WHITE PAPER ARM

Cortex-R4

A mid-range processor for deeply-embedded applications

John Penton and Shareef Jalloq, ARM **May 2006**

The ARM® Cortex™-R4 processor enables a wide range of deeply-embedded applications to be implemented with reduced system and development cost. Product applicability includes hard-disk drives, networking products, wireless and automotive applications.

Several microarchitecture and feature enhancements, including the use of the Thumb-2 instruction-set architecture, significantly reduce the Cortex-R4 processor's cycle per instruction (CPI) compared with similar processors in its class. Efficient implementation ensures class-leading power, performance and area attributes for the Cortex-R4 processor. A number of other features, including a single AMBA® 3 AXI™ interface and other memory system enhancements, make the Cortex-R4 processor easier to integrate.

Implementing the ARMv7R architecture, the Cortex-R4 processor is software-compatible with the ARM9E family of processors. Inclusion of the complete Cortex feature set ensures compatibility with future Cortex family processors.

This paper explores the market and technology drivers for deeply-embedded products and expands on the technical features that make the Cortex-R4 processor highly suited to a broad range of cost-sensitive, deeply embedded applications.

Market Drivers

Deeply embedded applications are commonly found across a broad range of markets. Key markets include automotive applications, specifically car chassis and braking systems. Products such as hard disk drives, wireless modems, printers and home gateways are all essentially consumer products and recent, strong consumer demand has resulted in increased processor shipments for many of these markets.

Market projections suggest that the entire deeply-embedded opportunity will reach in excess of 1.25 billion units per year by 2008.

Understanding the Needs of Deeply Embedded Applications

The common requirement across many deeply embedded products is to achieve optimum performance at the lowest possible cost. Selecting a design platform of much higher performance than the application requires can result in unnecessary expense and power dissipation, leading to an uncompetitive product.

For embedded CPU systems this presents an additional challenge - the need to compete on providing the lowest possible program cost to the end user. This has implications not only for individual unit cost, but significantly impacts upon the broader design picture.

Achieving faster time-to-market through advanced debug support, reducing overall system costs via improved code density to reduce memory size, as well as harnessing design expertise through industry standards such as AMBA are all factors which will enable system designers to satisfy the demands of these intensely competitive markets.

Many deeply embedded applications are growing in complexity and so require performance and functionality increases from the processor cores that enable them. Examples include:

Imaging
The personal imaging and printing markets have evolved from traditional PC-centric peripherals toward a new class of intelligent, PC-independent devices.

Today's imaging devices - from low-end inkjets and digital cameras to high-end laser printers and multifunction peripherals - are performing more compute intensive operations. Tasks such as image processing, colour correction, JPEG interpretation, and PDL/PCL handling can now be performed autonomously, allowing image-rich content such as web pages, digital photos, and scanned documents to be captured, stored, navigated, viewed, transmitted, and printed without ever touching a PC.

Manufacturers of imaging products operate in highly competitive consumer markets and developers need to be ahead of the industry with advanced products that anticipate consumer demand.

Automotive
In the automotive market, advanced electronic devices are driving innovation in areas such as in-car entertainment systems, in-car safety devices, navigation systems and diagnostic equipment.

For safety-critical applications, such as automotive anti-lock braking systems (ABS), reliability is paramount and qualification cycles are long. Car chassis and safety systems are becoming increasingly complex. For example, electronic stability control (ESP) is an order of magnitude more complex than ABS. For designers, the ability to provide testing logic within the processor or demonstrate a high level of fault coverage in a timely manner is essential.

Storage
Advances in disk drive technology and the spiraling demand for increased storage capacity in both enterprise and consumer electronics is driving the use of disk drives for data intensive and I/O intensive applications. This includes small form factor storage devices for portable consumer products such as MP3 players and digital cameras.

Such storage devices are being deployed in applications with wide-ranging constraints. A PC disk drive, for instance, will typically be optimized for random data access, whereas a portable MP3 player requires the same hardware but will need to be configured for continuous data access, low power, low noise and robust head-crash protection.

