SoC-Based Microcontroller Bus Design In High Bandwidth Embedded Applications

White Paper

Abstract

32-bit embedded designs increasingly require real-time control of highbandwidth data streams over a network. At the System-on-Chip (SoC) level, especially, it is crucial to move data and control information in a deterministic and non-contentious manner. It is also important that operations be directly under the programmable control of the system developer, which has not always been the case in bus-based SoC designs.

Designers and chip providers frequently borrow from board- and system-level architectural techniques to create SoC designs with minimal design time and development costs. Since appliances such as printers and PDAs seldom require deterministic, real-time responses, traditional solutions work very well for such applications.

But in many of the new networkconnected embedded designs, traditional bus architectures simply cannot handle the bus-sharing demands for high bandwidth and intense data flow. This is particularly true of applications such as Human-Machine Interface (HMI) displays, point-of-sale terminals, color printers and copiers, network-enabled projectors, surveillance cameras and others. Devices managing the combination of real-time data, a display and network activity need a deterministic way to share bus bandwidth.

A number of alternative approaches based upon on-chip serial interconnects similar to serial fabrics, crossbar switches and packet-based busses are being researched. But until these new approaches are perfected, time and cost constraints dictate that ways be found to modify the sharedbus architectures borrowed from board design to accommodate the deterministic and real-time requirements of the new embedded 32-bit network-connected designs.

Traditional SoC Bus Advantages and Disadvantages

SoC developers have been reluctant to give up the generic shared bus borrowed from the board-level world because it reduces the specification and validation effort in the design cycle and makes SoC top-level integration almost as simple as plugging expansion cards onto a back plane. By using generic buses, developers are free to concentrate on higher-level decisions.

ARM® Limited's use of a generic bus in the Advanced Microcontroller Bus Architecture (AMBA) has allowed licensees to focus on the application at hand and move quickly to market.

Microprocessors, DMA controllers, memory controllers and other higher performance blocks are connected via the AHB. Lower-performance blocks such as UARTs, General Purpose Input/ Output (GPIO) and Timers are suited for connection to the APB.

But many high-end embedded applications targeted by ARM-based SoCs require that while dealing with the deterministic, real-time requirements of the application, they are also able to access a high bandwidth network environment.

Such applications require an SoC that issues control signals, collects data, and moves data across the network in real time. Depending on the nature of the network and its bandwidth requirements, this can stretch the capabilities of existing SoC-bus architectures to their limit.

For example, a high-end network-connected embedded application may be handling video bit streams from a camera or graphics from a server to a printer via an Ethernet connection, and at the same time may be updating a local LCD display with precise requirements as to scan, refresh and update cycles. Working with the external LCD, the controller must know the precise number of bytes to be sent over the bus, the order in which the data is sent, and the specific time slots in which the data must be presented to the display and in what order. It is also necessary to constantly feed information to the LCD for updating.

Adding Burst Mode DMA to the Peripheral Bus

The traditional approach to the peripheral bus in AMBA-based designs assumes low performance applications for ARM core-based embedded devices. But today's devices frequently have one or more applications that must operate at high bandwidth without cutting off lower bandwidth peripherals from access to bus resources. This issue is particularly problematic in a peripheral-rich design such as the NS9750 or NS9360, which, in addition to providing high speed I/O for 10/100 Ethernet, LCD, external DMA, and PCI, provides low speed support for USB, I²C, four multifunctional serial modules (selectable as UART or SPI capable of up to 11 Mbps in the synchronous mode), 50 individually programmable GPIO pins, an IEEE 1284 peripheral port, and sixteen general purpose timers or counters, each with its own I/O pin.

In traditional implementations of the APB, the low transfer rates of communications peripherals such as UARTs are more than adequately handled by the inclusion of FIFOs, which allow several bytes to be transferred tothe interface before the processorneeds to intervene and accessthe APB. But in many of the high-end embedded applications summarized in the article, one or more of these peripherals may have high bandwidth requirements that require immediate access to the main high-performance bus via the APB/AHB bridge.

One way to allow the peripheral bus to operate in such a burst mode is to simply replace the APB bus with a burst-mode peripheral bus (Digi's BBUS) that has not one, but four bus masters each with burst mode support (Figure 3). One bus master is a DMA engine that has 13 channels supporting 12 USB end-points. A second bus master is a DMA engine that has 12 channels supporting the four serial modules with eight channels each as well as the 1284 port. The third bus master, the BBUS-to-AHB bridge, has a DMA engine with channels that provides an access to the AHB system bus. The fourth bus master is a USB host module. Additionally, this DMA engine has two separate dedicated DMA channels to support external peripherals connected to the external memory bus. To facilitate burst mode conditions, each internal DMA channel moves data between the system memory and the BBUS peripherals in a fly-by mode, while both external DMA channels use memory-to-memory style transfers.

