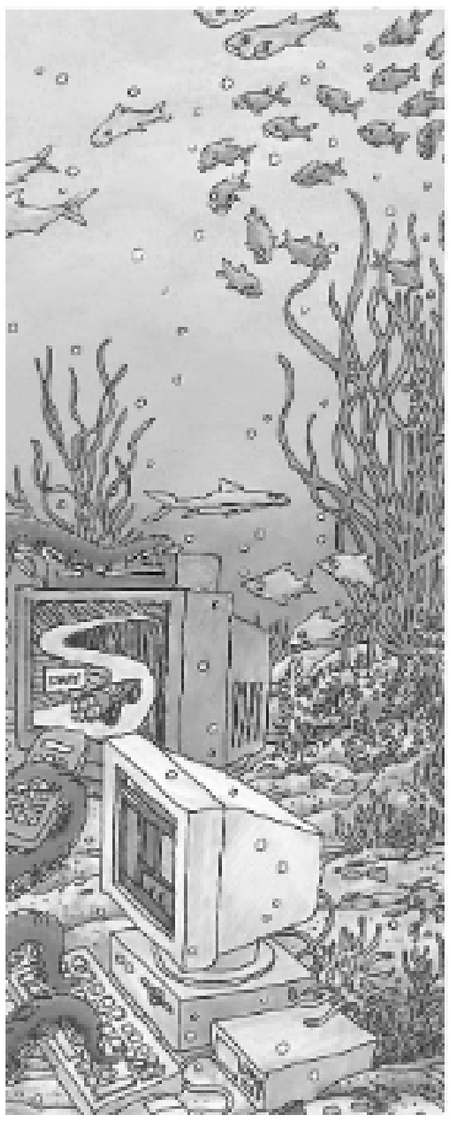
AMBA: Enabling Reusable On-Chip Designs

David Flynn

Advanced RISC Machines



AMBA helps designers of embedded microcontrollers achieve first-time-right designs that maximize reusability.

In 1995, Advanced RISC Machines released its Advanced Microcontroller Bus Architecture in response to input from key semiconductor licensees. AMBA's goal is to help designers of embedded CPU systems meet challenges like design for low power consumption and test access. Because input for AMBA came from designers of ARM-based microprocessors, ARM was also able to develop a solid design rationale and evolve an architectural design that would address the most common problems.

In this article, I describe some of AMBA's design methodology and provide a set of specifications that will aid designers in making detailed comparisons with other buses.

AMBA defines both a bus specification and a technology-independent methodology for designing, implementing, and testing customized, high-integration embedded controllers. The first range of cached cores with native AMBA system bus interfaces are due for release in late 1997.

Design rationale

AMBA is ARM's response to the problems and difficulties first-time customers have reported when designing around the ARM processor bus. Feedback from technical experts, semiconductor licensees, lead engineers, developers from key customers, and representatives of design groups helped identify the underlying issues, which continue to show up in design reviews:

- *Ad hoc design.* Designs tend to end up with ad hoc bus and control logic. Designers relish the opportunity to create custom on-chip systems, but they often create bottlenecks, especially when the semiconductor licensee is new. Because the bus interface on the ARM6 and ARM7 cores is extremely flexible, designers unfamiliar with the processor may inadvertently create inefficient or unworkable designs.
- *Design portability and reusability.* A side effect of ad hoc design is its lack of portability and reusability. DMA controllers for video or audio subsystems, for example, are not easily ported because the main state machines are inextricably merged with the processor bus controller. Similarly, the external memory interfaces that support narrow memory tend to expose the complex byte packing and unpacking to the central system's controller state machine. This makes reusability very difficult. If the designer does not separate the memory interface from the system design, it becomes harder to partition memory resources to minimize system cost.
- *System reset and clocking.* Designers are responsible for implementing contention-free, on-chip tristate buses and safe, low-power, gated clocking schemes. However, they seldom have time to review their designs in depth for potential pitfalls. The critical paths tend to revolve around the decoding of address ranges, and many designers fall into the trap of squeezing out wait states at the expense of reduced overall CPU clock frequency.
- *Power consumption.* The ARM CPU's low power consumption often drives the decision to design in an ARM core. However in the face of time-to-market pressure, designers of new product families often make decisions that sacrifice power elsewhere. The problems are both on chip, with many peripherals attached to the processor bus, and at the all-important off-chip external memory interface.
- *Test support for CPU macrocells with many I/Os.* It is no longer desirable or

0272-1732/97/$10.00 © 1997 IEEE

feasible to mandate a test mode that takes either a serial-scan-based or a multiplexed core isolation approach. Designs with only 22 visible address lines and a 16-bit data bus cannot justify the cost of developing a special test-pattern set for the core in each new device.

- *Infrastructure portability.* The porting of third-party real-time operating systems requires a more modular and defined system infrastructure for ARM-based microcontrollers. To port a microkernel, for example, the designer must target both the processor instruction set and the interrupt controller environment, as well as the basic counter/timer functionality.