Customer-Driven Enhancements
ARM has considerable experience in serving the deeply-embedded market. Continuously monitoring customer and market feedback has led to a number of requirements being identified, and subsequently addressed in the Cortex-R4 processor design. These include:

- Cost:
    - Less on-chip memory
    - Lower-cost, smaller external memories with relaxed timing

- Performance and power:
    - Maintain clock frequency even when increased application performance is a requirement, so that power consumption is not increased
    - Faster interrupt handling

    - Smaller external memories need less power

- Reliability:
    - Improved soft-error management

- Design Flexibility:
    - Local memory architecture (Tightly Coupled Memory – TCM) to be more flexible, with simple coherent DMA support for both instruction and data
    - The need to partition the I and D TCM at design time can be limiting

- Automotive: Application-specific requirement to support the automotive OSEK 2.1.1 specification

Cortex-R4 Processor: Features at a Glance

The ARM Cortex-R4 processor is a mid-range synthesizable CPU for deeply embedded applications, including automotive, baseband controller, imaging, mass storage/HDD and microcontrollers. Until now, the features that satisfy this breadth of deeply-embedded applications have required a range of processors to deliver the balance of performance with low cost and power.

The ARM9E family of processors, including the ARM966E-S™, ARM946E-S™ and ARM926EJ-S™ processors, has been designed-in to hundreds of deeply-embedded applications with manufacturing volumes of many millions.

The breadth of features within the Cortex-R4 processor enables design teams to adopt a single processor to efficiently target a broad range of deeply-embedded tasks, bringing further economies of scale in development and manufacturing costs.

The Cortex-R4 processor builds on the ARM9E foundation. The Cortex-R4 processor offers more performance than the ARM946E-S processor, and is 50 percent more efficient than the ARM946E-S processor running Thumb code. Depending on the configuration, at 200MHz the Cortex-R4 processor can be smaller and consume less power than the ARM946E-S processor. At reduced clock frequencies, the Cortex-R4 processor gate count can be as low as 180K gates. Targeting 0.13µ G processes, performance of 300MHz is achievable.

Based on the ARMv7 instruction set architecture, the Cortex-R4 processor utilizes Thumb-2 technology for enhanced performance and improved code density. Thumb-2 is a blended instruction set. It contains all the 16-bit instruction opcodes from the Thumb instruction set, as well as a large range of 32-bit instructions to provide almost the full functionality of the original ARM instruction set. This means that 16- and 32-bit instructions can be mixed on an instruction by instruction basis and, crucially for development costs, the optimum instruction size mix can be effectively selected by a compiler. The result is that Thumb-2 code can retain the high performance of ARM code, while giving the code density benefit of Thumb. The ARMv7 architecture improves exception and interrupt handling and provides improved support for non-maskable interrupts (NMI).

The Cortex-R4 processor makes use of several new ARM technologies, including the AMBA 3 AXI protocol for more efficient on-chip interconnect. Many of the enhanced features in the Cortex-R4 processor directly address feedback from ARM9E family customers:

- More flexible TCM interface, with DMA support. The time for RAM accesses is more than doubled compared with ARM9E family processors.
- Support for error detection on RAMs for improved reliability
- The enhanced MPU with 8 or 12 regions provides a finer granularity for stack checking. Regions may also overlap for better memory layout.

- Flexible configuration options for caches, TCM, MPU and debug which can save up to 40K gates depending on the chosen configuration.
- Prefetch unit and branch prediction to deliver more performance at the same clock frequency.
- A synthesis-time option to generate a redundant copy of the processor logic enables error detection in safety-critical systems, such as ABS. This is a specialised feature that enables two processors plus relevant checking logic to be implemented.

Cortex-R4 Processor Configurations

Key to maintaining efficiency with the Cortex-R4 processor's broad application coverage is the capability to configure the design at the synthesis stage. Table 1 summarizes three possible configuration options for the processor, with the main parameters listed.

Configuration Feature	**HDD**	**Automotive**	**Imaging/Wireless**
MPU	No	Yes (12 regions)	Yes (8 regions)
Caches	Sometimes	Sometimes	Yes
Breakpoints/ watchpoints	Minimum	Maximum	Maximum
Parity	Sometimes	Yes	No

Table 1: Configuration options for three target market applications

Cortex-R4 Processor Microarchitecture

Pipeline Structure

The Cortex-R4 processor targets lower frequency clock rates than, for example, the ARM11 family of processors, and a simplified pipeline structure accommodates the required performance while ensuring that the Cortex-R4 processor is some 30 percent smaller than the ARM1156T2-S processor.

The area efficiency is due to a number of optimizations that have been made to the microarchitecture. Compared with the ARM1156T2-S, one less pipeline stage in the data processing unit (DPU) and a smaller multiplier, as well as a much smaller (though slightly more efficient) prefetch unit, all contribute to the reduced gate count. In addition, the Load-Store Unit (LSU) is simplified and the Hit Under Miss (HUM) feature removed. Both the cache controllers and AMBA 3 AXI master are also smaller.