The shared bus concept is not sufficient to meet such requirements in an SoC. In a typical AHB design all of the primary resources on that bus are bus masters, which means that when the bus is free, they can requisition the bus for the time necessary to accomplish a task. But in an ARM-based SoC there is no direct control by the programmer over how much of the bus resources they acquire when in control of the bus.

There are a number of ways a shared bus architecture can prioritize these operations: daisy chain arbitration, centralized parallel arbitration, distributed arbitration by self-selection or collision detection, and bus arbitration with multiple bus requesters. But when the designated master takes over the bus, other operations are pushed aside. There is no mechanism by which multiple resources can gain access to the bus to the degree necessary to satisfy the application requirements without impacting the ability of other important operations to deliver deterministic and real-time responses.

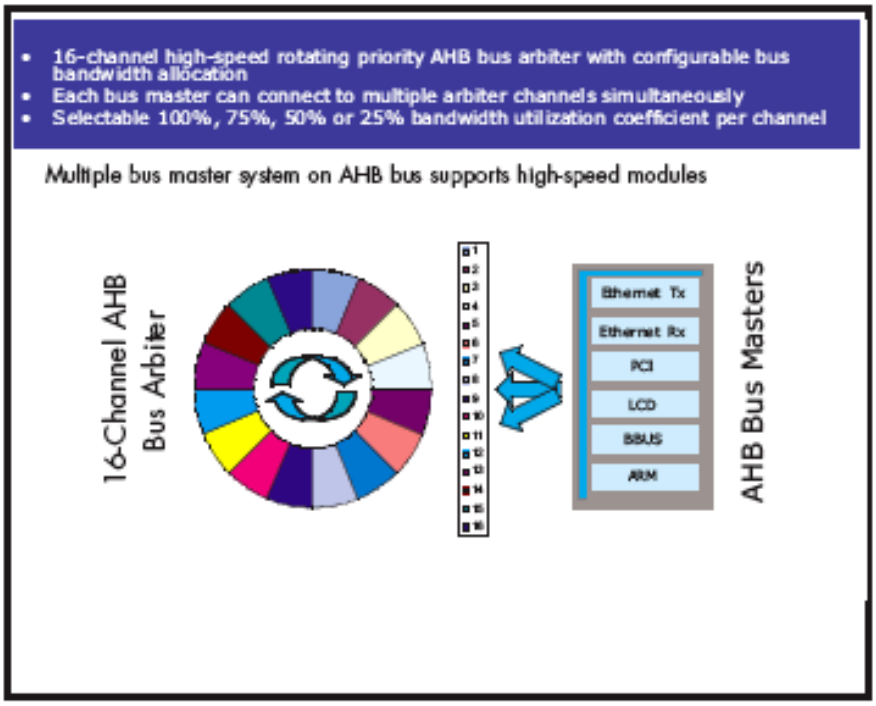

Figure 1: NS9xxx AHB-bus bandwidth control system

One common technique used within the AMBA environment to deal with such situations is the use of arbitration channels. If there are six bus masters, the bus is designed with six arbitration channels. But rather than dedicate each channel to a particular master, on-chip arbitration logic assigns the channels based on the number of masters requesting access to the bus. If four masters are requesting the bus, the six channels are divided among the four, ensuring that each master gets equal access to the bus.

However, this does not solve the basic problem of how to assign enough bus bandwidth to accomplish a particular task. If one of the operations needs three channels and the other operations cumulatively only require two, each will be assigned an equal amount of the available channel space. The result: several channels will be underused, others not used at all, and some channels will be overloaded, impacting the SoC's ability to respond to events deterministically and with sufficiently low latency.

The Solution: A Programmable Bus Bandwidth Control System

What is required is a programmable bus bandwidth allocation scheme that gives a particular master the bus allocation it needs at that particular moment, and allocates the remaining bus space to the other masters who may also be asking for access to the bus. And because this may change over time, some mechanism is needed to reallocate bus resources on a regular basis.

Digi has developed a new bandwidth control system to replace the one used with the AMBA architecture (patent granted, patent #6, 826, 640 B1). The system is based on the use of a 16-slot rotating priority bus arbiter (Figure 1) that incorporates a set of programmable pseudo-random or rotating priority cache replacement algorithms. In the case of the NetSilicon NS9750 (Figure 2), for example, rather than contending for allocation of six channels on the AHB, the six bus masters share access to a 16-slot bus allocation scheme. Via dedicated registers in the system control module, the system developer now has three ways to allocate bus resources within the SoC.

