Infinite Technology Corporation

"Configurable Bus Arbitration for Communication Between Multiple Concurrent IP Blocks"

Neal S. Stollon neal@itc-usa.com
Robert L. Veal les_veal@itc-usa.com
George Landers george@itc-usa.com

Abstract: This paper addresses issues and approaches to communication of concurrent intra-chip IP blocks over multi-channel shared bus fabrics and IP based methods of arbitration of both "IP block to IP block" and "IP block to shared resource" (ex. on-chip memory) bus resources. Arbitration issues of competing IP blocks for access to a common IP resource is addressed, along with discussion of deterministic design approaches including; scheduled, priority based, and peer-to-peer arbitration. Hardware system tradeoffs of arbitration schemes for datapath processing and arbitration and arbitration control issues for chips with distributed bus structures are presented. Approaches to bus arbitration for different functional requirements implemented as arbitration logic as IP blocks in configurable bus architecture are discussed with reference to Infinite Technology's IP-based RAMA (Reconfigurable Array and Memory Architecture) chip design.

1. Introduction

Robust communication between multiple independent IP blocks is key to effective usage of cores from different sources within a SoC environment. In many applications, IP blocks with varying and dynamic intercommunication requirements may be required to operate autonomously and asynchronously (herein defined as concurrent operation), and subsequently with differing bus interfaces and control requirements. Incorporating robust, low latency and deterministic methods of arbitration of common bus resources to these IP cores is part of the necessary SoC infrastructure to allow such blocks to be integrated and to communicate efficiently.

2. Intrachip IP Arbitration and Data Transfer

DataBus and MemBus Differences

Data busses and memory busses are primary resources for data movement and communication between intra-chip processor IP blocks (Figure 1). Processor IP blocks may consist of dedicated or instruction programmable logic with busses operating at differing speeds and with diverse latencies and throughput based on block performance and architectural requirements and limitations. When a single controller coordinates all IP resources, bus arbitration and data transfer between IP blocks (processing elements and memory) can be addressed in a deterministic manner under a unified instruction word control [1]. This is contrasted with the transfer of data between concurrent processors, where inter-communication access must be arbitrated. In addition, data transfer between concurrent processors presents a very different arbitration scenario compared with operations between concurrent processor(s) and memory blocks.

In addressing processor/memory arbitration, architectural considerations of multiple concurrent processor to memory arbitration interfaces include:

- Since memory does no (independent) processing work, it is an "always available" resource when properly shared between processors.
- Processors contend for access among themselves for bus and memory resources; therefore arbitration is a "processor access driven" operation.
- Memory is idle unless being accessed (read or write) by a processor. Processors always request access, memory never does. The relationship between processor and memory is "master/slave".
- Arbitration allows memory block bandwidth to be owned by a single processor at any given time. Memory blocks typically cannot be concurrently shared between concurrent processor resources.
- Memory operations are repetitive, have low latency and are well defined. There is no variable latency delay in memory operations once processor inputs have been made available.
- When a processor gains memory access, the memory is by definition ready for communications operations for the full duration of the processor ownership.

Memory Bus arbitration schemes reflect this processor to memory relationship by focusing on relative priorities of processors in gaining access to memory resources by addressing processor access requests and the sending of processor access acknowledges for memory access.

Processor to processor communication (within concurrent processor arrays) operates under more bi-directional and diverse sets of priorities including:

- Concurrent processors operate under a "peer to peer" relationship, where no processor has an inherent priority access to common resources such as memory blocks.
- Processors can be data sources or destinations to other processors on dynamic and time varying intervals. Latency and duration of processor-to-processor communications can vary significantly over time and by application.
- Whether communicating with other processors or not, a processor is typically concurrently performing internal computational work. This computational work can have variable latency, intervals, and complexity that can preclude or delay processor-to-processor communications.
- Processor computational work can compete with processor communication for critical resources (Execution unit memory, I/O controller prioritization).