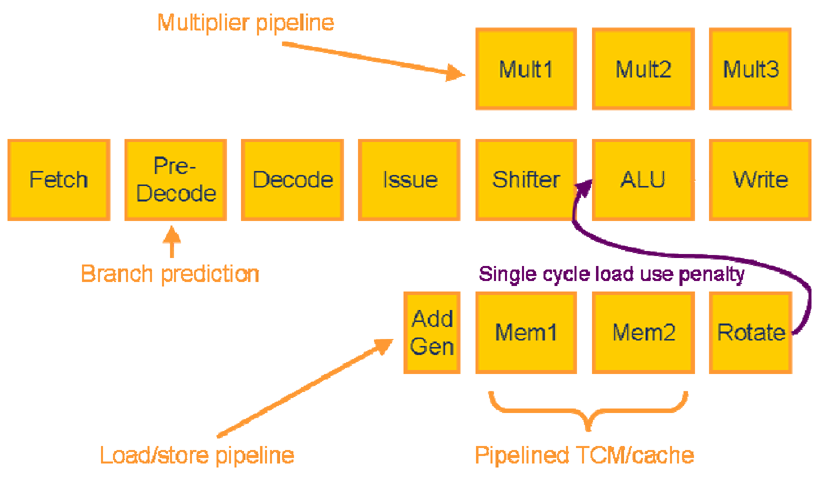

Figure 1: Cortex-R4 processor pipeline

The single cycle load use penalty for the Cortex-R4 processor makes the job of the compiler to schedule code into these “slots” much easier. Two cycles for RAM accesses are better able to accommodate the needs of compiled RAMs.

Prefetch Unit and Branch Prediction

The Cortex-R4 processor’s pipeline incorporates an instruction prefetch unit (PFU), a major part of which is the program flow predictor. Incorporating this functionality into the pipeline helps to lower the required cycles per instruction (CPI). Accurately predicting program flow minimizes the number of pipeline flushes due to mispredicted branches.

The main functions of the PFU are prefetching, program flow prediction and pre-decoding.

Prefetching

The prefetch unit comprises the Fe1, Fe2 and Pd stages of the pipeline. The PFU is responsible for fetching 64-bit wide packets of data from the I-side memory, extracting instructions from them (including ARM and Thumb-2) and presenting an instruction stream to the DPU.

The PFU was designed to be used with high latency level 1 cache memory and TCM, where the typical access time is two cycles rather than one. As such, the PFU incorporates a three-entry 64-bit prefetch buffer and as long as there is space in the prefetch buffer, the PFU is able to fetch the next data packet.

Program Flow Prediction

Two classes of prediction are used: ‘Taken-ness’ branch prediction and target address prediction.

‘Taken-ness’ branch prediction predicts branches where the target address is a fixed offset from the program counter. These branches are handled by the dynamic branch predictor using a global history scheme. A global history prediction scheme is an adaptive predictor that learns the behaviour of branches during program execution. The prediction is checked later in the pipeline, once the condition code of the branch has been evaluated. Conditional return instructions will have their ‘taken-ness’ predicted by the branch predictor. Predicting program-flow changes such as branches allows the PFU to start fetching instructions from the branch destination before the DPU has executed the branch, thus saving a number of cycles, provided that the prediction was correct.

For branches where the target address is not a fixed offset from the program counter, the target can only be easily predicted if it is a function return. Function return prediction enables instructions to have their target address predicted by the return stack.

The return stack is used to accelerate returns from functions. For each function call the return address is pushed onto a hardware stack. When a function return instruction is recognised by the hardware, an address is popped from the return stack and is used as the predicted return address. The prediction is checked at the end of the pipeline when the correct return address is available. The return stack is four entries deep.

Pre-decoding
One of the main functions of the PFU is to pre-decode the instruction before it is pushed onto the instruction queue (IQ). Different instruction classes are transformed into separate pre-decode formats and pushed onto the IQ. This reduces the complexity of the main decode blocks, which are already on the critical path, and allows data relating to branches, function calls and returns to be extracted for use by the branch predictor and return stack.

On each cycle, up to two instructions can be pre-decoded and moved into the IQ. The IQ is a FIFO that is capable of reading and writing one or two instructions at a time. The IQ decouples the main decode and execution pipeline in the DPU from the instruction fetch and branch prediction in the PFU. The first two instructions from the beginning of the prefetch queue are formatted, analyzed and can be moved into the IQ.

Dual Issuing Reduces CPI
Dual issuing, or issuing pairs of instructions in parallel, enables the Cortex-R4 processor to achieve a measure of superscalar performance, lowering its CPI without the addition of significant extra resources.

Dual issuing also improves usage of the register file. It is only possible to dual issue instructions when the next two instructions in the stream require the use of complementary, rather than competing datapath resources.

