THE EVOLUTION AND IMPROVEMENT OF THE "ARM " ARCHITECTURE
(The ARM Architecture (Version 6)ARM6.)

Abstract:

A microprocessor's architecture defines the instruction set and programmer's model for any rocessor that will be based on that architecture. Different processor implementations may be built to comply with the architecture. Each processor may vary in performance and features, and be optimized to target different applications. Future processors, based on the new ARMv6 architecture will provide developers of embedded systems with higher levels of system performance, whilst maintaining excellent power and area efficiency.

The introduction of the ARMv6 architecture brings a new set of features and a performance leap that will meet the needs of ARM's partners as they design nextgenerationproducts across a range of target markets. ARMv6 consolidates the developments in ARMv5, and provides 100% backwards compatibility. It also adds significant enhancements for next-generation applications.

New multimedia support provides 4x-processing improvements in some media applications. The new VMSA provides faster context switches enhancing performance of platform processors hosting complex operating systems. Improved multiprocessor support eases development and enhances the performance of systems based on multiple ARM cores, or ARM plus DSP core configurations.

Keywords: *ARM,VMSA,DSP,MULTIMEDIA*

Conclusion:

Functionality is growing dramatically as computing and communications continue to converge in many consumer products. Increasingly, consumers expect features such as advanced user interfaces, multimedia capability and improved productquality. ARMv6 will enable more efficient support for all of these new features andtechnologies across a number of market segments.A number of specific market drivers for ARMv6 have been identified. ARMv6 will benefit developers targeting wireless, networking, automotive and consumer entertainment markets.

1. Introduction:
(The Evolution Of The ARM Architecture)

The ARM architecture has evolved steadily to respond to the changing needs of ARM's partners, and of the design community in general. At each major revision of the ARM architecture, significant features have been added. Between major architecture visions, new features have been included as variants on the architectures. The key letters appended to the core names indicate specific architecture enhancements within each implementation. V3 introduced 32-bit addressing, and architecture variants:

 T - Thumb state: 16-bit instruction execution.

 M - long multiply support (32 x 32 => 64 or 32 x 32 + 64 => 64). This feature became standard in architecture V4 onwards.

. V4 added halfword load and store.

. V5 improved ARM and Thumb interworking, count leading-zeroes (CLZ) instruction, and architecture variants:

 E - enhanced DSP instructions including saturated arithmetic operations and 16-bit multiply operations

 J - support for new Java state, offering hardware and optimized software acceleration of bytecode execution.

All of the 'TEJ' enhancements above become part of the new ARMv6 architecture specification.

In order to maintain backwards compatibility, ARMv6 also includes ARMv5 compliantmemory management and exception handling. This enables the significant third-partydeveloper community to exploit existing development effort, and supports the reuse ofexisting software and design experience.

The introduction of a new architecture does not replace existing architectures, or make them redundant. Where the provisions of ARMv4 or ARMv5 meet market needs, new cores and derivative products will continue to be based on these architectures, whilst tracking technology and process trends. For example, the ARM7TDMI core based on the V4T architecture is still being 'designed-in' to many new products, where a performance level of 100MIPS or so is adequate. Processors based on the ARMv5 architecture continue in development. The ARM architecture will of course continue to evolve with appropriate enhancements in the future.
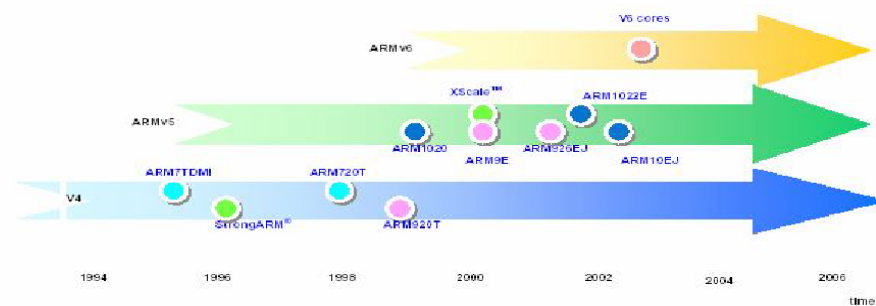

Figure 1. ARM Architecture Revisions

2 Driving Architecture Development:

Next generation architectures have been driven by the needs of emerging products and evolving markets. The key design constraints are predictable. The function, performance, speed, power, area and cost parameters must be balanced to meet the requirements of each application. ARMv6 offers better ways of optimizing these constraints across a number of vertical market segments. Delivering leading performance/power (MIPS/Watt) has been fundamental to ARM's success in the past, and will continue to be a critical benchmark for

future applications.