Comparing Processor-Memory and Processor - Processor Arbitration

For concurrent processors, arbitrated access to “passive” common resources (i.e. memory port availability) can be determined external to the processor in a memory arbiter. For multiple memory blocks, a separate arbiter process is required for each independent memory block, and for multi-ported memories, each memory port (Fig. 2).

However, processor availability (for inter-processor data transfer) is only determined by coordination of the transmitting and receiving processor. Inter-processor arbitration becomes a handshaking process by which arbitration must be resolved between competing transmitting processors. Processor1 and processor3 arbitrate for access to a requested receiving processor (2). Receiving processors (requested processor2 and processor4) arbitrate for access to requested processors (including processor1) with data to send. Implementing the mutual agreement that both selected processors (1 and 2) are available and able to send and receive/utilize data (and agreement on the amount of data being communicated between them) can be handled with varying levels of sophistication and complexity.

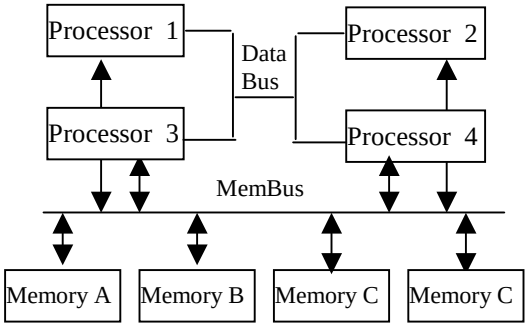

Figure 1. Data and Memory Busses in Concurrent Multi-Processor Architecture

An architectural advantage of multi-processor systems is that latency and “deadtime” in data movement can be minimized by arbitration over distributed resources. Since transmitted processor data is transient, (if not utilized (or stored) the receiving processor, it can be lost and must be recomputed), distributed resources increase the probability that an acceptable receiving processor is available. Since re-computation can be computationally expensive for streaming data or high-speed datapath applications, having distributed resources are especially useful in datapath operations. In comparison, transmission of memory data is static, and can typically be stalled or resent with low processor overhead (address recomputation) if necessary.

Relative complexity of memory and data bus arbitration can be seen by comparing states involved in (sending or receiving) processor accesses for processor/memory and processor/processor communication.

For processor/memory communication, there are only two states of interest:

1. Memory bus port is available
2. Memory bus port is not available

If the memory port is available (or can be freed to be available), the memory is always ready for processor communication. The only contention is between competing processors for access to a mutually requested memory port.

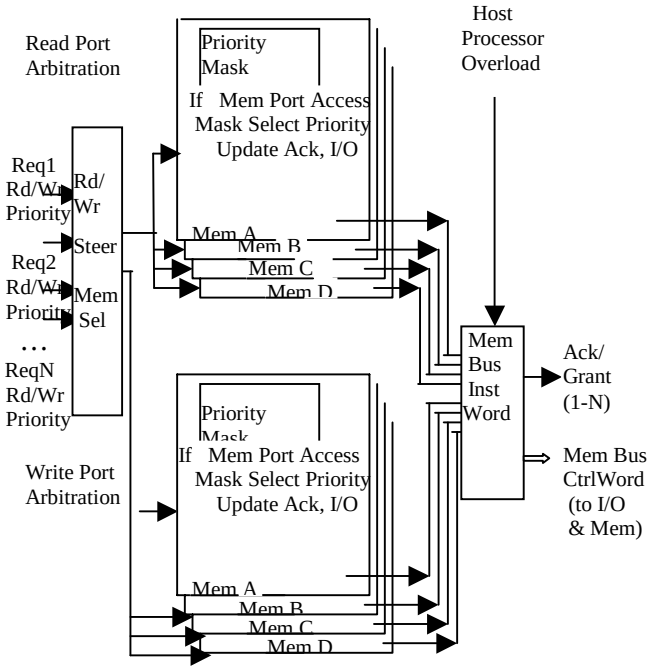

Figure 2. Memory Bus Arbitration of a (4 Block) Memory Array with Independent Write/Read Ports