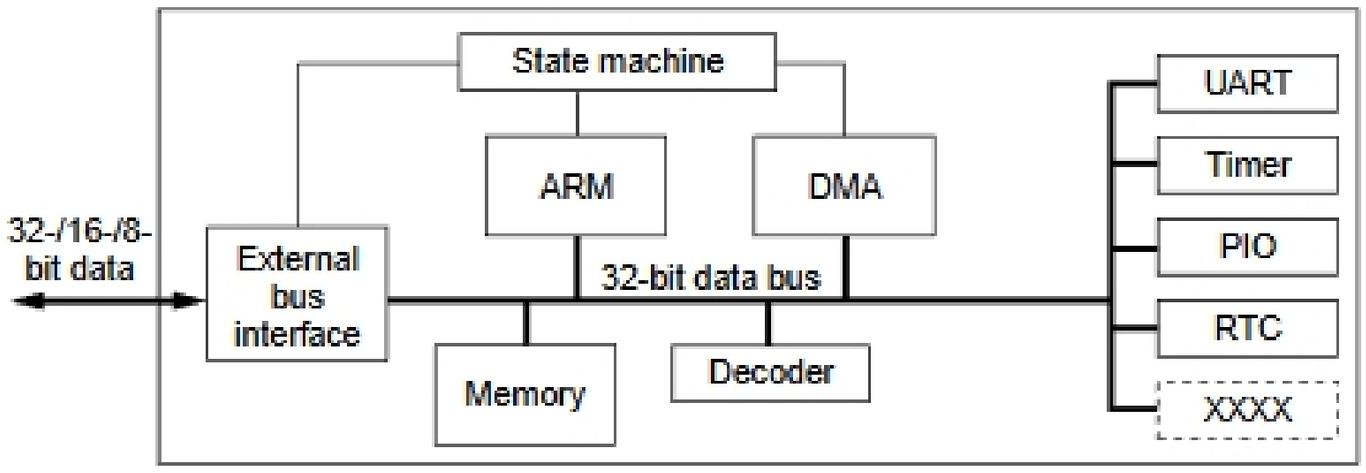


Figure 1. A typical first design of an ARM microcontroller using an ad hoc approach. The common disadvantages that affect design quality and reusability are the result of coupling all peripherals to the processor bus, and of using a centralized state machine. Manufacturing test access may use either multiplexer block isolation or scan and built-in self-test techniques.

Figure 1 shows typical aspects of an ad hoc ARM design. In such designs, the state machine tends to be complex and customized around the external and internal bus sequencing and control.

Goals. AMBA addresses the issues just listed by incorporating several important features:

- *Partitioning for modular design.* Its methodology for embedded processor design encourages both a modular and first-time-right system design and accelerates product migration by supporting module reuse.
- *Interface protocol, clocking, and reset.* AMBA specifies a flexible, low-overhead bus interface and clocking structure to the core CPU macrocells. This simplifies system design because interface protocols are clearly defined to have one or more bus masters—components that initiate bus transactions while enabled. The AMBA Terminology and Specification box on p. 22 explains bus masters in more detail.
- *Support for low-power designs.* By partitioning high- and low-bandwidth devices within the system, AMBA ensures energy-efficient designs, which fit well with low-power CPU cores.
- *On-chip test access.* AMBA integrates an optional on-chip test access methodology that reuses the basic bus infrastructure. This helps make the external test of embedded CPU and peripheral macrocells more efficient.
- *Support of multiple development platforms.* This allows cycle-accurate benchmarking and hardware prototyping. It also features a set of generic reference peripherals that make it easier to port real-time kernel software.

Motivations. Before starting work on a new bus design, the technical marketing group at ARM examined existing buses. We concentrate on off-chip buses such as PCI, generally suitable for open systems; on-chip buses, such as Motorola's InterModule Bus, generally suitable for closed systems with a particular microprocessor; and emerging standards, primarily in Europe, to support multiple microprocessor families.

Off-chip bus standards, including S-Bus, MicroChannel, and PCI, make it easier to create board-level and backplane (motherboard) modular systems.[1] However, because the clocking strategies in such buses (synchronous or asynchronous) are typically tuned for interchip communication, the bus specification must deal with handshaking and physical and electrical characteristics. It must accommodate the worst-case manufacturing skews and tolerances for the spread of best- and worst-case components (typically different silicon processes and technologies). In such open system buses, the overhead of variable wait states and time-outs is generally high.

With on-chip buses the manufacturing and process variations apply to the entire system and bus components. Motorola's InterModule Bus, or IMB, was derived from the 68000 bus interface, so the interrupts and CPU-specific signals effectively become the system bus signals. Such a bus standard is well suited to basic uniprocessor microcontroller designs.

However, neither the off-chip bus standards nor microprocessor-specific, on-chip derivatives apply RISC principles to the bus protocol and infrastructure required at each bus module (especially slaves). These principles are important to enforce a design that minimizes logic and emphasizes speed and efficiency. Relative to PCI, for example, AMBA requires no distributed time-out support at every node and no parity support on the data bus.