The Cortex-R4 processor's general-purpose register file has been designed with four read ports and two write ports, enabling the supply or consumption of 64-bits of data per cycle, to or from the memory system.

For example, a store double instruction (STRD) must read two registers to supply data to the memory system, and a further two registers for the base and offset to provide an address for the access to the memory system. In order to sustain STRD at a throughput of one per cycle (when accessing cached or TCM addresses), a total of four register file read ports are required.

Similarly, there are a number of instructions that require two write ports to the register file. Most important are the memory load instructions such as load double (LDRD) and load multiple (LDM) that are capable of reading 64-bits of data per cycle from the memory system. Additionally, any load of a single instruction that also requires the base address register to be updated (e.g. LDR r0, [r13, #4]!) requires two register file write ports if both the load and the writeback are to occur in the same cycle.

Because the memory system can only return a maximum of 64-bits of data per cycle, there are few cases that require more than two register file write ports. Such operations include loading 64-bits of data at the same time as base register writeback is required. These cases are rare enough that the extra area and speed cost of a third write port cannot be justified.

Even with just four read and two write ports, there are many instructions that do not utilize all these ports. For example, even a compound data processing instruction such as ADD r5, r6, ASR r0 only requires three read ports and one write port. Any data processing instruction using an immediate shift amount, or no shift at all will require one fewer read ports, and a simple move instruction such as MOV r1, r2 only requires one read port and one write port. Clearly, for a majority of instructions, having four read and two write ports on the register file is overkill. Similarly many instructions only use one of the datapath pipelines in the execute stages of the pipe, leaving the others standing idle.

The dual-issuing of complementary instructions takes advantage of unused register file ports and datapaths to improve the overall utilization of the Cortex-R4 processor resources, effectively providing a low-cost superscalar processor.

Low Cost Implementation

Keeping the dual-issue control logic simple ensures that the area overhead is minimized. The combinations of instructions that can be dual-issued were carefully chosen through a process of modelling and benchmarking to provide maximum performance improvement (Table 2). Many other pairs of instructions have complementary resource requirements, but adding the capability to dual-issue these adds complexity to the design (and therefore area) while being of little benefit.

Category	First Instruction	Second Instruction
A	Any instruction except load/store-multiple (LDM/STM), flag-setting multiplies (eg. MULS), various control instructions (eg. MSR) and any exception instruction (eg. SVC – formerly SWI).	B<condition> #immed IT NOP
B1	Single load instructions with immediate or register offset, or register offset with logical-left shift of less than four. No base-register writeback. eg. LDRGT r10, [r0, r1, LSL #3]	Most data processing instructions, excluding multiplies, that do not require a shift by a value in a register. eg. RSB r3, r2, r6, LSL #1
B2	Single store instructions with immediate or register offset. No base-register writeback. eg. STRH r5, [r1, #2]	
C	Any move instruction, immediate or register, that does not require a shift. eg. MOVW r0, #0x55AA	Most data processing instructions, excluding multiplies. eg. BFIEQ r10, r11, #4, #5

Table 2: Dual issue instruction pairs

To keep the branching and exception logic simple, the processor only allows dual-issue of instructions in the order shown in Table 2. This means that the decoder for the second instruction is only required to decode a subset of the instruction set. In particular it does not need to be capable of generating any control signals for the memory system, or decoding load and store instructions. This reduces the area of the decode logic required. If instructions occur in the wrong order to be dual-issued, only the first instruction will be issued. In the next cycle, the next pair of instructions in the stream will be considered for dual-issue. For example:

a) MOV r10, #0x6C
b) LDR r0, [r12, #0x140]
c) ADD r1, r10, #13
d) LDR r2, [r12, #0x144]
e) ADD r1, r1, r3
f) STR r1, [r12, #0x140]
g) B #0x2000

The above sequence could, if any order of dual-issuing were allowed, be issued in the sequence: ab, cd, ef, g. Since MOV followed by LDR is not allowed, instruction a) must be single-issued, but this allows subsequent pairs of instructions to dual-issue, in the sequence: a, bc, de, fg. The total number of issue cycles is the same in each case. Because of this effect, the ordering restriction does not significantly impair the ability of the processor to dual-issue.

Because the possible dual-issue cases are strictly defined, the datapath resource allocation for any given instruction can be statically determined from its position in the instruction stream. When the decode stage is presented with a pair of instructions, each instruction is decoded independently in such a way that its allocation of resources complements the allocation of the other. Taking an example from the above table:

- LDRGT r10, [r0, r1, LSL #3] : will always be decoded to write the destination register, r10, through write port 0, W0.
- RSB r3, r2, r6, LSL #1 : as the second instruction, will always be decoded to write the destination register, r3, through write port 1, W1.