In the case of multi-processor (processor1/processor2) communication, where processor1 initiates a request to communicate data with processor2, (and processor2 in turn

indicates that it is ready for communication with processor 1) there are additional possible states for consideration:

1. Processor2 IO port is not available, processor2 not ready for operation (Processor1 must wait or do other tasks until port/processor2 is available)
2. Processor2 IO port is not available, processor2 involved in other log term IO tasks (Processor1 must retarget data to work around port/processor2)
3. Processor2 IO port is not available, processor2 idling (Processor1 must get processor2 attention to make its IO port available).
4. Processor2 IO port is available, processor2 not ready for operation (due to internal processing for example)
5. Processor2 IO port is available, does not want processor1 data (processor2 has its own request (not to processor1) ongoing)
6. Processor2 IO port is available, processor2 ready for operation (set bus connection, start communicating)

Approaches to Multi-processor Arbitration:
Arbiter implementation tradeoffs include size (max. number of ports supported under a single arbitration scheme), performance (latency, algorithm complexity) and resources dedicated to arbiter functions. Multi-Processor arbitration schemes can be defined depending on type of robustness and performance required.

1. Basic - Processor1 determines if processor2 port is available, establishes connection regardless of whether processor2 is ready. (This is memory arbitration mode, in that processor2 is treated as a "slave" IO)

2. Simple - data is only sent when both transmitting and receiving processor request each other. Unrequainted requests are ignored; the arbiter must address how processors align on requests without excessive wait time

3. Less Simple - "matchmaking" arbiter communicates to processors that a request has been made. Requested Processors decide whether to send acknowledges and start communication

4. Complex - "traffic cop" arbiter processes competing requests based on information on processor current and future communications availability/readiness

5. More Complex - processors sort among competing requests and establish a handshaking protocol. In this case, arbitration may be handled as part of the process function rather than by external arbitration logic.

3. Bus Arbitration in RAMA

RAMA™ (Reconfigurable Array/Memory Architecture) is a 2nd generation Reconfigurable Arithmetic Datapath (RAD) architecture, developed by Infinite Technology Corporation, which integrates RISC, Datapath Execution Unit (EXU), and Memory IP blocks to perform extremely high-speed arithmetic calculations in DSP applications. RAMA implements a massive processing capability that may be applied as Intellectual Property to a custom ASIC. The datapath core element of this chip is a RADarray™ processing engine, based on Infinite Technology's RADcore™ [2] signal processor architecture. RAMA RADArrays may be configured with between multiple RADcore datapath engines under RISC host processor coordination; each RADcore performs processing work independently or in conjunction with other RADcores over a high speed Databus (RADbus) and a lower speed memory bus (MemBus). On-chip memory and bus arbitration is integrated to optimize the RADarray performance and throughput. The RAMA architecture is seen in figure 3.

In RAMA, Memory Read and Write bus fabrics (MemBus) are independent, with separate address and data channels running through dedicated read and write bus fabrics. Memory requests from the processor and external memory ports are supported in addition to RADcores and internal memory.

