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.

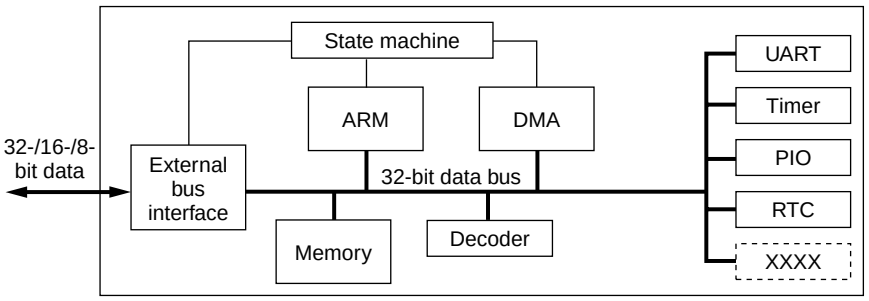


Figure 1. A typical first design of an ARM microcontroller using an ad hoc approach. The common disadvantages that affect design quality and reusability are the result of coupling all peripherals to the processor bus, and of using a centralized state machine. Manufacturing test access may use either 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).

AMBA terminology and specification (continued)

pleted the data transfer. The slave must provide the response when selected by the decoder.

As mentioned in the bus master section, the bus transaction protocol allows slave devices to break up bursts of transactions over page or peripheral address boundaries.

The centralized decoder is typically responsible for generating an ERROR response when no valid transaction slave device is addressed, or when an access violates protection or permission restrictions. In the bus protocol, this appears simply as a default null slave that always causes a bus error when selected.

System decoder. Slaves are essentially memory mapped within a global 32-bit address space. The system decoder provides

- address decoding,
- global control of the bus that acknowledges the slave response transaction,
- top-level memory decoding (in hierarchical decoding schemes),
- generation of system bus transaction responses,
- management of burst-length address boundaries, and
- (optional) basic memory protection for control of supervisor/user mode access.

Distributed address decoders typically handle the decoding of external memory and internal peripheral registers, which gives the map a finer granularity.

The critical path in many systems is the address decoding from a (new) address to the selection of the module. AMBA supports early signaling of sequential burst-mode activity. Designers can then bypass the address-decoding path if they do not cross system-dependent address boundaries.

In systems with a low clock frequency (many pagers, current mobile phones), there is sufficient time to decode and select slave modules without adding wait states for nonsequential accesses. In higher bandwidth applications, it pays to add a wait state for nonsequential addresses to maximize the sequential burst-mode data bandwidth.

Delegating this function to the system's top-level decoder lets designers create both bus master and slave modules with the same bus interface, regardless of whether they are to be used in fast or slow designs.

Multiple bus master support. The bus supports multiple bus masters, all of which communicate in a unified manner with slave devices.

The arbitration scheme is centralized. Masters must request the bus using a centralized arbiter; the arbitration protocol for each master is strictly defined, but the arbitration priority is left to the designer.

By specifying the arbitration protocol rather than the arbitration scheme itself, all decisions about priorities and DMA latencies rightly become a system design issue to suit the application constraints. Designers may implement a fixed-priority scheme or a more complex round-robin or adaptive arbitration mechanism, depending on the real-time requirements.

The arbitration is pipelined by one clock cycle to allow changes in bus mastership without incurring dead bus cycles in handover phases. Request and acknowledge handshake signals support the arbitration signaling for each bus master. The arbiter must observe the state of the LOCK signal from the currently active bus master before granting access to another bus master.

Data transfer width. The processors replicate byte-write information across all four bytes and halfwords across both high and low halves of the 32-bit bus. A two-bit size encoding supports the 8-bit bytes, 32-bit words, and 16-bit halfwords and reserves an extra code for future 64-bit or multiword extensions.

The bus master indicates the size of data transactions at the start of a burst; slaves decode appropriate byte enables as required.

