ARM920T

System-on-Chip Platform OS Processor

Product Overview

Applications

- Applications running a platform OS:
  - EPOC
  - Linux
  - WindowsCE
- High performance wireless applications:
  - Smart phones
  - PDAs
- Networking applications
- Digital set top boxes
- Imaging
- Automotive control solutions
- Audio and video encoding and decoding

Benefits

- Designed specifically for System-On-Chip integration
- Supports the Thumb instruction set offering the same excellent code density as the ARM7TDMI.
- High performance allows system designers to integrate more functionality into price and power-sensitive applications demanding more performance
- Cached processor with an easy to use lower frequency on-chip system bus interface.

The ARM920T™

The ARM920T is a a high-performance 32-bit RISC integer processor macrocell combining an ARM9TDMI™ processor core with:

- 16KB instruction and 16KB data caches
- instruction and data *Memory Management Units* (MMUs)
- write buffer
- an AMBA™ (Advanced Microprocessor Bus Architecture) bus interface
- an *Embedded Trace Macrocell* (ETM) interface.

High performance

The ARM920T provides a high-performance processor solution for *open* systems requiring full virtual memory management and sophisticated memory protection. An enhanced ARM® architecture version 4 MMU implementation provides translation and access permission checks for instruction and data addresses.

The ARM920T high-performance processor solution gives considerable savings in chip complexity and area, chip system design, and power consumption.

Compatible with ARM7™ and StrongARM®

The ARM920T processor is 100% user code binary compatible with ARM7TDMI and backwards compatible with the ARM7 Thumb® Family and the StrongARM processor families, giving designers software-compatible processors with a range of price/performance points from 60 MiPS to 200+ MIPS. Support for the ARM architecture today includes:

- WindowsCE, EPOC, Linux, and QNX operating systems
- 40+ Real Time Operating Systems
- cosimulation tools from leading EDA vendors
- variety of software development tools.

.

ARM920T Macrocell

The ARM920T macrocell is based on the ARM9TDMI Harvard architecture processor core, with an efficient five stage pipeline. The architecture of the processor core or integer unit is described in more detail page 9.

To reduce the effect of main memory bandwidth and latency on performance, the ARM920T includes:

- instruction cache
- data cache
- MMU
- TLBs
- write buffer
- physical address TAG RAM

Caches

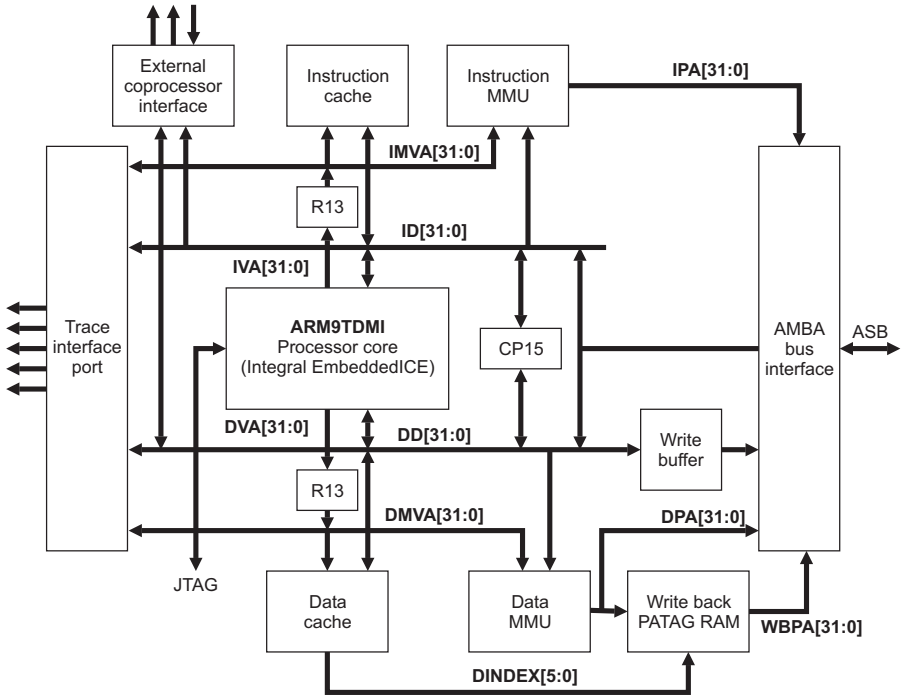
Two 16KB caches are implemented, one for instructions, the other for data, both with an 8-word line size. A 32-bit data bus connects each cache to the ARM9TDMI core allowing a 32-bit instruction to be fetched and fed into the instruction Decode stage of the pipeline at the same time as a 32-bit data access for the Memory stage of the pipeline.