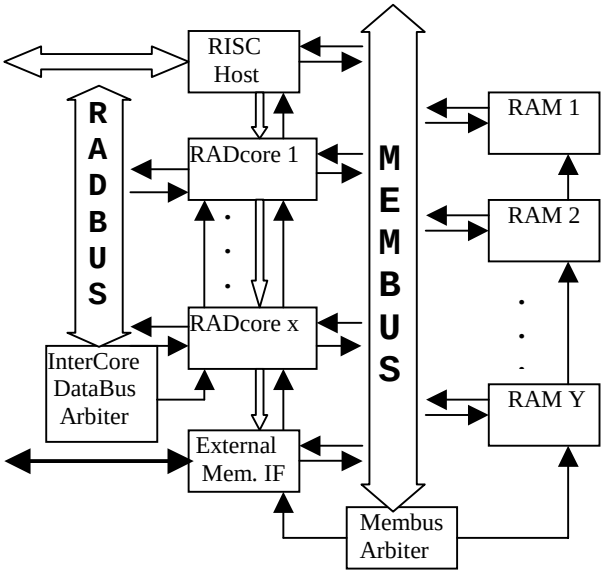

Figure 3. RAMA Block Diagram

Memory Bus Arbitration
The RAMA Memory Arbiter controls memory access between RadArray processor IO, RISC processor IO, on-chip memory IO, and external memory IO by granting ownership of a memory port (Read or Write) based on requests from RAD, processor or external memory IO. Ownership is based on a priority mask based algorithm where each IO requesting memory access is assigned a priority value for a given memory port. Each memory port has its own priority mask, so arbitration is determined independently for each Memory Port. The arbitration scheme is kept deterministic by assigning a one priority of ownership (max, high, low) per Memory IO for each Memory Port mask (i.e. the mask can not have two IOs both having max priority with an assurance of equal

priority access). Arbitration may be overridden by acknowledge (grant) and Connection Word signals provided by an auxiliary register from the host processor.

In RAMA Memory arbitration, dual -ported read and write memory access is handled by two Arbiters (Figure 2). The Arbiters allow independent RADcores to request and take ownership of an on-chip Memory port using a request/grant IO architecture. The Arbiter insures that only one processor IO can own a given memory port at a time. The arbitration is autonomous (no processor support is required) under normal operating conditions. Priority masked interrupt scheme supports ownership control with variable priority access for each IO. Priority masking allows dynamic QOS (Quality of Service) based memory allocation.

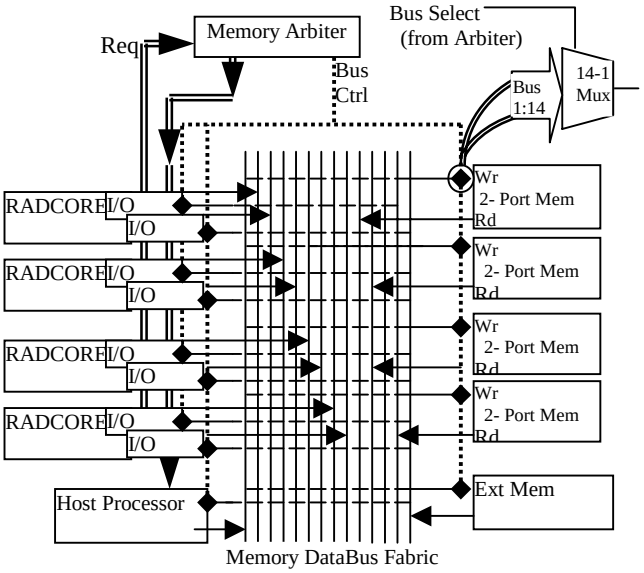

Figure 4. A Processor to Memory Bus Fabric with Arbitration Driven Connectivity

The Arbiters set the RADcore/Memory paths (Data and Address) based on the memory port ownership. A single RADcore Memory I/O may have ownership of a Read and a Write port based on different Read and Write priorities. These ports need not be in the same memory block. A Round Robin arbitration scheme of equal priority IO requests statistically minimizes data starved IO operation. When a processor IO Request goes Idle, ownership of Memory Port is automatically updated. Implementing parallel arbitration logic for each memory and each port allows low latency ownership update (3-5 cycles). A programmable provision allows a forced priority interrupt of internal memory ownership by external memory or on-chip processor. A host override provision allows the on-chip processor to take over control of arbitration.