The byte alignment is natural—an addressed byte or halfword must be provided on the appropriate byte lanes. Word accesses disregard the low two address bits; half-

Memory subsystems unable to complete the bus transaction in a single clock cycle either stretch the clock phases or add a synchronous wait state. As always, the designer handles the bus driver enables to cope with back-to-back processor read and write accesses.

To support a multiple-master bus, we added a basic state machine to the CPU core. Its role is to manage the interface for the arbitration request/grant handshake. This addition makes it easier to support the natural concurrency of bus masters that can continue processing until they need to communicate on the system bus.

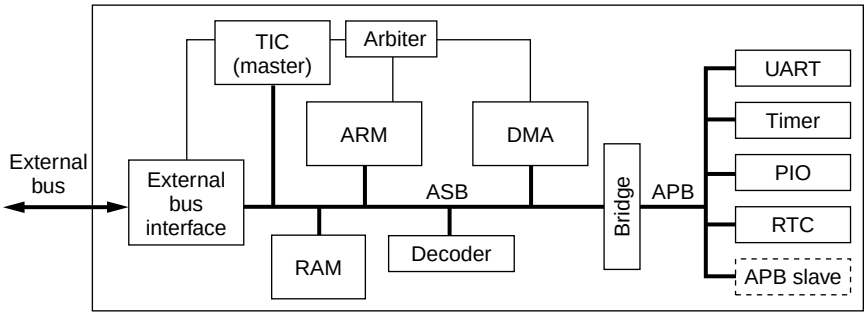


Figure 2. AMBA-based design of a microcontroller with functionality similar to that of the microcontroller in Figure 1. An AMBA-based microcontroller encourages modularity and design reusability by distributing the control state machines into independent bus masters and partitioning high- and low-bandwidth components on independent buses.

AMBA terminology and specification (continued)

word accesses disregard the lowest address bit.

ARM systems may be configured for either little-endian or big-endian byte addressing. The choice of configuration is left to the designer; the CPU macrocells provide a configuration input or coprocessor control flag.

Data transfer handshake. Data transfers require an explicit acknowledge to complete transactions: the bus master initiates the access by driving the direction, size, and start address of a burst. The slave must acknowledge with success (operation complete), failure (bus access terminated after some error), or busy-wait (transaction not complete).

The three-bit response field encodes

- *Done*—transfer successfully completed,
- *Wait*—transfer not completed,
- *Error*—access violation or nonexistent slave, or
- *Retract*—slave unable to complete, so master must retry.

To handle clean protocol reset without building watchdog timers into every slave device, an orthogonal reset mechanism clears both the active master and slave without overdriving the status bus signals.

The retract mechanism simply allows a slave that cannot complete to release the bus for a cycle so the bus arbiter can rearbitrate for any higher priority bus masters requesting the bus.

Designers implement the retract support only in a complex slave device that may suffer from arbitrary wait-state activity—such as PCMCIA card interfaces—and simply allow the bus to be rearbitrated. To build fully generic, deadlock-free systems, the basic mechanism must be extended to observe which shared resource slaves block which bus masters.

Burst transfers. Both the bus masters and bus slaves handle burst transfer signaling. The master indicates whether a burst is to be continued or terminated; the slave indicates whether a burst can be continued or not. The signals are pipelined; they are broadcast in the cycle preceding the transaction to which they apply. The bus master indicates if the next cycle is

- an *address* cycle, address broadcast next cycle;
- a *nonsequential* transaction, new address, potentially new direction and size; or
- a *sequential* transaction, the address is incremental to the last transaction address.

The bus slave interface indicates if the slave can continue burst access in the next cycle, or the slave requests to break up burst access in the next cycle.

There is also a signal from the currently selected master to the arbitration unit (BLOK, described in the signals section, next).

Signals. The advanced system bus (see main text), or ASB, has 77 signals, all of which have a "B" prefix. These are grouped into common signals that all modules, all bus masters, and all bus slaves share. Global signals are

- Clock (BCLK) and
- Reset (BnRES).

Bus master signals are

- Burst transaction indicators (BTRAN[1:0]),
- 32-bit Address (BA[31:0]),
- Control (BWRITE, BSIZE[1:0]—the direction and 32-/16-/8-bit width),
- Protection (BPROT[1:0]—User/Privileged, Opcode/Data access), and
- arbiter LOCK request (BLOK).