Cache lock-down

Cache lock-down is provided to allow critical code sequences to be locked into the cache to ensure predictability for real-time code. The cache replacement algorithm can be selected by the operating system as either pseudo random or round-robin. Both caches are 64-way set-associative. Lock-down operates on a per-set basis

Write buffer

The ARM920T also incorporates a 16-entry write buffer, to avoid stalling the processor when writes to external memory are performed.



ARM920T function block diagram

PATAG RAM

The ARM920T implements PATAG RAM to perform write-backs from the data cache.

The physical address of all the lines held in the data cache is stored by the PATAG memory, removing the need for address translation when evicting a line from the cache.

MMUs

The standard ARM920T implements an enhanced ARMv4 MMU to provide translation and access permission checks for the instruction and data address ports of the ARM9TDMI.

The MMU features are:

- standard ARMv4 MMU mapping sizes, domains, and access protection scheme
- mapping sizes are 1MB sections, 64KB large pages, 4KB small pages and new 1KB tiny pages
- access permissions for sections
- access permissions for large pages and small pages can be specified separately for each quarter of the page (these quarters are called subpages)
- 16 domains implemented in hardware
- 64-entry instruction TLB and 64-entry data TLB
- hardware page table walks
- round-robin replacement algorithm (also called cyclic).

System controller

The system controller oversees the interaction between the instruction and data Cache and the Bus Interface Unit. It controls internal arbitration between the blocks and stalls appropriate blocks when required. The system controller arbitrates between instruction and data access to schedule single or simultaneous requests to the MMUs and the Bus Interface Unit. The system controller receives acknowledgement from each resource to allow execution to continue.

Control coprocessor (CP15)

The CP15 allows configuration of the caches, the write buffer, and other ARM920T options.
Several registers within CP15 are available for program control, providing access to features such as:

- invalidate whole TLB using CP15
- invalidate TLB entry, selected by modified virtual address, using CP15
- independent lock-down of instruction TLB and data TLB using CP15 register 10
- big or little-endian operation
- low power state
- memory partitioning and protection
- page table address
- cache and TLB maintenance operations.

Registers

The ARM9TDMI processor core consists of a 32-bit datapath and associated control logic. That datapath contains 31 general-purpose registers, coupled to a full shifter, Arithmetic Logic Unit, and multiplier. At any one time 16 registers are visible to the user. The remainder are synonyms used to speed up exception processing. Register 15 is the *Program Counter* (PC) and can be used in all instructions to reference data relative to the current instruction. R14 holds the return address after a subroutine call. R13 is used (by software convention) as a stack pointer.

Modes and exception handling

All exceptions have banked registers for R14 and R13. After an exception, R14 holds the return address for exception processing. This address is used both to return after the exception is processed and to address the instruction that caused the exception. R13 is banked across exception modes to provide each exception handler with a private stack pointer. The fast interrupt mode also banks registers 8 to 12 so that interrupt processing can begin without the need to save or restore these registers. A seventh processing mode, System mode, does not have any banked registers. It uses the User mode registers. System mode runs tasks that require a privileged processor mode and allows them to invoke all classes of exceptions.

Status registers

All other processor states are held in status registers. The current operating processor status is in the *Current Program Status Register* (CPSR). The CPSR holds:

- four ALU flags (Negative, Zero, Carry, and Overflow),
- two interrupt disable bits (one for each type of interrupt),
- a bit to indicate ARM or Thumb execution,
- and five bits to encode the current processor mode.