Inputs to the Memory Arbiter are:

1. Requests from a Radcore (Write or Read) IO for a particular memory port. Ownership is by type such that a processor Read Port only requests ownership of Memory Read port and processor Write Ports only requests access to Memory Write ports.
2. An Immediate_priority signal for each memory block allows immediate updating of memory port ownership based on priority. If Immediate_priority is not set, then ownership does not change until a processor IO gives up ownership of a memory port by changing its request input.
3. A set of updateable priority masks, which order the prioritized access of a processor IO to a given memory port in cases of multiple requests for memory port ownership.
4. Auxiliary data from the host processor to override the arbitration process with predefined grants and bus connection word controls in arbiter override mode.

Outputs from the Memory Arbiter are:

1. Acknowledge (ACK) signals, sent to all processor IOs to indicate whether the IO is currently granted ownership of a requested memory port.
2. Connectivity Word (CW) signals sent to Read port input muxes of all processor IO and memory blocks to select what memory bus channels are connected.

Multi-Processor Handshaking Arbitration in RAMA

RADarray inter-core communication is distributed over multiple data channels, running through a dedicated bus fabric, as shown in Figure 5. This allows multiple inter-core I/O blocks to communicate concurrently. Multi-Processor Handshaking Arbitration algorithms, as implemented in the RAMA DataBus Arbiter, control a full speed bus fabric (RADbus) access between the RADcores within a RADArray. Control of RADcore IO access occurs by a process of simultaneously granting ownership of processor Input and Output ports based on RADcore IO requests and processor availability. Ownership is granted if there is an overlapping request from a Processor1 Write port for a Processor2 Read port and a mirrored request from the same Processor2 Read port for the Processor1 Write port. Since mutual requests are required for setting up RADcore to RADcore processor connections, there is no need for priority masks or encoding rules. As in the memory arbiter, arbitration may be overridden by acknowledge (grant) and RADbus connectivity signals provided by the host processor.

The handshaking algorithm used by the multi-processor arbiter uses two sets of signals in determining available and required processor port access. The algorithm is updated on a dynamic basis based on changes in inputs.

1. Requests from Read or Write IO of a RADcore processor block for access to a particular IO port of another processor. Ownership is segmented such that a

processor Output (Read) Port may only request ownership of another processor's Input (Write) port and processor Input (Write) Ports may only request access to Output (Read) ports of another processor. Once a request is acknowledged, then connection does not change until an I/O gives up connectivity by changing its request input.

2. A set of host processor based arbitration registers that allow the host processor to take override ownership of the core-to-core communication.

Multi-Processor Arbiter outputs include acknowledge outputs to each processor IO port that define the I/O to I/O connectivity and synchronize the initiation and conclusion of inter-processor communications. The acknowledge signals are updated on a clock-by-clock basis.

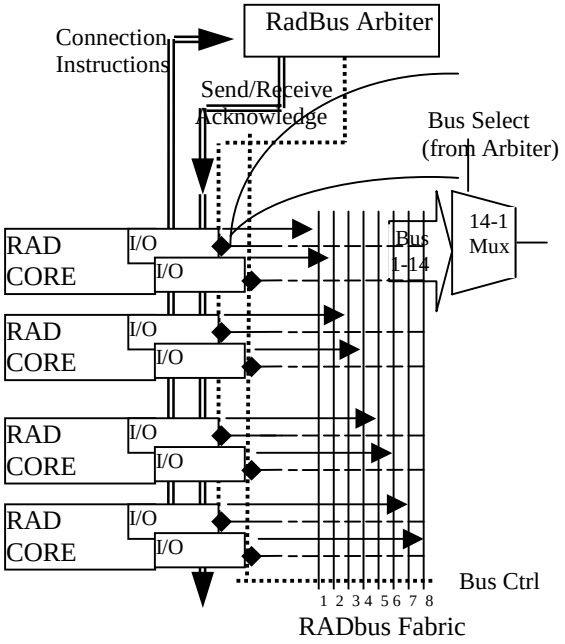

Figure 5: A Processor-to-Processor Bus Fabric using Handshake arbitration

4. System Considerations

Specific bus interfaces and arbitration schemes will differ based on types of processors being used and on applications and goals of the architecture. Some considerations however must be part of any arbitration decision.