3.Key Armv6 Improvements:

In developing the ARMv6 architecture, effort has been focused on five key areas:

i)Memory Management:

System design and performance is heavily affected by the way that memory is managed. The memory management architectural enhancements improve the overall processor performance significantly - especially for platform-type applications where operating systems need to manage frequent task changes. With the changes in ARMv6, average instruction fetch and data latency is greatly reduced; the processor has to spend less time waiting for instructions or data cache misses to be loaded. The memory management improvements will provide a boost in overall system performance by as much as 30%. In addition, the memory management enhancements will enable more efficient bus usage. Less bus activity will yield significant power savings as a result of reduced memory access.

ii)Multiprocessing:

Application convergence is driving system implementations towards the need for multiprocessor systems. Wireless platforms, especially for 2.5G and 3G, are typical applications that demand integration between ARM processors, ARM and DSPs, or other application accelerators. Multiprocessor systems share data efficiently by sharing memory. New ARMv6 capabilities in data sharing and synchronization will make it easier to implement multiprocessor systems, as well as improving their performance. New instructions enable more complex synchronization schemes, greatly improving system efficiency.

iii)Multimedia Support:

Single Instruction Multiple Data (SIMD) capabilities enable more efficient software implementation of high-performance media applications such as audio and video encoders. Over sixty SIMD instructions are added to the ARMv6 Instruction Set Architecture (ISA). Adding the SIMD instructions will provide performance improvements of between 2x and 4x, depending on the multimedia application. The SIMD capabilities will enable developers to implement high-end features such as video codecs, speaker-independent voice recognition and 3D graphics, especially relevant for next generation wireless applications.

iv)Data Handling :

A system's endianism refers to the way data is referenced and stored in a processor's memory. With increasing system on a chip (SoC) integration, a single chip is more likely to contain little-endian OS environments and interfaces (such as USB, PCI), but with bigendian data (TCP/IP packets, MPEG streams). With ARMv6, support for mixed-endian systems has been improved. As a result, handling data in mixed-endian systems under ARMv6 is far more efficient. Unaligned data is data that is not aligned to its natural size boundary. For example, within DSP applications there is sometimes a requirement to treat words with half-word data alignment. For a processor to handle this situation efficiently requires that it be able to load a word aligned to any half-word boundary. Current versions of the architecture require a number of instructions to manage unaligned data. ARMv6 compliant architectures will manage unaligned data more efficiently in hardware. In algorithms that rely heavily on DSP operations with unaligned data, ARMv6 implementations will have a performance advantage and may also benefit from reduced code size. Unaligned support also makes it more efficient for ARM to emulate other processors, such as Motorola's 68000 family. Similar to recent ARMv5 implementations such as ARM10 and XScale1™, ARMv6 is based on a 32-bit processor. ARMv6 will support implementations based on bus widths of 64-bits and above - ARM10 and XScale support 64-bit buses today. This provides bus throughput equivalent to, or even better than a 64-bit machine, but without the power and area overhead of a full 64-bit CPU.

v)Exceptions and Interrupts:

For implementations targeted at real-time systems, efficient handling of interrupts can be critical.

Examples include systems such as hard disk controllers, and engine management 1 XScale is a registered trademark of Intel Corporation. applications, where the consequences

can be severe if a critical interrupt does not get serviced in time. More efficient handling of exception and interrupt conditions also improve overall system performance. This is especially important in reducing system latency. In ARMv6, new instructions have been added to the ISA to improve the implementation of interrupts and exceptions. These provide the ability to efficiently nest exception handling onto a different privileged mode.Each of these architectural advances is described in more detail in the following sections.

4.Compatibility:

ARMv6 maintains 100% backward compatibility at the binary level for operating systems and applications. The ARMv6 architecture requires that all Thumb and 'E' instructions be implemented for backwards compatibility with ARMv5.