The master/slave data transaction bus is a 32-bit data bus (BD[31:0]). The slave response is a 3-bit response (BWAIT, BLAST, BERROR).

The system decoder monitors the bus master signals and provides the slave module selection, typically in a centralized implementation to minimize gate count. This generates a DSELxxx point-to-point signal for each slave module.

When AMBA test is present, each master also includes a slave interface that supports the response bus handshake signals. The address and control signals then become inputs where required.

The advanced peripheral bus (see main text), or APB, signals have a "P" prefix:

- Address (PA[a:0], where *a* is typically less than 31),
- Access strobe (PSTRB, active enable referenced to BCLK),
- Access control (PWRITE—high for write, low for read),and
- Bidirectional data (PD[d:0], where *d* is typically 7, 15, or 31).

The APB signals are generated by an APB bridge, which is simply a slave module on the ASB. The global clock and reset signals (BCLK, BnRES) provide timing synchronization and initialization references.

Peripheral decoding is provided by a local distributed decoder function that transmits a point-to-point PSELxxx signal to each peripheral.

Interface protocol, clocking, and reset

Reset and clocking strategies are fundamental to guaranteeing that all levels of bus protocol are free of contention and lockup. This is especially important in low-power designs, in which, typically, an ARM processor may have both on-chip clock synthesizers for voltage control oscillators and clock-gating power-down modes.

To guarantee contention-free power-up, the bus master's state machine has a defined reset protocol and autonomously controls the shared-signal tristate bus drivers. This guar-

antee is particularly important in the period before system clock oscillators are stable.

A response handshake bus provides for slave modules when protocols are shared. An address decoder protocol guarantees that there is only one driver at any given time.

The address decoder assumes the role of default slave device when no active slave has been addressed or selected, or more important, when the new address decoding is not yet resolved (when no valid slave module yet owns the response handshake bus).

ARM has filed a key patent application to cover the basic bus master-slave protocol.

Low-power designs

As we worked with leading low-power customers and OEMs (Psion on the design of the ARM7100, for example), we were motivated to define a layered approach to system bus design. The goal was to address the critical-path problems introduced by connecting every peripheral and memory subsystem to the main CPU bus. The solution was to minimize bus loading on critical bus components.

AMBA covers two distinct buses:

- *Advanced system bus.* The ASB is the main system bus. It has a 32-bit data bus that supports burst-mode signaling and multiple bus masters and allows high-bandwidth communication between the masters and the most significant slave devices, such as the external memory controller.
- *Advanced peripheral bus.* The APB is a minimalist peripheral I/O bus (typically 8, 16, or 32 data bits) that emphasizes a low-gate-count implementation for each peripheral. The APB is static except when an I/O access takes place. Thus, the high-bandwidth bus activity between the processor, any on-chip memory, and the external bus interface is decoupled from the peripheral bus.

On-chip test support

As Figure 3 shows, AMBA supports on-chip test through the test interface controller module, an integrated bus master. The TIC provides a 32-bit parallel access port suitable for manufacturing and in-circuit test by controlling the embedded system's external memory interface. As a result, designers can implement the TIC with minimal gate count to provide the control sequencing of the external bus data path. This bus master is given highest priority with the bus arbiter so that it can always gain access to the system bus.

ARM has filed a key patent application to cover test access to embedded cores and peripherals. The system infrastructure supports the direct testing of individual modules using the normal ASB transactions.[3]

Reconfiguring for test mode. In test mode, the TIC reconfigures the application-specific external bus interface to provide a high-speed, 32-bit, parallel vector interface. It uses a minimal three-wire handshake with the tester, optimized for intermodule bus transactions.

The only requirements for test reconfiguration are to provide a 32-bit bidirectional port for test vector access and provide a way to control the system clock. For a system with a 32-bit external data bus interface, the test port is trivial; however, for 16-bit and 8-bit data bus designs, the designer or tester must reconfigure 16 or 24 address lines as bidirectional test port signals for test mode access.

