Communication Mechanisms for Parallel DSP Systems on a Chip

Joseph Williams, Bryan Ackland, and Nevin Heintze
Agere Systems
Circuits and Systems Technology Laboratory
Holmdel, NJ 07733, USA

Abstract

We consider the implication of deep sub-micron VLSI technology on the design of communication frameworks for parallel DSP systems-on-chip. We assert that distributed data transfer and control mechanisms are necessary to manage many independent processing subsystems and software tasks. An example of a parallel DSP architecture is given and used to demonstrate these mechanisms at work. We show the similarity of these mechanism and those used in large scale computing networks.

1. Introduction

Rapid advances in semiconductor technology have had several implications on VLSI circuit design. The ability to integrate 10's of millions of transistors on a single die means that entire systems can be integrated on a chip. A system-on-chip (SoC) allows designers to design solutions with greater performance, reduced power and lower cost. However, there are many challenges to implementing a SoC with high performance.

One of these challenges is communication over long distances on a chip. Until recently the clock speed of a chip was largely determined by the number of gates in the critical path of the design. As the feature size of process technology continues to shrink, the performance of interconnect has not scaled as rapidly as the transistor switching speeds. Critical paths are becoming dominated by delays due to interconnect, especially when the logic is spread across the chip [1].

As a result, tightly coupled chip architectures are increasingly impractical, and SoC designs have evolved towards collections of loosely coupled largely autonomous subsystems, each performing a different function such a processing, memory and I/O. These subsystems must be efficiently integrated and coordinated, and they must share chip resources. This requires an interconnection network and associated communication mechanisms to allow the different subsystems to transfer data and control information. Critical to such designs are communication mechanisms with distributed control. Moreover, to reduce the complexity of programming and managing the system, we want to hide as much as possible of the low-level detail of these mechanisms.

The evolving architecture of the SoC is not unlike a large data network. A data network also has different subsystems such a processing (computers), memory (file servers), and I/O (fiber optic links). It also requires an interconnection network (routers interconnected with fiber optic links) and communication mechanisms (network protocols). Many of the techniques developed for data networks can be applied to a SoC. In this paper, we will present some of the architectural features of a SoC platform designed for high performance signal processing. We will summarize some of the architectural features of the platform. We then give an example of a data transfer that demonstrates communication mechanisms that are used to manage data transfer and control the execution of applications. We will demonstrate the similarity between the distributed processing and control of the SoC and a typical data network.

2. Daytona DSP architecture

Daytona is a programmable DSP platform designed for communication infrastructure applications [2]-[4]. Two example applications are 3G wireless basestations and DSL access multiplexors. Figure 1 gives an illustration of a typical Daytona DSP SoC. Multiple processing elements (PEs) are interconnected with a pipelined split-transaction bus called the Daytona Bus. The PEs can range from programmable RISC and DSP processors for flexibility to fixed function hardwired accelerators optimized for performance and power. The Daytona Bus allows PEs to exchange data and control information as well as access shared resources such as global memory and chip I/O. Daytona uses a sophisticated memory hierarchy to keep most accesses local to the PE and minimized the amount of bandwidth required on the Daytona Bus. Programmable priorities can be used to minimize latency of critical accesses across the bus such as blocking memory requests.

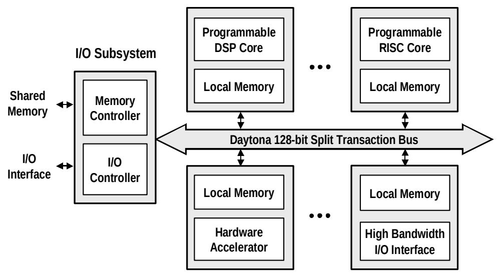

Figure 1. Daytona parallel DSP architecture.

A controller called the I/O Subsystem manages data transfers between PEs and global resources. Daytona provides various hardware mechanisms to support communication such as cache coherency, DMA, and semaphores. Multiple tasks are scheduled dynamically by an embedded RTOS [5].