Some of the newly introduced ARMv6 instructions also have Thumb equivalents - for example the new 'REV*' instructions. The BXJ instruction is also a requirement within ARMv6 for consistent Java support - regardless of whether Jazelle technology is implemented or not.

5.Improved Memory Management:

Memory management is primarily concerned with two issues. First, the translation of virtual addresses into physical addresses within a system. Second, ensuring appropriate levels of protection between different processes and tasks.

The ARM architecture is a load-store architecture, where the ARM core instructions canonly operate on data in registers that form part of the core. Load and store instructions areused to transfer data to and from this register file.

A multi-level memory system is part of normal system design hierarchy. Closer coupledmemory systems tend to run faster, with level 1 memory systems ideally having no waitstates. In practical terms, this limits the size of memories that can be supported at coreclock speeds. Many high performance systems are now supporting additional (larger) L2caches with some wait states, but less latency than if the memory was located off-chip. L3 cache may be provided as fast off-chip SRAM, with "normal" DRAM a level behindthat.ARM first introduced cores (e.g. ARM7TDMI), then developed and offered cached coreswith MMU's (e.g. ARM720/920).

ARMv6 is a logical progression on this - providing acomplete definition of the L1 memory system, and to a lesser extent how memory levelsbeyond this need to behave for overall system correctness.
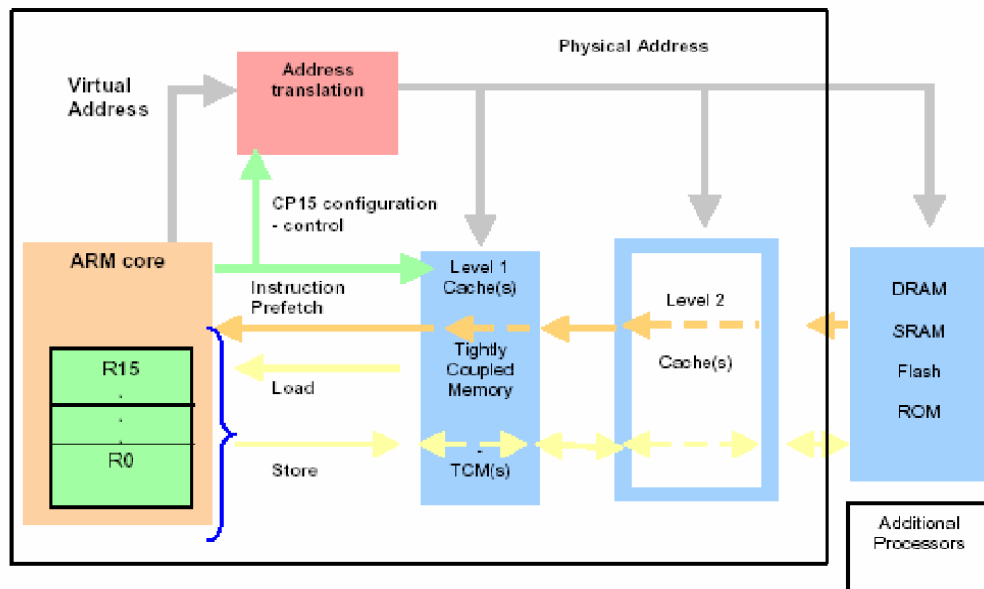

Figure 2. ARMv6 Memory Model

L1 memory will run synchronized to the core. Where different clock domains are introduced into a design, memory synchronization becomes dependent on the implementation.

6 Arm Virtual Memory System Architecture:

The ARM Virtual Memory System Architecture v6 (VMSAv6) fully specifies the new Level 1 cache system - that most tightly coupled to the processor. The VMSA also specifies a Tightly-Coupled Memory (TCM) and DMA system. The architecture permits a range of implementations of these systems, with software-visible configuration registers to allow identification of the resources that exist. V6 supports hierarchy and memory ordering rules to ensure system correctness for additional levels of cache in both single processor and multiprocessor systems. Memory ordering rules define the architecture, without constraining the implementation .Version 6 now supports physically tagged caches, reducing software overhead on context switches. This can save up to 20% of the processor utilization by eliminating the need to perform cache flushing by the OS.

i)ARM v6 L1 Cache:
The L1 cache is architected to reduce the requirement for cache clean and invalidation on a context switch. The cache may be organized as a Harvard system with separate instruction and data caches, or as a single unified von Neumann cache. The TCM is a physically-addressed area of scratchpad memory, which is implemented alongside the L1 cache. Similarly, the TCM can be organized as a Harvard or von Neumann system. The L1 DMA subsystem is designed to allow background transfers to and from the TCM.