In this way, after each instruction is decoded, the resulting control signals can simply be multiplexed together to produce a single control for each datapath resource, rather than requiring complex allocation logic, which might otherwise take more time to the extent of needing an additional pipeline stage. Most of the datapath logic is agnostic as to whether it is being driven by the first or second instruction of a pair.

Dual-issuing is also limited by data dependencies between the pairs of instructions, a small number of restrictions on the use of r15 (the program counter) and flag setting. Hazard detection logic in the decode stage determines whether or not a pair of instructions can be dual-issued. If the dual-issue conditions are not met, or if only one instruction is available to be decoded, then one instruction is issued.

Instructions are presented to the decode stage by the instruction queue. When the DPU is busy executing a slow instruction, this allows the PFU to queue up instructions for execution, and whenever there are two or more instructions in the IQ, the DPU may be able to dual-issue them. Similarly, when the memory system takes a number of cycles to return data to the PFU, the IQ provides a buffer of instructions to keep the execution pipeline fed. The IQ also breaks some of the timing paths from the execution pipeline back to the instruction fetch logic.

The PFU is capable of fetching 64-bits of data from the memory system at a time. This data is held in a prefetch queue, and instructions are extracted from it. Each double word of data can contain two or more instructions, and the PFU pre-decode (Pd) stage is capable of extracting two instructions at a time and writing them into the IQ. In this way, the whole pipeline is capable of a sustained throughput of two instructions per cycle, being neither starved by the memory system, nor stalled by the execution pipeline capability.

The dual-issue capabilities, along with the branch prediction mechanism, help to enable the Cortex-R4 processor to achieve performance of 1.6 DMIPS/MHz.

Hardware Divide

The ARMv7R architecture incorporates both signed and unsigned integer divide instructions, and the Cortex-R4 processor incorporates a hardware divider to execute these instructions. The divider uses a Radix-4 algorithm in which it considers two bits, and therefore four possible values, of the resulting quotient per cycle. For a 32-bit divide, the algorithm takes a maximum of 16 cycles to generate the full result. This maximum can be reduced by counting the number of leading zeros in the dividend and divisor and prescaling the data so that the division is only computed for bits that might be non-zero.

This early termination mechanism reduces the number of cycles in many cases, but adds an overhead, since an extra cycle is required to perform the prescale.

The division algorithm does not support signed division, so if this is required, any negative numbers must first be negated before the division is performed. At the end of the division, the resulting quotient can finally be negated if the original operands were of different signs.

Even though it employs a radix-4 algorithm and supports early termination, the hardware divider may still take a large number of cycles to operate. To mitigate the effect this has on performance, the divider, in contrast to the rest of the datapath, can operate out-of-order. This allows other instructions (except divides) to continue to execute, provided they are not dependent on the result of the divide, while the divider calculates the result.

When a divide instruction is executed in the Cortex-R4 processor pipeline, it is split into two parts. A phantom instruction is sent down a control pipeline, in lock-step with the other pipelines. At the same time, the operands for the division are sent to the divide unit, and the calculation is started. The phantom instruction carries with it the condition codes for the divide, and also maintains the ordering of the program. If the phantom fails its condition codes, or is flushed from the pipeline as a result of a preceeding mispredicted branch or an exception, then the divide calculation is terminated. Once the phantom has reached the end of the pipeline, the processor commits to finishing the divide calculation.

At the end of the divide calculation, the divider signals that it has finished, and the pipeline stalls for one cycle in order to allow the result of the divide to be written to the register file. This operation re-uses the logic in the main pipeline to perform the final negation, if required, for a signed divide.

Interrupt Handling

Interrupt latency is an important parameter for many deeply embedded systems, and a number of new features incorporated within the Cortex-R4 processor dramatically reduce the latency associated with interrupt handling.

Multi-cycle instructions can be terminated before completion after they have been initiated. This includes load and store multiples (LDM, STM) to normal memory.

The Cortex-R4 processor has an interface for an interrupt controller to supply the vector address of the required IRQ service routine. This allows a dedicated hardware unit (not part of the Cortex-R4 processor) to perform a look-up of the start address of the handling routine for the highest priority outstanding interrupt. This interface requires a handshake from the Cortex-R4 processor to acknowledge when a particular interrupt has been taken.

