ARM6 a High Performance Low Power Consumption Macrocell

Mike Muller

mmuller@armltd.co.uk

Advanced RISC Machines Limited
Park End, Swaffham Bulbeck, Cambridge England CB5 0NA

Abstract

The ARM6 is an ASIC CPU macrocell of 35530 transistors which offers high performance (28K Dhrystones at 20MHz) from an extremely small die size (3.1mm x 1.9mm on a 0.8micron CMOS process) and consumes very low levels of power (54mWatts @ 20MHz @ 3Volts). The device has been designed to allow it to be easily incorporated into larger chip designs to meet the requirements of many of todays highly integrated products. For example, the ARM610 comprising ARM6, 4kBytes Cache, Write Buffer and Memory Management Unit and the ARM250; a single chip computer with integrated Video Controller, Memory Management Unit, direct ROM and DRAM interface and IO controller with counter timers and serial interface.

1: Introduction

The ARM6 is an ASIC CPU macrocell of 35530 transistors which offers high performance (28K Dhrystones at 20MHz) from an extremely small die size (3.1mm x 1.9mm on a 0.8micron CMOS process) and consumes very low levels of power (54mWatts @ 20MHz @ 3Volts). The device has been designed to allow it to be easily incorporated into larger chip designs, using full-custom or ASIC tools, to meet the requirements of many of today's highly integrated products. For example, the ARM610 comprising ARM6, 4kBytes Cache, Write Buffer and Memory Management Unit and the ARM250; a single chip computer with integrated Video Controller, Memory Management Unit, direct ROM and DRAM interface and IO controller with counter timers and serial interface.

ARM achieves this small size, low power consumption and high code density by using RISC and CISC design techniques as well as careful attention to VLSI implementation. Examples of this are the conditional execution of all instructions, the multi-cycle block data transfer instructions, and a barrel shifter the use of which may be specified in any data processing operation.

1.1: ARM610

The ARM610 is a general purpose 32-bit microprocessor of 359,000 transistors with 4kByte cache, Write Buffer and Memory Management Unit (MMU) combined in a single chip, see Figure 1. The ARM610 is software compatible with the ARM family and can be used with the existing ARM support chips, MEMC10, MEMC20, IOC and VIDC10 and VIDC20.

The MMU supports a conventional two-level page-table structure and a number of extensions which make it ideal for embedded control, UNIX and Object Oriented systems. The ARM6 family is fully static and has been designed to minimise its power requirements. This makes it ideal for portable applications where both these features are essential. The ARM610 is available in a 144-pin TQFP only 1.4mm thick at around $25 in quantities of 100,000.

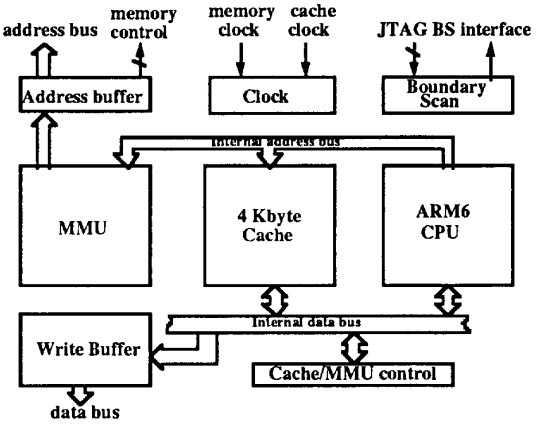

Figure 1 ARM610

1.2 ARM250

The ARM250 comprises 4 main blocks; the ARM processor, the Memory Controller (MEMC), Video Controller (VIDC) and IO Controller (IOC and IOEB), see Figure 2. It connects directly to DRAM and ROM with programmable timings. The interface uses fast-page-mode accesses where possible to maximize memory bandwidth. The ARM250 will deliver 14,800 Dhrystones using low cost DRAM without the need for a cache.

The MEMC block includes a 3 channel DMA controller and a 128-entry MMU that maps physical memory onto virtual address space. Each entry can map 4K, 16K, or 32K pages and three levels of protection are available.

The VIDC connects directly to a CRT display and supports 1-, 2-, and 4-bits per pixel at up to 800 x 600. The video output timing and polarity are fully programmable. In addition two eight-bit audio DAC's are provided capable of generating stereo sound.