3. Packet Switched I/O Subsystem

Sharing resources amongst subsystems on a SoC is especially challenging when the resource is limited. For example the number of I/O pins in an IC package is limited and often must be shared among the various SoC subsystems. The system architecture must allocate I/O bandwidth to each subsystem in an intelligent way to insure good system performance. The I/O performance requirements of each subsystem could vary significantly. For example, one subsystem could require large bandwidth but be relatively insensitive to latency, while another could require little bandwidth but will suffer significant performance loss if the latency is too large. In addition, different applications running on the same SoC could have varying I/O requirements. Programmers need flexibility to allocate I/O dependent on the application needs.

Daytona uses the I/O Subsystem to manage shared off-chip interfaces. An illustration of the I/O Subsystem and how it interfaces with the rest of the system is shown in Figure 2. It is effectively a memory switch that interconnects off-chip shared memory, off-chip shared I/O, and the Daytona Bus. These subsystems are all memory mapped. A 32-bit source and destination address indicates which subsystem data is coming from and going to. Data is segmented into fixed length 32 byte payloads and a header is appended to the data. The interpretation of the 32-bit source and destination address depends on the particular subsystem. For example the least significant 24-bits of the address is used by the shared memory to specify

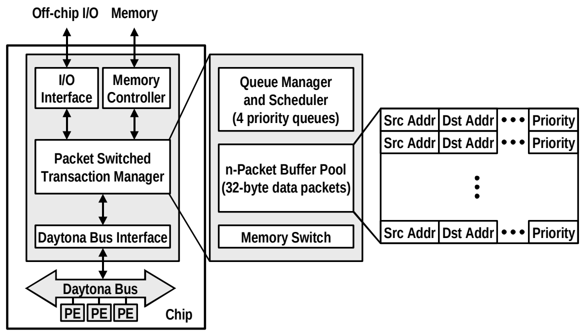

Figure 2. I/O Subsystem

the physical address of the data to read or write. In addition, several other fields of the header give semantics for the data transfer such as which bytes in the 32-byte payload are valid. One interesting field to note is the 2-bit priority field that is used to manage the quality of service (QOS). All higher priority requests are serviced before a lower priority request is serviced.

The I/O Subsystem has a buffer that stores pending transactions. Logically the buffer has four queues, one for each transaction priority. Each queue is in fact a linked list and the I/O Subsystem has a pool of memory used for building linked lists. When a data packet arrives with a given priority it is added to the beginning of the appropriate queue. An entry is removed from a pool of free entries and added to the beginning of the queue. The queue manager handles the task of linking and removing entries from the queues. Each clock cycle the scheduler services the highest priority non-empty queue. Once the transaction is completed the entry from the serviced queue is returned to the free pool. A memory switch physically transfers data from one subsystem to another. For example, the memory switch is used to transfer 32-bytes from the Daytona Bus to the memory controller.

Since the packet switch transaction manager can buffer multiple transactions and service them simultaneously, it is possible to exploit parallelism. For example when a memory access is being performed in one memory bank, a second memory access can be performed simultaneously in another memory bank. An important property of this architecture is isolation of data flows. Slow data accesses (maybe because an I/O port is busy) do not block other time critical accesses. This fine-grained distributed communication between subsystems (and software tasks) allows them to share resources efficiently.