The interrupt interface is designed to handle interrupts managed by a controller which is clocked either synchronously or asynchronously to the main clock of the Cortex-R4 processor. This capability enables the controller to be used in systems which have either a synchronous or an asynchronous interface to the processor clock and the AXI clock.

This new functionality within the Cortex-R4 processor means that the maximum response time is reduced from 118 cycles for the ARM946E-S processor, to 20 cycles in the Cortex-R4 processor.

A non-maskable interrupt option is also available on the Cortex-R4 processor, preventing software from disabling the fast interrupt requests (FIQ). This is particular important for safety critical applications.

Memory Architecture

Selecting the right memory architecture is key to attaining maximum system performance and cost optimization. Deciding between the use of SRAM and ROM, the size and partitioning of the cache or tightly coupled memories (TCM), are choices that are fundamental to the system design and depend upon the real-time constraints of the application and the flexibility of the processor.

The Cortex-R4 processor incorporates a Harvard level 1 memory system. This consists of optional caches, optional MPU and optional TCM interfaces. These features can be omitted at synthesis in order to save silicon area. This level of configuration provides flexible and efficient support to implement a memory architecture that is highly suited to the application requirements.

By supporting two-cycle level 1 memory accesses, the Cortex-R4 processor relaxes the required response time of the RAM compared with the processor's clock frequency. Having the flexibility to specify lower speed on-chip memory reduces area, power consumption and cost. With less stringent timing demands, the processor becomes easier to harden since attaining timing closure will take less time and effort.

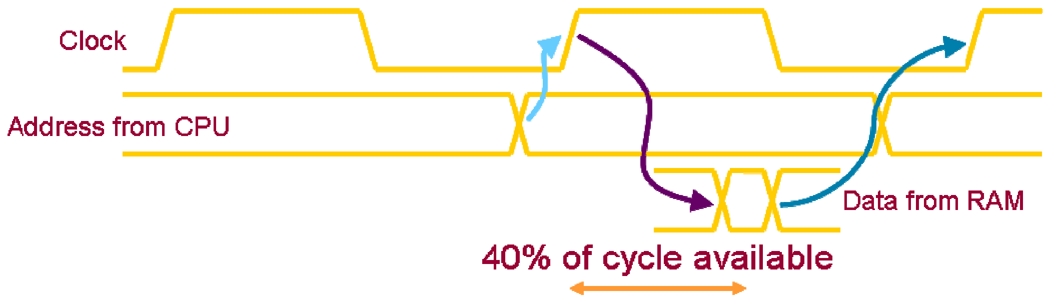

Figure 2. Cortex-R4 processor Memory Access Timing

Tightly-Coupled Memory (TCM) Support

Cortex-R4 processor provides flexible support for tightly-coupled memories. Up to three 64-bit memory ports are available. Each port has independent wait and error signals for connecting RAM, ROM, e-DRAM, and error correction logic.

The ports are grouped into two interfaces – A and B. Interface B has two ports to support greater bandwidth through interleaved access. This configuration enables the TCM to be organized as a single bank of memory on interface A for the lowest system cost, which is just 5 percent slower than the fastest configuration; or as two interleaved banks on ports B0 and B1 for a 'medium cost' system that is just 2 percent slower than the fastest configuration; or as three banks using all the ports which gives the best performance.

The key benefit of providing support for interleaved TCM-B is apparent when DMA into the TCM (via the slave port) occurs at the same time as data access from the LSU to the TCM. The interleaving makes the most significant difference if sequential data is being accessed. If both DMA and the LSU try to access the same RAM initially, the slave will stall on the clash, and then the two will continue to access complementary RAMs. Interleaved memory support is particularly useful if the processor is working on frames of data. The next frame can be DMA'd into the TCM while the processor is working on the current frame.

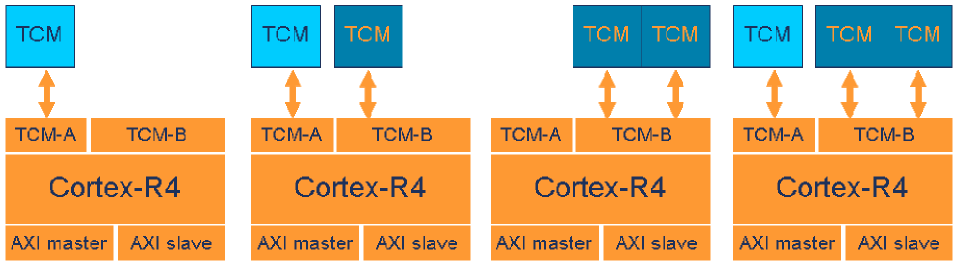

Figure 3. TCM configurations, from low cost to maximum performance.