At the highest level, every time a particular bus master issues a request for access it would be responded to in the order of the request, until all six masters are polled. Based on needed bandwidth, each bus master is then assigned a specific number of slots and has exclusive access to those allocated slots. For example, if four slots in the NS9750 are assigned to the CPU, four slots to Ethernet, four slots to the BBUS Bridge (see sidebar), three slots to the LCD, and three slots to the PCI/Cardbus, this arrangement would be re-evaluated as needed by system software during the system's operation, which can be keyed to the number of AHB bus cycles. If the situation is unchanged on the next evaluation cycle, the setup remains as before. If it has changed, a new set of bus master slot assignments is negotiated.

For even more precise control of bus resources, this cyclical arbitration scheme provides two additional levels of programmability: the amount of bus bandwidth allocated to the ARM CPU and the degree of bandwidth utilization on each of the 16 slots.

Rather than being allowed to take control of all of the bus resources when it is the bus master, the NS9750's ARM926EJ-S core is by default allowed only 50 percent of the total bus bandwidth, or eight of the 16 slots. This ensures that the other five bus masters have at least 50 percent of the bus between them at all times. However, under direct programmer control it can be directed to release some of its allocation to another bus master or to take control of additional slots for just that bus arbitration cycle, or for any number of cycles the programmer determines is required.

The programmer can also select a 100, 75, 50 or 25 percent bandwidth utilization coefficient per slot. This is done by controlling when and in what sequence access to each slot is allocated: for a 25 percent coefficient, that particular slot would be polled only once every four cycles; 50 percent, every other cycle; and 75 percent, three out of four cycles.

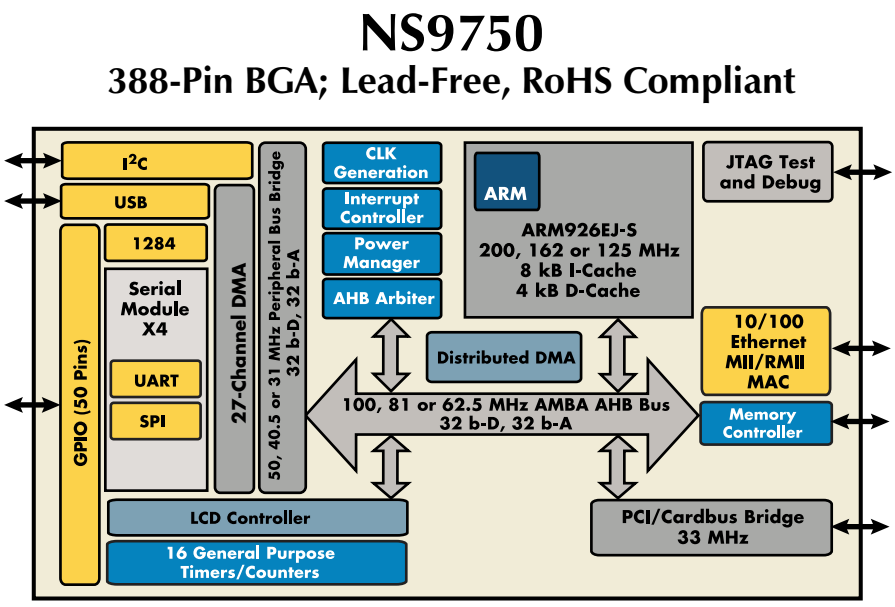

Figure 2: NS9750 block diagram

Programming the Rotating Bus Arbiter

The various options can be specified by the programmer via several registers contained in the system control module. The first is a 16-entry Bus Request Configuration (BRC) register. Each entry in this register represents a bus request for a master and a grant slot.

Each request/grant slot is assigned to only one bus master at a time, but each bus master can be connected to multiple request/grant slots at the same time, depending on the bandwidth requirement of that bus master. When multiple channels are assigned to one master, these channels should be evenly distributed among the 16 channels. Each request/grant slot has a two-bit Bandwidth Reduction Field (BRF) to determine how often each slot can arbitrate for the system bus - 100, 75, 50 or 25 percent. The BRC gates the bus requesting signals going into a second 16-entry Bus Request Register (BRR). As a default, unassigned slots in the BRC block the corresponding BRR entries from being set by any bus request signals.

A fourth register is used to store which bus master has data waiting for transfer to the AHB, while a fifth register is used by the programmer to assign weighted values to each bus request and grant slot assigned to a particular bus master.