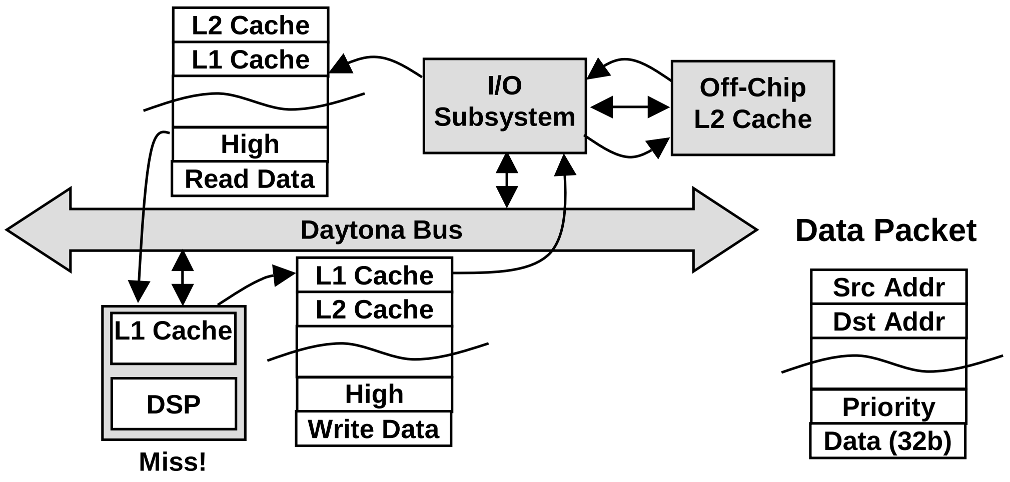

Figure 3. Data transactions to service a data cache miss

Figure 3 illustrates an example of a transaction through the I/O Subsystem. A processor in the system has a data cache miss and requires that the cache line be refilled from the shared memory. The cache controller in the processor generates a transaction on the Daytona Bus and uses an address that is mapped to the shared memory (off-chip L2 Cache). The cache is write-back so a modified line being replaced must be written back to the L2 cache.

The Daytona Bus controller of the I/O Subsystem snoops (observes) the bus for addresses that it must service. When it sees the transaction addressed to the global memory it queues the request into the highest priority queue. Since the processor cannot continue execution until this data is returned, making the transaction high priority improves performance, since a non-blocking request can be delayed. If the queue already has at least one entry then the transaction will not be serviced until the existing ones are serviced. Once the I/O Subsystem services the transaction, the data packet is forwarded to the global memory controller. The global memory controller performs a read of the global memory and returns the data to the I/O Subsystem. Once again the transaction is queued to the highest priority queue, except that the destination in now the processor that requested the data originally. Once the queue is serviced the data is transferred through the Daytona Bus to the processor.

The data transfer through the I/O Subsystem is not unlike the forwarding of cells in an ATM network [6]. An ATM network is also designed to perform fine-grained distributed communication and to isolate data flows between different end-points.

4. Conclusions

As transistor budgets continue to grow, the design complexity of SoCs will rapidly increase. Careful problem partitioning and parallelization of applications is necessary to manage VLSI implementation. Centralized control will become prohibitive due to the dominance of routing delays. Distributed processing and control will be the only way to achieve good system performance. The trend is very similar to the shift of centralized computing in a mainframe environment towards distributed computing in a data network. As we have demonstrated many of the lessons learned in that transition can be applied to the design of SoCs.

References

[1] J. Williams, et al., “The Implementation of Two Multiprocessor DSPs: A Design Methodology Case Study”, ISSCC 2001.
[2] C. Nicol et-al., "A Single-Chip 1.6 Billion 16-b MAC/s Multiprocessor DSP", IEEE Custom Integrated Circuits Conference, May 1999
[3] J. Williams, et al., “A 3.2GOPS Multiprocessor DSP for Communication Applications”, ISSCC 2000.
[4] C. Nicol, et al., “A Single-Chip, 1.6-B MAC/s Multiprocessor DSP”, IEEE JSSC, pp. 412-424, March 2000.
[5] A. Kalavade, J. Othmer, B. Ackland, K.J.Singh, "Software Environment for a Multiprocessor DSP", Design Automation Conference, June 1999
[6] R. Perlman, “Interconnections: Bridges, Routers, Switches and Internetworking Protocols”, Addison-Wesley, 1997