The TCM interfaces differ from the main AMBA memory interface in that they are optimized for low latency memory access. The Cortex-R4 processor issues speculative read accesses on these interfaces and as such it is not suitable for read sensitive memory or peripherals. Memory access attributes are not exported on this interface. The TCM interfaces have wait and error signals to support error detection, correction and slow memories.

The TCMs are composed of a TCM-A and TCM-B interface, which may be connected to zero wait state memory. Each interface is also waitable, and has an error input allowing complex external error identification and correction. The processor also supports parity generation and parity error detection and, as with the ECC support, is configurable on a per port basis.

Unlike previous ARM processors there is no restriction on the data type that can be stored in the TCMs, allowing symmetrical access from the I-side and D-side. On earlier processors, access to the ITCM and DTCM from the opposing side would result in larger latencies, and instructions fetches could not be made from the DTCM. With the Cortex-R4 processor, the only restriction is that any LSU access takes priority over the PFU.

ARM9E family processors use a fixed-size memory map for the TCM regions, which means that a smaller TCM would be aliased over the region. With the Cortex-R4 processor, the TCM size is pin configurable which means that the TCM regions can exactly match each TCM size.

Caches

The Cortex-R4 processor supports separate instruction and data caches. Each cache is a physically addressed 4-way set associative cache with a line length of 8 words. The cache sizes can be independently varied from 4KB to 64KB. Both the instruction and data caches are capable of providing 64-bits per cycle to the processor. The caches may be disabled independently from the TCM.

Cache Miss Handling

A cache miss results in the requests required to do the line fill being made to the level 2 (L2) memory interface, and a write-back occurring if the line to be replaced contains dirty data. The cache supports allocate on read miss and allocate on write miss. The write-back data is transferred to the write buffer, which is arranged to handle this data as a sequential burst.

The caches perform critical word first cache refilling. The internal bandwidth from the L2 data read port to the data caches is 8 bytes/cycle, and supports streaming.

The processor can also abandon a linefill request in order to accelerate interrupt entry. The linefill will still complete on the bus (and be written into the cache) because an AXI burst will have been committed to, but the processor can continue with fetching data and instructions from the cache or TCM.

AMBA 3 AXI Memory Interface
The Cortex-R4 processor includes a 64-bit AMBA 3 AXI memory interface, which supports high-bandwidth transfer to second level caches, on-chip RAM, peripherals and interfaces to external memory.

AMBA 3 AXI supports the issuing of multiple outstanding addresses and for data to be returned out of order. The most significant advantage for many applications will be that a slow slave does not block the bus for the duration of its access, allowing the processor to perform further accesses rather than waiting for completion of the slow access.

The Cortex-R4 processor has an AMBA AXI slave port to allow masters on the AMBA AXI system access to its RAMs. This supports full DMA access to the TCMs.

Memory Protection Unit

For certain markets memory protection is vital, for example safety-critical automotive applications. For other applications an MPU can be a useful software development tool, allowing out of range memory accesses to be quickly trapped.

The Cortex-R4 processor includes an optional memory protection unit (MPU). This can be configured with 8 or 12 regions, combining flexibility with area efficiency. If the MPU is omitted completely, this results in a fixed mapping of protection attributes. The minimum size of an MPU region is 32-bytes, allowing fine control and reduced memory wastage.

Managing RAM Errors

Smaller geometry semiconductor processes are more susceptible to soft errors caused by neutrons and alpha particles. A minimum level of parity protection has been a required feature of safety critical systems for some time.

The Cortex-R4 processor includes extensive support for parity detection and error correction codes (ECC).

TCM Errors

Since the TCM interface is external, designers have full flexibility to develop a memory system with the most appropriate level of error detection and correction to suit the application.

Parity and ECC support is facilitated through byte-wise parity inputs and outputs, as well as error and wait inputs. The wait pin signals the Cortex-R4 processor to wait while error correction is performed. The error pin allows the external memory system to signal errors from more complex error-checking schemes. While implementing parity support requires no additional external logic, ECC logic must be implemented by the design team outside of the processor.

Each of the three TCM ports provides independent error and wait inputs, the configuration registers enabling parity and error checking on an individual TCM basis.

I and D Cache Errors

Cache RAMs include a synthesis option to enable parity support. If parity generating and checking logic is implemented, wider RAMs are also required to incorporate the parity bits. Parity logic is only generated if this option is selected. One parity bit is stored per byte in the data RAMs, while in the tag and dirty RAMs, one parity bit is stored per set/way combination.