Page Table Formats:
Page table formats have been revised in ARMv6. Figure 3 illustrates the new first level page table format.

SBZ											0	O
Coarse page table page address						P	Domain	SBZ		0	1
Section base address	SBZ	nG	S	APX	TEX	AP	P	Domain	XN	C	B	1	O
Reserved											1	1

Figure 3. ARMv6 First Level Page Table Format

The XP bit in Coprocessor 15 is used to enable this format, otherwise an ARMv5 legacy mode is invoked for backwards compatibility.
New features include:
. an execute never bit (XN)
. a "not Global" (nG) bit for address matching
Application Space Identifier - or ASID - support is another key feature in this area. When the nG-bit is set, address translation uses the virtual address and ASID for translation matching. This provides a significant saving in software overhead on context switches, avoiding the need to flush on-chip translation buffers in most cases. The result is improved performance. The architecture also supports its use in task- aware debugging.
The ASID forms part of a process ID that can be used in task aware debugging.
Type extension, shared, and access permission bits are used to provide all the attributes necessary for the ARMv6 memory model. A P-bit, which is compatible with the mechanism already available on Intel's XScaleTM product, has been added for memory protection.

Multiprocessing:

While many ARM processors today are used in isolation, or with simple communications links to another resource with its own memory, there are increasing requirements for unified memory models, and closer coupling of processors in general. Systems consisting of multiple processors - either multiple ARM processors or a mixture of ARMs and DSPs, are becoming more common. Improvements to the ARMv6 memory management unit (MMU) are important in ensuring that processors get predictable and consistent (coherent) views of memory when it is shared between multiple processors Improvements include defining the level1 memory system, and the memory order model

- how loads and stores to memory relate to each other.

ii)ARMv5 ARMv6:

FIQ2handler. FIQs are now re-enabled, with original R2, R3, R14, SPSR on stack. Includes code to stack any more registers required, process the interrupt and unstack extra registers.

ARMv5	ARMv6
FIQ2handler. FIQs are now re-enabled, with original R2, R3, R14, SPSR on stack. Includes code to stack any more registers required, process the interrupt and unstack extra registers.	

`STMIA   R13, {R0-R3}`
`MOV     R0, LR`
`MRS     R1, SPSR`
`ADD     R2, R13, #8`
`MRS     R3, CPSR`
`BIC     R3, R3, #0x1F`
`ORR     R3, R3, #0x1B   ; = Abort mode No.`
`MSR     CPSR_c, R3`
`STMFD   R13!, {R0,R1}`
`LDMIA   R2, {R0,R1}`
`STMFD   R13!, {R0,R1}`
`LDMDB   R2, {R0,R1}`
`BIC     R3, R3, #0x40   ; = F bit`
`MSR     CPSR_c, R3`
	`SUB     R14, R14, #4`
	`SRSFD   R13_abt!`
	`CPSIE   f, #0x1B   ; = Abort mode`
	`STMFD   R13!, {R2,R3}`

Exit code including the LDR/STR instructions needed to acknowledge the VIC	

`ADR     R2, #VICaddress`
`MRS     R3, CPSR`
`ORR     R3, R3, #0x40   ; = F bit`
`MSR     CPSR_c, R3`
`STR     R0, [R2,#AckFinished]`
`LDR     R14, [R13,#12] ; Original SPSR value`
`MSR     SPSR_fsxc, R14`
`LDMFD   R13!, {R2,R3,R14}`
`ADD     R13, R13, #4`
`SUBS    PC, R14, #4`
	`LDMFD   R13!, {R2,R3}`
	`ADR     R14, #VICaddress`
	`CPSID   f`
	`STR     R0, [R14,#AckFinished]`
	`RFEFD   R13!`

Approximate cycles: 35	Approximate cycles: 11

Table 1a. Efficient code handling in ARMv6

```
Entry code:
add R2, R3, R14 and SPSR to the target (ABORT) stack
        switch mode => ABORT

exit code:
recover R2 and R3 context
        return from handler (pop values from the ABORT stack)
            - "LR" => PC
            - "SPSR" => CPSR
```