All five exception modes also have a *Saved Program Status Register* (SPSR) which holds the CPSR of the task immediately before the exception occurred.

Exception types

ARM9TDMI supports five types of exception, and a privileged processing mode for each type. The types of exceptions are:

- fast interrupt (FIQ)
- normal interrupt (IRQ)
- memory aborts (used to implement memory protection or virtual memory)
- attempted execution of an undefined instruction
- software interrupts (SWIs).

Conditional execution

All ARM instructions (with the exception of BLX) are conditionally executed. Instructions optionally update the four condition code flags (Negative, Zero, Carry, and Overflow) according to their result. Subsequent instructions are conditionally executed according to the status of flags. Fifteen conditions are implemented.

Four classes of instructions

The ARM and Thumb instruction sets can be divided into four broad classes of instruction:

- data processing instructions
- load and store instructions
- branch instructions
- coprocessor instructions.

Data processing

The data processing instructions operate on data held in general purpose registers. Of the two source operands, one is always a register. The other has two basic forms:

- an immediate value
- a register value optionally shifted.

If the operand is a shifted register the shift amount might have an immediate value or the value of another register. Four types of shift can be specified. Most data processing instructions can perform a shift followed by a logical or arithmetic operation. Multiply instructions come in two classes:

- normal - 32-bit result
- long - 32-bit result variants.

Both types of multiply instruction can optionally perform an accumulate operation.

Load and store

The second class of instruction is load and store instructions. These instructions come in two main types:

- load or store the value of a single register
- load and store multiple register values.

Load and store single register instructions can transfer a 32-bit word, a 16-bit halfword and an 8-bit byte between memory and a register. Byte and halfword loads might be automatically zero extended or sign extended as they are loaded. Swap instructions perform an atomic load and store as a synchronization primitive.

Addressing modes

Load and store instructions have three primary addressing modes:

- offset
- pre-indexed
- post-indexed.

They are formed by adding or subtracting an immediate or register-based offset to or from a base register. Register-based offsets can also be scaled with shift operations. Pre-indexed and post-indexed addressing modes update the base register with the base plus offset calculation. As the PC is a general-purpose register, a 32-bit value can be loaded directly into the PC to perform a jump to any address in the 4GB memory space.

Block transfers

Load and store multiple instructions perform a block transfer of any number of the general purpose registers to or from memory. Four addressing modes are provided:

- pre-increment addressing
- post-increment addressing
- pre-decrement addressing
- post-decrement addressing.

The base address is specified by a register value (that can be optionally updated after the transfer). As the subroutine return address and the PC values are in general-purpose registers, very efficient subroutine calls can be constructed.

Branch

As well as allowing any data processing or load instruction to change control flow (by writing the PC) a standard branch instruction is provided with 24-bit signed offset, allowing forward and backward branches of up to 32MB.

Branch with Link

There is a Branch with Link (BL) that allows efficient subroutine calls. BL preserves the address of the instruction after the branch in R14 (the Link Register or LR). This allows a move instruction to put the LR in to the PC and return to the instruction after the branch.

The third type of branch (BX and BLX) switches between ARM and Thumb instruction sets optionally with the return address preserving link option.

Coprocessor

There are three types of coprocessor instructions:

- coprocessor data processing instructions invoke a coprocessor specific internal operation
- coprocessor register transfer instructions allow a coprocessor value to be transferred to or from an ARM register
- coprocessor data transfer instructions transfer coprocessor data to or from memory, where the ARM calculates the address of the transfer.