AMBA targets battery-powered, low-cost embedded applications—an application area not specifically addressed in the 1991 Open Microprocessor Initiative standards activities.[2] It also supports built-in manufacturing test access for deeply embedded processor cores. ARM essentially cut the main system bus to the basic clocking, reset, and memory bus interface functions that generic processors and DMA controllers require. AMBA treats processor-specific interrupts and configuration signals as orthogonal to the bus specification. Such signals connect directly to the appropriate I/O subsystem rather than via the system bus. AMBA also treats the arbiter request and grant signals as point-to-point signals between individual bus masters and the arbitration unit itself.

AMBA terminology and specification

Bus cycle. The basic unit of one bus clock period; for protocol description, the bus cycle is a falling edge to falling edge transition. Bus signal timing references the bus cycle clock.

Bus transaction. A read or write transfer of a data object of given size, which may take one or more bus cycles, terminated by a completion cycle from an addressed device. System reset also terminates transactions to guarantee that the bus is initialized.

A bus master asserts a LOCK signal to the bus arbiter to make multiple transactions indivisible. In a nonmultiplexed implementation, the address and control information is broadcast in parallel with the data read or write operation. A multiplexed implementation requires the broadcast of an address cycle at the start of each new burst transaction that precedes the data transfer.

Bus burst operation. An address transaction plus one or more data transactions. Bursts may be of arbitrary length and may be broken down into smaller packets by

- the *arbiter*, which may constrain burst lengths to meet critical interrupt or DMA timing latencies; or
- the *slave*, which may be able to handle only short bursts.

Bursts must use sequential incremental addresses and fixed direction and size throughout.

Bus master. Component that initiates bus transactions while it is enabled (see the multiple-master support section). The master generates address and control information, including the direction and size of data transfers, plus burst indication when multiple sequential data transactions are required.

The transactions initiated by the master are completed by acknowledgment from the addressed target device. A slave device may request the master to rebroadcast the next address. This feature allows page boundaries, for example, to be handled to prevent burst address wrapping.

In multiplexed bus implementations, addresses are resynthesized externally to the bus master in a shared-address incrementer. In nonmultiplexed implementations, the address is broadcast for every burst access from the bus master on a full 32-bit parallel address bus.

Examples of bus masters are

- CPU,
- dedicated digital signal processor unit,
- DMA (multichannel) controller(s),
- test interface controller for external test access (see main text), and
- diagnostic controller (for remote debugging).

Bus slaves. Components that respond to addresses within a decoded region of the address map and perform bursts of read or write cycles on demand. A slave may assert a WAIT signal to delay access using a synchronous bus transfer protocol. A slave module may communicate at the bus interface only when selected. The selection process depends on the system decoding (described later); typically only one slave is selected at a time. The bus protocol defines the basic module selection timing; a slave response must then be generated by, or on behalf of, the addressed slave. Slave devices typically require only a subset of the 32-bit address bus; these addresses may be from a parallel address bus, the system bus master, or a shared-address latch/incrementer.

Examples of slave devices are

- memory banks,
- external bus interface (which typically supports 16- or 8-bit-wide external memories),
- basic memory-mapped peripherals, and
- customer-developed macrocells.

A bus transaction, initiated by a bus master, must be terminated by a response when the slave device has com-

Modular design

AMBA evolved from ARM's internal bus development work. We generated an initial bus specification from the basic ARM6 RISC core and the cached macrocells (ARM610 family with cache, MMU, and write buffer).

We then formalized a basic synchronous master/slave bus protocol, in which bus masters (typically CPUs, DSPs, or DMA controllers) request the bus and a bus-arbitration unit grants access to a single bus master. The master can initiate read or write transactions with an address-mapped slave (such as memory or peripheral I/O registers).

The address-space decoding and arbitration priorities are not constrained in ARM embedded controller designs (apart from the basic requirement to have ROM-resident vectors at reset). The decoder and arbiter functions require well-defined interface protocols, but designers should be able to modify and extend them to suit specific applications.

In the bus infrastructure, we wanted to retain features that complement the ARM CPU designs with minimal hardware and complexity, but still support high-bandwidth, multiword burst transactions. The infrastructure emphasizes the need for a simple slave interface and a low gate count, in particular.

Figure 2 shows an AMBA-based implementation of a microcontroller that is functionally similar to the one in Figure 1. The test interface controller (TIC) and bridge between system (ASB) and peripheral (APB) buses are described in detail later.

The ARM6 and ARM7 bus families use a 32-bit-wide system memory bus, which can transfer data on every clock cycle from a memory subsystem. They internally generate addresses and control signals in the half clock cycle preceding the memory access. In addition, two "next transaction" burst access signaling flags indicate whether the next address is related (sequential) or unrelated (nonsequential) to the current access, or unused (corresponding to an idle bus cycle).