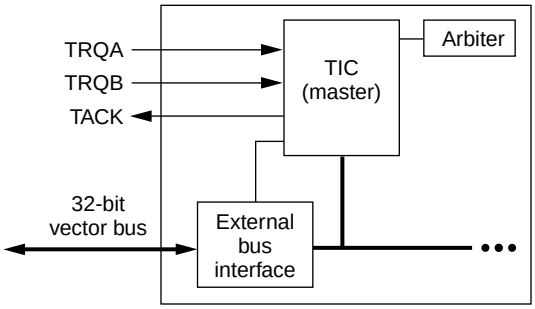


Figure 3. The test interface controller and external bus interface, which provide the 32-bit test access port.

As Figure 3 shows, the TIC has two dedicated inputs. TRQA is the test bus request input (request external bus). TRQB is the test control input (test vector mode). The TIC also has one output signal, TACK, test acknowledge (external test request is granted). These signals support the sequencing of the reused external interface for reading and writing 32-bit test vectors. They also provide an external bus enable/disable interface in normal operation, so it is easy to interface with external DMA controllers and do production IC testing.

To let the external tester clock the bus for TIC access, an external clock pin is required. In low-power designs with phase-locked loops and voltage-controlled oscillators, testers use the normal phase-locked loop bypass mode typically provided for analog subsystem testing.

When TRQA (active high) is asserted in normal system mode, it requests that the bus arbiter grant the bus access to the TIC's external bus interface. The arbiter conventionally gives the TIC the highest priority. By asserting the TACK signal, the TIC informs the external tester that the bus is guaranteed to be undriven.

Test mode cycle. In test mode, TRQA and TRQB encode

- *Address vector*—next cycle is burst transfer start address.
- *Write data vector*—next cycle is write data access.
- *Read data vector*—next cycle is read data access.
- *Exit test mode*—return the external bus to normal operating mode.

We built a bus-turnaround protocol into the test bus sequencing to ensure a contention-free tristate bus handover between the IC and tester.

In test mode, the TACK signal provides the wait-state handshake of the addressed module under test.

Test harness. To test on-chip bus masters, we created a test harness that decouples all inputs and outputs from system configuration. The harness also fully exercises core I/O as if the core were fully multiplexed with all pins visible to

the tester. Each master device simply becomes a normal bus slave during test, using the ASB handshake response.

Figure 4 depicts the test harness. We wrote a test language, TICTalk, in simple C macros and compiled them to produce the stimulus and expected test vectors for simulation or production test. The external bus interface (EBI) is typically application specific but is reused for test in a standardized manner.

The test access primitives to read and write data provide the mechanism for testing basic peripheral modules up to complex cached CPU macrocells. Designers can then apply vector test sets developed for cache memories and CPUs efficiently with high fault coverage.

Once the test harness is in place for a design, system design and diagnostic accesses become unified across application-specific microcontroller designs.

Several product designs have used this test harness successfully. Generally, designers developed the hardware using a simple uncached ARM7 core and substituted a custom cached core transparently at the last minute. The test harness requires no change; the designers run the more complex validation software with cache and MMU page table initialization only on the simulation of the final chip—when the rest of the peripherals and memory have already been validated. The ARM7100 design, for example, was developed using an ARM7TDMI bus master simulation model before the cached core was designed and ready.

Multiple development environments

As an aid to developing AMBA-based systems, we have developed both a microcontroller design kit and a prototyping platform for design evaluation and software and hardware development. The design kit, MicroPack, provides VHDL and Verilog sample designs. The prototyping platform, PID7T, permits the modeling of both on-chip and off-chip memory subsystems. Designers can conduct what-if experimentation and comparisons when trading off on-chip area and external commodity memory. The PID7T also integrates seamlessly with the ARM software development toolkits.

Figure 5 shows the PID7T. The card implements an entire, discrete AMBA microcontroller, memory, and peripheral environment in programmable logic devices (PLDs), to provide a full 32-bit ASB (top) and APB (bottom). The decoder, arbiter, and reset controller are implemented discretely, as is support for the TIC.