The ARMv4T ARM instruction set

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	TEQ	Test Equivalence
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
MUL	Multiply	MLA	Multiply Accumulate
SMULL	Sign Long Multiply	SMLAL	Signed Long Multiply Accumulate
UMULL	Unsigned Long Multiply	UMLAL	Unsigned Long Multiply Accumulate
CLZ	Count Leading Zeroes	BKPT	Breakpoint
MRS	Move From Status Register	MSR	Move to Status Register
B	Branch		
BL	Branch and Link	BLX	Branch and Link and Exchange
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
SWP	Swap Word	SWPB	Swap Byte
CDP	Coprocessor Data Processing		
MRC	Move From Coprocessor	MCR	Move to Coprocessor
LDC	Load To Coprocessor	STC	Store From Coprocessor

The Thumb instruction set

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	NEG	Negate
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
LSL	Logical Shift Left	LSR	Logical Shift Right
ASR	Arithmetic Shift Right	ROR	Rotate Right
MUL	Multiply	BKPT	Breakpoint
B	Unconditional Branch	Bcc	Conditional Branch
BL	Branch and Link	BLX	Branch and Link and Exchange
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
PUSH	Push Registers to stack	POP	Pop Registers from stack

Modes and registers

User and System mode	Supervisor mode	Abort mode	Undefined mode	Interrupt mode	Fast Interrupt mode
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

Mode-specific banked registers

ARM9TDMI processor core

The ARM9TDMI processor core implements the ARMv4T *Instruction Set Architecture* (ISA). The ARMv4T ISA is a superset of the ARMv4 ISA (implemented by the StrongARM® processor) with additional support for Thumb instruction set (16-bit compressed ARM instruction set). The ARMv4T ISA is implemented by the ARM7™ Thumb family; giving full upward compatibility to the ARM9™ Thumb family.

Performance and code density

ARM9TDMI executes two instruction sets:

- 32-bit ARM instruction set
- 16-bit Thumb instruction set.

The ARM instruction set allows a program to achieve maximum performance with the minimum number of instructions.

The majority of ARM9TDMI instructions are executed in a single cycle. These are shown in the ARM9TDMI instruction execution timing table on page 10.

The simpler Thumb instruction set offers much increased code density increasing space optimization. Code can switch between the ARM and Thumb instruction sets on any procedure call.

ARM9TDMI integer pipeline stages

The integer pipeline consists of five stages to maximize instruction throughput on ARM9TDMI:

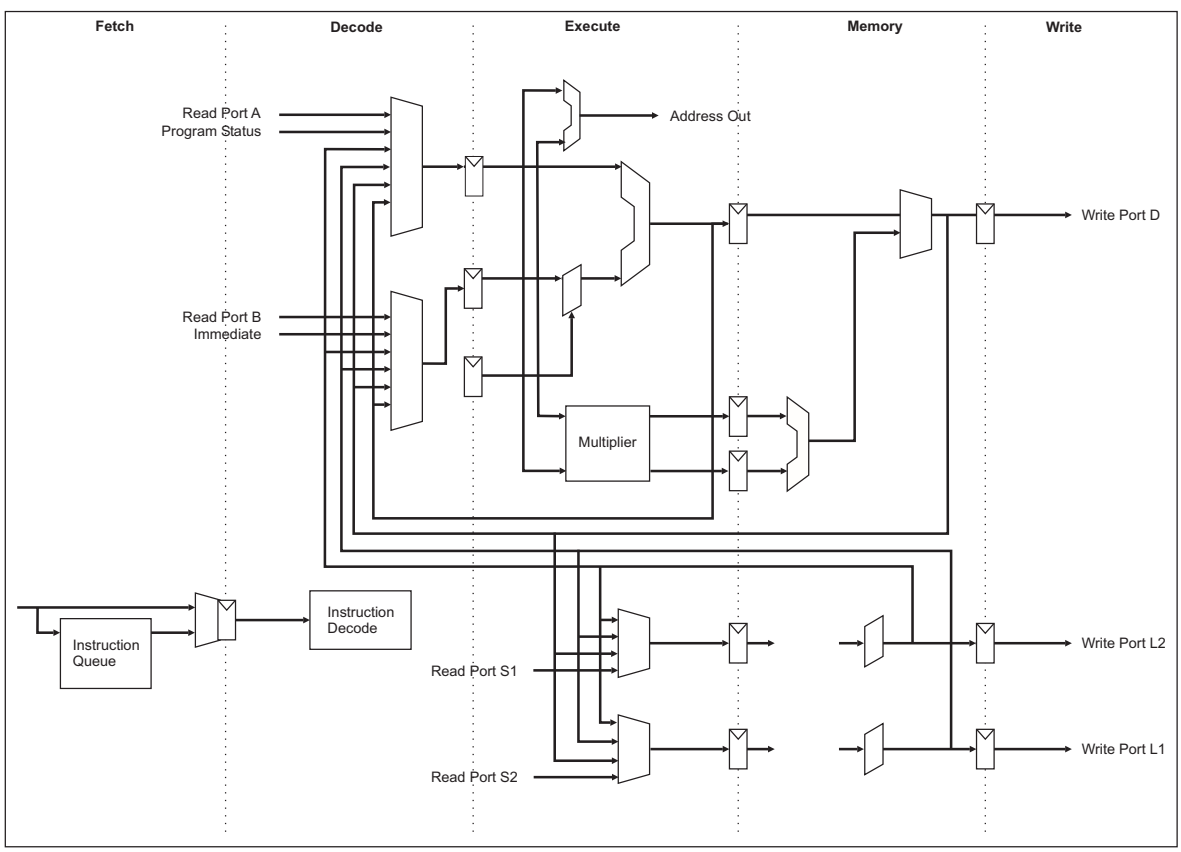
F: Fetch

D: Decode and register read

E: Execute shift and alu, or address calculate, or multiply

M: Memory access and multiply

W: Write register



ARM9TDMI integer pipeline

Pipelining

By overlapping the various stages of execution, ARM9TDMI maximizes the clock rate achievable to execute each instruction. It delivers a throughput approaching one instruction per cycle.

32-bit data buses

ARM9TDMI provides 32-bit data buses:

- between the processor core and the instruction and data caches
- between coprocessors and the processor core.

This allows ARM9TDMI to achieve very high-performance on many code sequences, especially those that require data movement in parallel with data processing.

Coprocessors and pipelines

The ARM9TDMI coprocessor interface allows full independent processing in the ARM execution pipeline and supports up to16 independent coprocessors.

Debug features

The ARM9TDMI processor core also incorporates a sophisticated debug unit to allow both software tasks and external debug hardware to perform hardware and software breakpoint, single stepping, register, and memory access. This functionality is made available to software as a coprocessor and is accessible from hardware via the JTAG port. Full-speed, real-time execution of the processor is maintained until a breakpoint is hit. At this point control is passed either to a software handler or to JTAG control.

Table 1: ARM9TDMI instruction execution timing

Instruction class	Issue cycles	Result delay
Condition failed	1	NA
ALU instruction	1	0
ALU instruction with register shift	2	0
MOV PC, Rx	3	NA
ALU instruction dest = PC	4	0
MUL	1..7	1..7
MSR (flags only)	1	0
MSR (mode change)	3	NA
MRS	1	0
LDR	1	1
LDR with shifted offset	1	2
STR	1	NA
LDM	1	Position in list + 1
STM	1	NA
SWP	2	1
CDP	1	NA
MRC	1	1
MCR	1	NA
LDC	1	Number of words
STC	1	NA

System Issues and Third Party Support

JTAG debug

The internal state of the ARM920T is examined through a JTAG-style serial interface, which allows instructions to be serially inserted into the pipeline of the core without using the external data bus. Therefore, when in debug state, a store-multiple (STM) can be inserted into the instruction pipeline. This exports the contents of the ARM9TDMI registers. This data can be serially shifted out without affecting the rest of the system.

AMBA bus architecture

The ARM9 Thumb Family processors are designed for use with the AMBA multi-master on-chip bus architecture. AMBA includes an *Advanced System Bus* (ASB) connecting processors and high-bandwidth peripherals and memory interfaces, and a low-power peripheral bus allowing a large number of low-bandwidth peripherals.

