than that of round robin, we can say that slot reservation is counterproductive in this case. In fact, there is no reason for guaranteeing a constant bandwidth to a single stage of a pipeline if the same bandwidth cannot be guaranteed to all stages. On the other hand, TDMA guarantees a constant bandwidth to all processors in the pipe, but its overall performance is lower than that of round robin. This fact can be explained only by looking at the hardware implementation of high-level interprocessor communication primitives. In our system, the producer consumer paradigm is implemented by means of the RTEMS message manager, which makes use of a communication protocol among tasks based on message queues. At the core of this protocol there is the inter processor communication mechanism seen in Figure 4. The procedure is initiated by the producer, which creates a global queue in its private memory, and writes messages to be sent in it. When the consumer is ready to receive a message, a notification is given to the producer by writing a request message into the shared memory and by generating an interrupt for the producer itself. The interrupt service routine of the producer reads the message from shared memory and assembles data to be sent in a message which is written back to shared memory. Finally, a write transaction to the consumer interrupt slave asserts an interrupt which allows the consumer to pick up its message from shared memory. In this context, TDMA poor performance can be explained in terms of its inability to support the communication handshake between the producer and the consumer, which is necessary for the hardware implementation of the high level inter-processor message passing. This handshake involves a Ping-Pong interaction between the two tasks, and is inefficiently accommodated in a TDMA based architecture, wherein only one processor is active during each slot. This results in a higher latency for the interaction respect to the round robin case, and this explains the poor performance of TDMA observed in the experiments. This low level implementation of message passing primitives made available by RTEMS to the applications involves a large overhead in terms of bus transactions. This overhead may result in a relevant system performance penalty, and derives from a mismatch between the software architecture and the underlying hardware platform. In other words, these two layers should be aware of each other to maximize system performance. Finally, we observe that the poor performance exhibited by TDMA is also related to the fact that it is inefficiently accommodated in AMBA based communication architecture. In fact, the ultimate objective of the AMBA bus protocol is contention avoidance, and the signals used by masters and slaves have to be seen under this perspective (e.g., HBUSREQ, etc.). On the contrary, TDMA would require a simpler communication protocol, as the whole contention management procedure is arbiter driven As a consequence, TDMA might outperform other arbitration algorithms in proprietary communication architectures. Despite the lower performance, TDMA-based arbitration is also attractive in many real-time applications where predictability is a critical requirement.

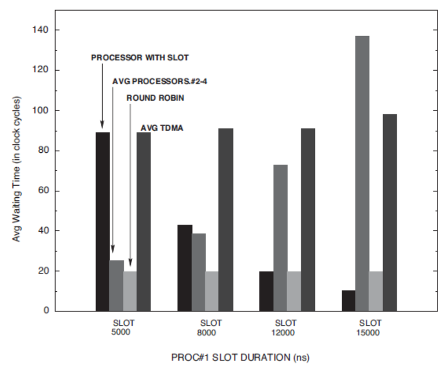

Fig.9 Bus access delays for benchmark with independent task

In fact,TDMA reserves a slot to each processor regardless of the current workload, thus making constant in time the bandwidth perceived by each processor, independently of the traf®cgenerated by the other masters. Consider, for instance, a system composed of 10 processor cores. If ®ve of the cores are used to implement the pipelined streaming application described in this subsection the frame rate achieved will be constant and predictable, independently of the traffic generated by the processors that do not take part in the pipeline (hereafter called external processors).Using round robin, the frame rate would be much better than that provided by TDMA when the traf®cgenerated by the external processors is negligible, but it would be strongly dependent on the overall workload, possibly becoming worse than that of TDMA when external processors perform memory/communication intensive tasks. No determinism is not acceptable in many real-time situations.

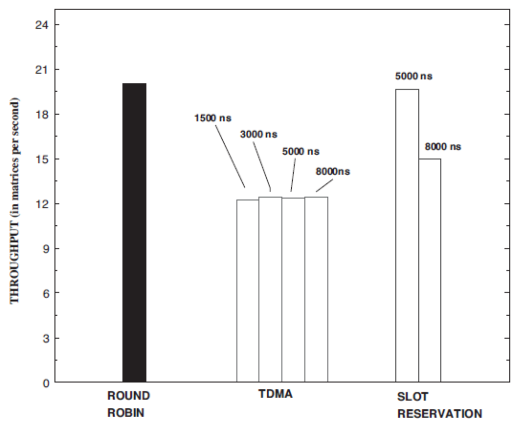

Fig.10 Throughput of system for different arbitration schemes

5: CONCLUSION

Beyond two traditional bus arbitration policies (round robin and TDMA) we consider another technique that periodically allocates fixed predictable bandwidth to time-critical processors (``slot reservation"). Three workloads are analyzed on our multiprocessor simulation platform (mutually dependent tasks, independent tasks and pipelined tasks), and some important guidelines for designers of SoC communication architectures have been derived:

1. The optimal bus arbitration policy is not unique, but strongly depends on the traffic conditions (computation-dependent, communication-dependent, etc.).

2. The software support for inter-processor communication plays a crucial role in determining system performance, as it has to be matched with the underlying hardware platform. High level communication primitives, although facilitating the programming step, could be inefficiently implemented on the available platform, degrading system performance.

3. There exists a trade-off between contention-avoidance bus arbitration policies (such as TDMA) and contention-resolution bus protocols (such as AMBA bus). Even though commercial standards provide degrees of freedom for performance optimization, the performance achievable by contention avoidance policies implemented within contention resolution protocols cannot be fully exploited, because of their different characteristics.

REFERENCES

1.F Massimo Conti, Marco Caldari, Giovanni B. Vece, Simone Orcioni, Claudio Turchetti:. “Performance Analysis of Different Arbitration Algorithms of the AMBA AHB Bus”.

2.Yu-jung huang, Yu-hung chen, Chien-kai yang, and Shih-jhe lin:”Design and Implementation of a Reconfigurable Arbiter”.

3.Archana Tiwari1 & D.J.Dahigaonkar: “Amba dedicated DMA controller with multiple masters using VHDL”.

4.Yu-Jung Huang, Ching-Mai Ko, and Hsien-Chiao Teng:”Design and Performance Analysis of A Reconfigurable Arbiter”.

5.Francesco poletti, Davide bertozzi,Luca benini: Performance Analysis of Arbitration Policies for SoC Communication Architectures.

6.Yu-Jung Huang, Ching-Mai Ko, and Hsien-Chiao Teng: “Design and Performance Analysis of A Reconfigurable Arbiter”.

7.Varsha vishwarkama, Abhishek choubey, Arvind Sahu: "Implementation of AMBA AHB protocol for high capacity memory management using VHD".

8.Massimo Conti, Marco Caldari, Giovanni B. Vece, Simone Orcioni, Claudio Turchetti: "Performance Analysis of Different Arbitration algorithms of the AMBA AHB Bus".

9.Vimlesh Sahu, Dr. Ravi Shankar Mishra, Puran Gour: " Design of High Performance AMBA AHB Reconfigurable Arbiter on system- on- chip".

10. Ashutosh kumar Singh, Vimlesh sahu, Kush soni: "Design and Implementation of High Performance AHB Arbiter for on chip Bus Architecture".