The IOC provides a simple 8-bit I/O bus by latching data from the main data bus. The I/O bus can be extended to 16 bits. The I/O bus may be clocked at a lower frequency than the main bus. This allows video refresh cycles to use the main bus during an I/O access. The IOEB logic provides a variety of I/O timing options for each device including an ISA-style interface. IOC also includes a serial port that can run asynchronously at up to 31,250 bits per second. Three 16-bit timers are available; two can be used to generate timed interrupts while the third generates the clock for the serial port.

The chip is built on a 1.0 micron CMOS process with two layers of metal. The die is 58mm^2 and uses 98,019 transistors. The ARM250 is available in a 160-pin PJQFP at a price of around $25 in quantities of 100,000.

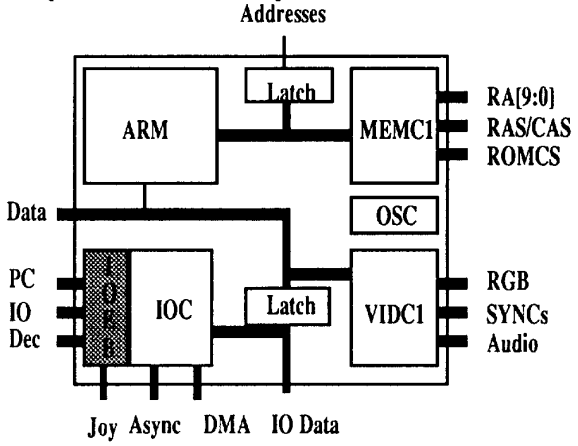

Figure 2 ARM250

2: ARM6 overview

ARM6 has a 32 bit data bus and a 32 bit address bus. The data types supported are Bytes (8 bits) and Words (32 bits), where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations (e.g. ADD) are only performed on word quantities. Load and store operations can transfer either bytes or words.

ARM6 supports six modes of operation:-

(1) User: the normal program execution state

(2) FIQ: to support a data transfer or channel process

(3) IRQ: used for general purpose interrupt handling

(4) Supervisor: a protected mode for the operating system

(5) Abort: entered after a data or instruction prefetch abort

(6) Undefined: entered if undefined instruction is executed

Mode changes may be made under software control or may be caused by external interrupts or exception processing. Most application programs will execute in User mode. The other modes, known as *privileged modes*, will be entered to service interrupts or exceptions or to access protected resources.

2.1: Instruction set

The ARM6 instruction set comprises ten basic instruction types. Two of these make use of the on-chip arithmetic logic unit, barrel shifter and multiplier to perform high-speed operations on data in a bank of 31 registers, each 32 bits wide. Three classes of instruction control the transfer of data between main memory and the register bank, one optimised for flexibility of addressing, another for rapid context switching and the third for swapping data. Two instructions control the flow and privilege level of execution, and another three types are dedicated to the control of external co-processors which allow the functionality of the instruction set to be extended on or off-chip in an open and uniform way. A summary of the instruction set is show in Figure 1.

The ARM instruction set has proved to be a good target for compilers of many different high-level languages and is significantly denser than other 32bit RISC processors and is comparable to many 32bit CISC processors [1]. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors which depend on sophisticated compiler technology to manage complicated instruction interdependencies.