Table 1b. Entry/Exit code handling in ARMv6

For ARMv5 the FIQs are disabled for some time at the start of the lower-priority FIQs. The worst-case interrupt latency for the FIQ1 interrupt occurs if a lower- riority FIQ2 has just fetched its handler address, and is approximately:

. 3 cycles for the pipeline refill after the LDR PC instruction fetches the handler address

. + 24 cycles to get to and execute the MSR instruction that re-enables FIQs

. + 3 cycles to re-enter the FIQ exception

. + 5 cycles for the LDR PC instruction at FIQhandler

. or about 35 cycles.

For ARMv6, the worst-case interrupt latency for a FIQ1 now occurs if the FIQ1 occurs during a

FIQ2's interrupt entry sequence, just after it disables FIQs, and is
   approximately:
. 3 cycles for the pipeline refill for the FIQ2's exception entry sequence
. + 5 cycles to get to and execute the CPSIE instruction that re-enables FIQs
. + 3 cycles to re-enter the FIQ exception
. or about 11 cycles.
The underlying mechanism illustrated can be used from any privileged mode, to stack
and swap state to a different privileged mode, then return from this mode using the stack
values.

Data Handling:
Version 6 has introduced two features for mixed-endian support:
E-bit:
A state bit (E-bit) is set and cleared under program control using the SETEND
instruction. The E-bit defines which endian to load and store data. Figure 4 illustrates the
functionality associated with the E-bit for a word load or store operation.
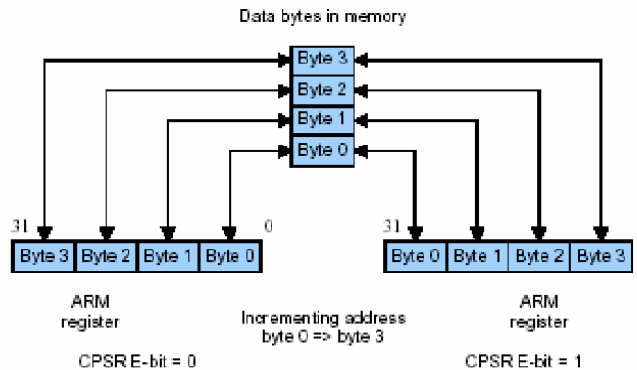

Figure 4. Endian support - Word Load/Store with E-bit

This mechanism enables efficient dynamic data load/store for system designers who
know they need to access data structures in the opposite endianness to their
OS/environment. Note that the address of each data byte is fixed in memory. However, the
byte lane in a register is different
iii)REV Instructions:
Three byte reverse instructions are available in both ARM and Thumb states. The byte
reverse (REV) instructions can be used to improve byte-swap routines present in
many code bases today typically replacing four instructions with a single instruction (Figure 5)
New instructions (ARM and Thumb variants)
. REV - byte reverse a word
. REV16 - byte reverse packed (2 x) halfwords
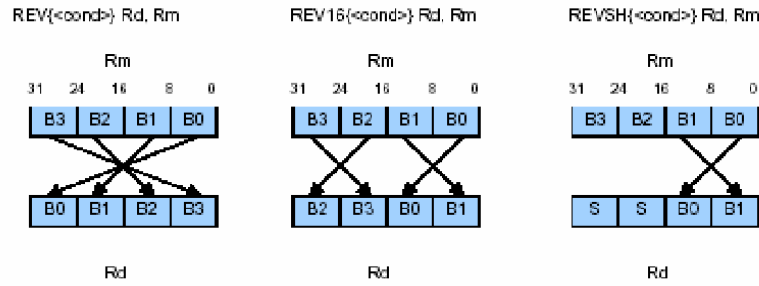. REVSH - byte reverse + sign extend halfword

Figure 5. ARMv6 Byte Reverse Instructions

ARMv6 provides better support for the sum of absolute differences calculation, with the inclusion of the USAD8 (sum of differences) and USADA8 (sum of differences and accumulate) instructions. These are particularly useful for video encoding and motion estimation applications.

Table 3 shows the relative performance of the sum of absolute differences. The comparison with version 5TE relates to a software implementation in ARM registers. This can also be accelerated with the MOVE coprocessor