Odd or even parity is selected by pin configuration. The processor generates parity for each byte of data and the address tag, and checks the parity bits returned. With the parity bits present in each RAM, it is possible to detect and correct a parity error at the expense of a linefill.

When the processor detects a parity error, there are two options available. These can be selected and controlled in software. If both options are turned off, parity is disabled.

- o Hardware correction: The processor can be configured to automatically invalidate the faulting cache line (forcing the cache to be Write Through). The corresponding instruction is re-executed. Second time around the instruction misses in the cache and the correct data is fetched from the external memory system.
- o Software notification: A precise abort can be generated on the instruction or data access to allow the software to fix the error. This allows error profiling and process termination in fatal circumstances. The Data Fault Status Register (DFSR) allows the abort type to be determined – AFSR indicates a set and way fault. The Data Fault Address Register (DFAR) indicates the address that generated the abort. Provided the cache line is not dirty, the data abort handler can invalidate it and rely on the external memory to provide an error-free copy of the data when it is next required by the processor. If the cache line is dirty, then the parity error may be unrecoverable.

Hardware correction can be enabled at the same time as software notification. In this case, the cache line is automatically invalidated, but a data abort is also taken, which can be useful for monitoring parity errors that occur. Parity can be disabled using the config register in CP15. This register includes separate enables for instruction and data cache.

RAM BIST

All RAM blocks can be tested by BIST through a Cortex-R4 processor BIST interface. This enables flexibility in selecting the BIST methodology, allowing appropriate algorithms to be selected for different RAM topologies.

Dedicated BIST ports provide an optimised path to the controls of the RAMs, which avoids introducing frequency-limiting critical paths into the design.

Managing Logic Errors

For safety critical systems there is a synthesis option to include a redundant copy of the processor logic to be instantiated in a design so that external logic can compare the outputs of both processors in order to check for soft and hard errors. The interface allows the redundant processor to share the RAMs of the master processor to help keep the overall system area down. Otherwise the redundant processor contains all of the expected Cortex-R4 processor logic. The comparison logic is not part of the Cortex-R4 processor.

Enhancing Performance

The microarchitecture enhancements that have been implemented in the Cortex-R4 processor such as dual issuing, branch prediction and a pipeline that allows single-cycle load-use penalty are aimed at delivering more work per cycle. Improved efficiency reduces the CPI, enabling the processor to run more slowly while delivering the same performance. Enabling a lower frequency clock allows reduced supply voltages which benefit power and can also positively affect hardware reliability.

Dhrystone MIPS benchmark measurements show that compared with the ARM946E-S processor the Cortex-R4 processor increases the DMIPS delivered per MHz by more than 40 percent, while the maximum frequency may increased by more than 30 percent.

- o ARM946 DMIPs/MHz = 1.14. Max f = 210MHz (0.13µm)
- o Cortex-R4 DMIPs/MHz > 1.6. Max f = 280MHz (0.13µm)

Cortex Product Family

The Cortex Family of processors provides a range of solutions optimized around specific market applications across the full performance spectrum. This underlines ARM's strategy of aligning technology around specific market applications and performance requirements.

The ARM Cortex family comprises three processor series, which all implement the Thumb-2 instruction set to address the increasing performance and cost demands across a range of markets:

ARM Cortex-A Series, applications processors for complex OS and user applications. Supports the ARM and Thumb-2 instruction sets

ARM Cortex-R Series, embedded processors for real-time systems. Supports the ARM and Thumb-2 instruction sets

ARM Cortex-M Series, deeply embedded processors optimized for cost sensitive applications. Supports the Thumb-2 instruction set only.

Summary

The Cortex-R4 processor delivers improved performance, cost and power savings across a broad range of deeply-embedded applications. Significant microarchitectural enhancements include the use of Thumb-2 and a new pipeline design that implements instruction prefetch and branch prediction to enable an appropriate level of performance to be achieved at moderate clock frequencies. This approach is fundamental to delivering low power and reduced cost implementation.

In addition, modifications to TCM and DMA implementation, as well as the provision of a single AMBA 3 AXI interface, ensure that the Cortex-R4 processor is easier to integrate than other cores. Relaxed level 1 memory timing ensures significant system area and power savings can be achieved, and make it easier to meet synthesis timing constraints during the development process.

Further features enhance the Cortex-R4 processor's deeply embedded credentials for specific applications. The more flexible MPU allows greater levels of OS protection. The improved interrupt capabilities permit exception entry time to be dramatically reduced compared with the ARM946E-S processor. More detailed comparison with the ARM9E processor family is available [1].

References

[1] ARM White Paper:
Cortex R4. A comparison with the ARM9E processor family. Peter Lewin, May 2006