- Concurrent processors should be able to effectively communicate at full bandwidth using a standard IO architecture.
- Instruction based requests to bus access should be processed expediently
- Arbitration should ensure only one IO can source data on a given bus to another I/O port at one time.
- Arbitration should be autonomous (no external processor support required) under normal operating conditions (however options for host processor arbitration override can be useful)
- In multi-port memory operations, arbitration should set Data and Memory channel (Data/Address) priorities based on a memory port ownership scheme.
- Independent and simultaneous Read and Write arbitration of multi-ported (Read and Write) IO blocks is desirable for communication throughput.
- Masked or otherwise controlled prioritized support of each IO port is useful in allowing QOS (Quality of Service) based bus bandwidth allocation.
- Arbitration logic should allow Read and Write requests for given IO to be prioritized differently. Parallel arbitration logic for each port allows low latency ownership updates for the processor I/O block.
- Arbitration should have provisions for forced priority interrupt and redirection of bus ownership by allowing override control of arbitration in cases of deadlocks or other stalling conditions
- Arbitration should be factored on an equal basis with other processor decisions, and distributed into the architecture as appropriate.

For datapath operations, where arbitration is over multi-channel data and memory busses, more robust arbitration approaches provide advantages for concurrent multi-processor communication. In architecture where IP processing blocks have very different throughput or are controlled under different clock domains, arbitration must address additional IO constraints such as elastic storage and buffering in inter-block communication. Reference 3 addresses such an architecture that integrates processor and CPLD blocks with very different IO characteristics [3].

5. Summary

The combination of increased usage of IP from multiple sources and the impetus to efficiently integrate this IP into SoC solutions makes the development of robust inter-IP block bus solutions increasingly critical. Arbitration is an increasingly critical consideration in IP communication. This paper discusses design approaches and issues in bus arbitration developed with the goal of providing a robust communication between processor and memory IP blocks. Infinite Technology's RAMA architecture exemplifies some of the considerations in using multiple IP based concurrent processor blocks and the requirements for arbitration, both between processors and with common memory resources.

References:

[1] N. Stollon, "A Reconfigurable Bus IP for Modular Function Integration", Proceedings of IP99/Systems on a Chip Conference April 1999

[2] N. Stollon, G. Landers, T. Lahutsky "A High Performance Reconfigurable Signal Processor with Distributed IW Architecture", Proceedings of DesignCon99/IP World Forum February 1999

[3] N. Stollon, R. L. Veal. " A Reconfigurable Datapath and Programmable Logic Based System-on-a-Chip Architecture", Proceedings of DesignCon2000/IP World Forum February 2000

Author Biographies:

Neal Stollon is the Manager for Reconfigurable System-on-a-Chip efforts and the Senior Member of the Technical Staff at Infinite Technology Corp. Dr. Stollon has previously worked in diverse areas of design, automation, and program management at Texas Instruments and DSC Communications. Dr. Stollon has a Ph.D. in EE from Southern Methodist University and is a licensed P.E. (Tx).

Robert L. Veal is Vice President of Marketing and Sales, and leads the business development activities of Infinite Technology Corporation. Mr. Veal is a 26-year veteran of the electronics industry, working in areas of sales, marketing, and product and general management at Texas Instruments and Compass. Mr. Veal has a M.A. in Physics from Harvard University.

George Landers is Vice President of RAD Architecture and Development at Infinite Technology. He has 35 years experience in the semiconductor industry in product development and technical marketing in areas of memory, reconfigurable logic and signal processing. At AMD, Mr. Landers was Manager of PLD Strategic Marketing and is a co-inventor of AMD's MACH product line. Mr. Landers has a BSEE from University of California at Los Angeles.