The ARM920T ASB implementation provides a 32-bit address bus and a 32-bit data bus for high-bandwidth data transfers made possible by on-chip memory and modern SDRAM and RAMBUS memories.

Everything you need

ARM provides a wide range of products and services to support its processor families, including software development tools, development boards, models, applications software, training, and consulting services.

The ARM Architecture today enjoys broad third party support. The ARM9 Thumb Family processors strong software compatibility with existing ARM families ensures that its users benefit immediately from existing support. ARM is working with its software, EDA, and semiconductor partners to extend this support to use new ARM9 Family features.

Current support

Support for the ARM Architecture today includes:

- *ARM Developer Suite* (ADS)
  - Integrated development environment
  - C, C++, assembler, simulators and windowing source-level debugger
  - Available on Windows95, WindowsNT, and Unix.
  - ARM Multi-ICE™ JTAG interface
  - Allows EmbeddedICE software debug of ARM processor systems
  - integrates with the ADS.
- ARMulator instruction-accurate software simulator
- Development boards
- Design Simulation Models provide signoff quality ASIC-simulation
- Software toolkits available from ARM, Cygnus/GNU,Greenhills, JavaSoft, MetaWare, Microtec, and Windriver allowing software development in C, C++, Java, FORTRAN, Pascal, Ada, and assembler.
- 20+ Real Time Operating Systems including:
  - Windriver VxWorks
  - Sun Microsystems Chorus
  - JavaOS
  - Microtec VRTX
  - JMI
  - Embedded System Products RTXC
  - Integrated Systems pSOS.
- Major OS including:
  - Microsoft WindowsCE
  - PSION EPOC
  - NetBSD
  - Linux UNIX
  - Geoworks
- Application software components:
  - speech and image compression
  - software modem
  - Chinese character input network protocols
  - Digital AC3 decode
  - MPEG3 encode and decode
  - MPEG4 decode and encode
- Hardware/software cosimulation tools from leading EDA Vendors.

For more information, see `www.arm.com`

Contacting ARM

America

ARM INC.
750 University Avenue
Suite 150,
Los Gatos CA 95032
USA

Tel: +1-408-579-2209
Fax:+1-408-399-8854
Email: info@arm.com

Austin Design Center
ARM
1250 Capital of Texas Highway
Building3, Suite 560
Austin
Texas 78746
USA

Tel: +1-512-327-9249
Fax:+1-512-314-1078
Email: info@arm.com

Seattle
ARM
10900 N.E. 8th Street
Suite 920
Bellevue
Washington 98004
USA

Tel: +1-425-688-3061
Fax:+1-42-454-4383
Email: info@arm.com

‚Boston
ARM
300 West Main St., Suite 215
Northborough
MA 01532
USA

Tel: +1-508-351-1670
Fax:+1-508-351-1668
Email: info@arm.com

England
ARM Ltd.
110 Fulbourn Road
Cherry Hinton
Cambridgeshire
CB1 9NJ
England

Tel: +44 1223 400400
Fax:+44 1223 400410
Email: info@arm.com

France
ARM France
12, Avenue des Prés
BL 204 Montigny le Bretonneux
78059 Saint Quentin en Yvelines
Cedex
Paris
France

Tel: +33 1 30 79 05 10
Fax: +33 1 30 79 05 11
Email: info@arm.com

Germany
ARM
Rennweg 33
85435 Erding
Germany

Tel: +49 8122 89209-0
Fax:+49 8122 89209-49
Email: info@arm.com

Japan
ARM K.K.
Plustaria Building 4F
3-1-4 Shin-Yokohama
Kohoku-ku,
Yokohama-shi
Kanagawa 222-0033

Tel: +81 45 477 5260
Fax: +81 45 477 5261
Email: info-armkk@arm.com

Korea
Room 1809,
Hofficetel
Yangjae-dong
Seocho-ku
Seoul
Korea

Tel: 00 82 2 3462 8271
Fax: 00 82 2 3462 8274
Email: info@arm.com