Cyclic Arbitration in Action

In the example earlier, where the LCD on a particular arbitration reassignment schedule requests additional bus access, the programmer can specify that the LCD should be given priority, based on the nature of the data stream it must handle. If the programmer decides that ten slots need to be assigned specifically to the LCD controller, the six slots left would then be assigned to the other bus masters on the original arbitration schedule. In this situation, the LCD controller would have available ten times the bandwidth it normally would have and ten times the bandwidth of the other masters to handle the load dictated in this particular situation.

This capability would be crucial when the device is streaming data over the Ethernet connection at the same time the LCD screen is being refreshed. The LCD needs to be refreshed in a timely, deterministic manner, uninterrupted by the demands of the Ethernet.

In a typical AMBA bus architecture, if the LCD requested the bus it would have to wait until the Ethernet master released it, no matter what the refresh requirements. With the new cyclic programmable arbitration scheme, the programmer can "back off" the Ethernet transfer, allow it to stream data out at a lower but still acceptable rate, guaranteeing that the LCD would get its appropriate level of refresh, so that the screen will not go blank.

Where the requirements of the LCD's timing and bandwidth for guaranteeing an active display are very precise, the Ethernet protocol requirements are much more forgiving of lower rates of transmission. But it is not acceptable for the data stream to be stopped, which would have been the case if the LCD master had control of the bus and only gave it up when the refresh chores were completed.

Advantages of NS9xxx Architecture

In high bandwidth embedded systems, programmers must optimize throughput of multiple parallel data streams. The patented Digi solution described above gives programmers a powerful tool to meet specific requirements of their applications.

As sub-micron process geometries continue to evolve, system on chip integration will incorporate more and more diverse functionality into a given device. This will require that RTOS and DMA architectures provide the systems engineer with flexible control over his system resources. As IP networking capability continues its penetration of embedded electronics, Digi is committed to providing the hardware and software services to make this intergration successful.

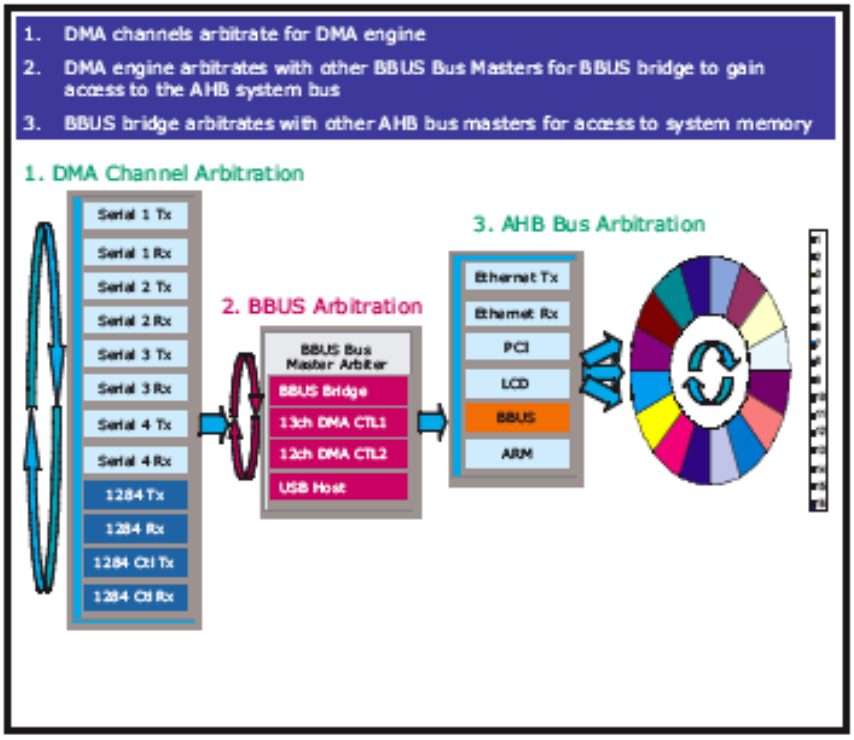

Figure 3: NS9xxx bus architecture

91001317
B1/1115

DIGI INTERNATIONAL WORLDWIDE HQ
877-912-3444 / 952-912-3444 / www.digi.com

DIGI INTERNATIONAL FRANCE
+33-1-55-61-98-98 / www.digi.fr

DIGI INTERNATIONAL JAPAN
+81-3-5428-0261 / www.digi-intl.co.jp

DIGI INTERNATIONAL SINGAPORE
+65-6213-5380

DIGI INTERNATIONAL CHINA
+86-21-50492199 / www.digi.com.cn