Specific features of the PID7T include:

- *Zero-wait-state RAM.* Designers can emulate on-chip memory (modeled using fast synchronous SRAM cache devices) or on-chip RAM from 4 to 128 Kbytes.
- *External static memory emulation bank.* The bank is 512 Kbytes, user-programmable, and split into two banks of emulated SRAM or ROM, each with DIP switch configuration of 32-, 16-, and 8-bit-wide memory with one to four wait states.
- *Two DRAM SIMM slots.* These support 16 or 32 Mbytes of bulk DRAM for large program development.
- *FPGA-modeled subsystem of reference peripherals.* This includes interrupt controller and counter-timer functionality for operating system development.
- *CPU daughter card.* The card in (top left, Figure 5) acts as a test chip with the ARM macrocell plus basic I/O pad ring and a bus master state machine PLD. The card interfaces with 5-V level shifters to permit implementation of the system motherboard with higher speed standard voltage PLDs and FPGAs. Designers can thus upgrade cached CPUs and in-circuit emulation.
- *Support for interfaces to the development toolkits.* Support is for standard serial, parallel, and Ethernet (via the PCMCIA card) host interfaces.

Building full on-chip bus systems with discrete devices and standard programmable logic naturally limits the bus clock frequency to the tens of MHz range. For many telecommunications applications, however, this is quite sufficient for at-speed real-time prototyping and software validation. Some developers still require real-time hardware prototypes so that they can develop the software and ASIC concurrently.

The development cards provide a platform that third-party tool developers can use as an interface to ARM processors. A commercial in-circuit emulator developed by Lauterbach GmBH (http://www.lauterbach.com) connects to the processor-daughter card interface. External connectors allow access suitable for logic analysis connection or add-on prototype hardware subsystems.

With our modular approach to the CPU header card design, designers can directly plug in upgrades of the CPU test chips to support cached standard parts and new test chips.

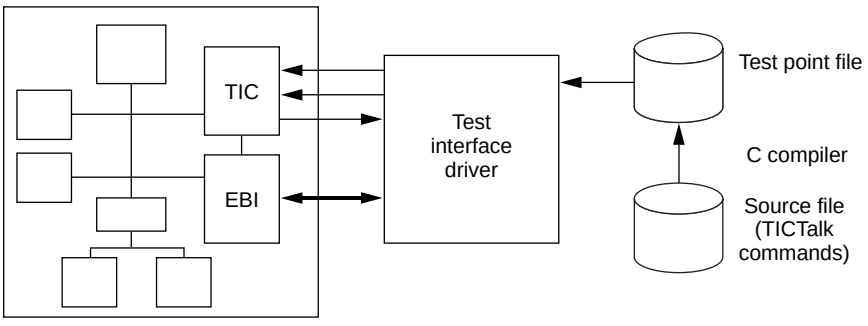


Figure 4. AMBA test harness. The harness supports the testing of an on-chip module using canned vectors. Because test patterns for individual blocks are modular, having a common test harness makes it easier to reuse the blocks.

AS ORGANIZATIONS continue to emphasize design reuse, on-chip buses such as AMBA will become increasingly important. We have opened access to the AMBA specifi-

cation to encourage design-ins and the development of peripheral library components.[4] We are also licensing MicroPack to existing ARM licensees to support the development of AMBA-based microcontrollers and let them use the test access and burst-mode protocol patents royalty-free when developing ARM-based designs.

ARM continues to work with leading semiconductor licensees and integrators. We have already identified several possible areas of extension to accommodate products that integrate on-chip DRAM and have higher bandwidth external memory addresses.

- *Bidirectional and unidirectional bus implementations.* We plan to carefully define bus protocols to work with bidirectional tristate buses. The bus communication layer would then let designers substitute separate unidirectional read and write buses, leaving the protocol unchanged. Multiplexers would route the selected master and slave transaction ports.
- *Split bus transactions.* The retract mechanism in the current AMBA specification (see the AMBA Terminology and Specification box) does not allow generalized multiple outstanding transactions between bus masters unless the slave device tracks the bus master grant signals to prevent deadlock. Adding split transaction support would let the bus master separate or split the request for a read or write transaction from the subsequent acknowledgment of the addressed slave. A declaration of the full transaction length at the start of the transfer would ensure that adequate buffer or first-in, first-out space is available to handle the full burst transaction.
- *Higher bus clock rates.* Most handheld telecommunications products for pagers or mobile phones currently run with bus clock frequencies between 4 and 27 MHz. However, new multimedia product designs demand higher performance, with the system bus speed typically tuned to optimally use off-chip memory bandwidth. Decoupling the peripheral bus (or multiple APB stubs) from the system bus offers designers a path to higher bandwidth system buses without redesigning peripheral subsystems. Cached processor cores may then run at a higher multiple of the bus clock.
- *Wide on-chip DRAM.* Several ARM semiconductor licensees use embedded DRAM technology. Merging modular test techniques will allow them to exploit both the CPU and memory in a range of smart-phone and organizer products.[5] DRAM is inherently built as a row/column accessed array, in which the width of a row has always been multiplexed down heavily for cost-effective standard-part memory device packages. Once the DRAM, processor, DMA, and DSP functions are integrated in the same device, electromagnetic emissions from an interchip memory bus decrease. Widening the on-chip system bus then becomes a very attractive option, allowing designers to tune the system for 64- or 128-bit widths. This provides ideal access for cache line fills and video DMA for LCD display, for example. Again, the complexity of wide bus widths is isolated from peripherals in the hierarchical bus partitioning adopted in AMBA.

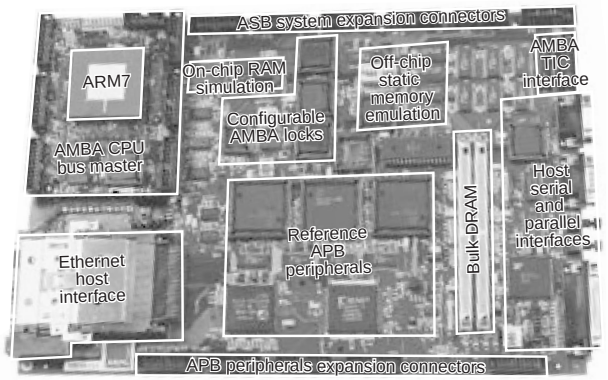
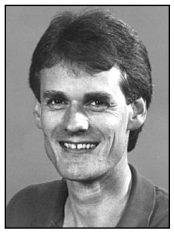


Figure 5. PID7T, the AMBA-based development card for the ARM7 processor family.

References

1. J. Hennessey and D. Patterson, *Computer Architecture—A Quantitative Approach*, 2nd ed., Morgan Kaufmann, San Mateo, Calif., 1996.
2. OMI-324 specification: http://www.omimo.be/public/data/_indstan.htm.
3. D. W. Flynn, "Modular Bus Supports On-Chip Testability," *IEE Colloquium: Systems Design for Testability*, No. 1995/083, paper 2.
4. I. Phillips and T. Dent, "The Development and Debug of Microsystem Controllers," http://www.isdmag.com/ic_uP/Development.html.
5. D. Bursky, "Combo RISC CPU and DRAM Solves Data Bandwidth Issues," *Electronic Design*, Mar. 4, 1996.



David Flynn is the engineering manager for the Systems Technology Group at Advanced RISC Machines, Ltd. He holds patents in low-power, on-chip bus and video/audio system applications. Previously, he was responsible for both systems and software design in the Advanced Research and Development Group at Acorn Computers, Cambridge, which developed the original ARM processor chip set.

Flynn received a BSc in computer science from Hatfield Polytechnic.

Direct questions about this article to Flynn at Advanced RISC Machines, Ltd., 90 Fulbourn Rd., Cherry Hinton, Cambridge, CB1 4JN UK; david.flynn@arm.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156 Medium 157 High 158