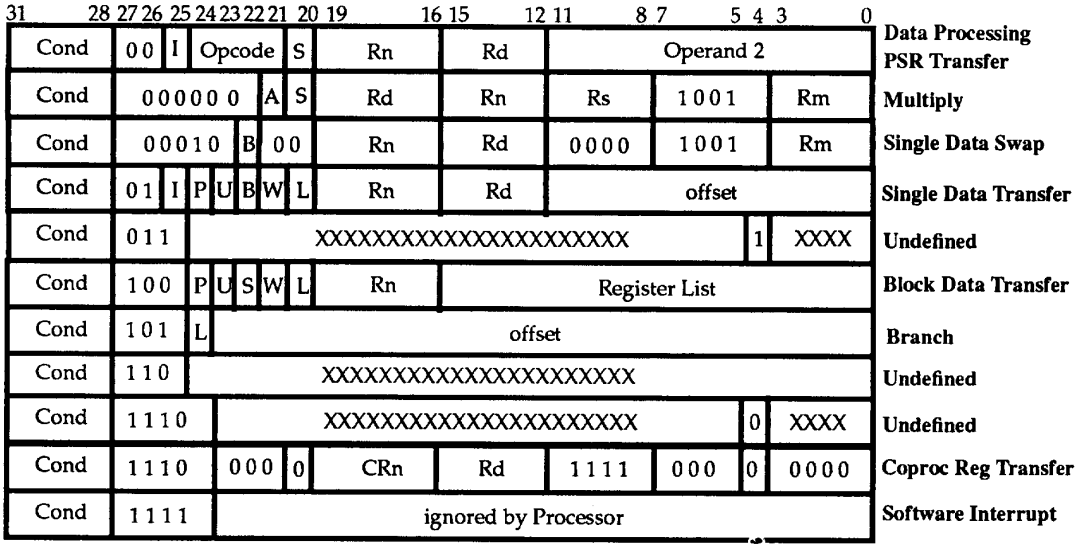

Figure 3 : Instruction Set Summary

2.2: Memory interface

The memory interface has been designed to allow the performance potential to be realised without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals permit the exploitation of paged mode access offered by industry standard DRAM.

2.3: Registers

The processor has a total of 37 registers made up of 31 general 32 bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode and the other registers (the *banked registers*) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing.

In all modes 16 registers, R0 to R15, are directly accessible. All registers except R15 are general purpose and may be used to hold data or address values. Register R15 holds the Program Counter (PC). When R15 is read, bits [1:0] are zero and bits [31:2] contain the PC. A seventeenth register (the CPSR - Current Program Status Register) is also accessible. It contains condition code flags and the current mode bits and may be thought of as an extension to the PC.

R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed. It may be treated as a general purpose register at all other times. R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). Many FIQ programs will not need to save any registers. User mode, IRQ mode, Supervisor mode, Abort mode and Undefined mode each have two banked registers mapped to R13 and R14. The two banked registers allow the these modes to each have a private stack pointer and link register. Supervisor, IRQ, Abort and Undefined mode programs which require more than these two banked registers are expected to save some or all of the caller's registers (R0 to R12) on their respective stacks. They are then free to use these registers which they will restore before returning to the caller. In addition there are also five SPSRs (Saved Program Status Registers) which are loaded with the CPSR when an exception occurs. There is one SPSR for each privileged mode.

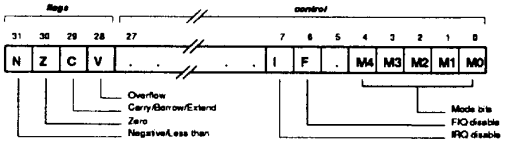

Figure 4 Program Status Register

The format of the Program Status Registers is shown in Figure 4. The N, Z, C and V bits are the *condition code flags*. The condition code flags in the CPSR may be changed as a result of arithmetic and logical operations in the processor and may be tested by all instructions to determine if the instruction is to be executed.

The I and F bits are the interrupt disable bits. The I bit disables IRQ interrupts when it is set and the F bit disables FIQ interrupts when it is set. The M0, M1, M2, M3 and M4 bits (M[4:0]) are the *mode bits*, and these indicate the mode in which the processor operates. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used.

The bottom 28 bits of a PSR (incorporating I, F and M[4:0]) are known collectively as the *control bits*. The control change when an exception arises and in addition can be manipulated by software when the processor is in a privileged mode. Unused bits in the PSRs are reserved.

3: Exceptions

Exceptions arise whenever there is a need for the normal flow of program execution to be broken, so that (for example) the processor can be diverted to handle an interrupt from a peripheral. The processor state just prior to handling the exception must be preserved so that the original program can be resumed when the exception routine has completed. Many exceptions may arise at the same time.

ARM6 handles exceptions by making use of the banked registers to save state. The old PC and CPSR contents are copied into the appropriate R14 and SPSR and the PC and mode bits in the CPSR bits are forced to a value which depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings of exceptions. In the case of a re-entrant interrupt handler, R14 and the SPSR should be saved onto a stack in main memory before re-enabling the interrupt; when transferring the SPSR register to and from a stack, it is important to transfer the whole 32 bit value, and not just the flag or control fields.When multiple exceptions arise at the same time, a fixed priority system determines the order in which they will be handled:

(1) Reset (highest priority)

(2) Data abort

(3) FIQ

(4) IRQ

(5) Prefetch abort

(6) Undefined Instruction, Software interrupt

Note that not all exceptions can occur at once. Undefined instruction and software interrupt are mutually exclusive since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (i.e. the F flag in the CPSR is clear), ARM6 will enter the data abort handler and then immediately proceed to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection; the time for this exception entry should be added to worst case FIQ latency calculations.

3.1: FIQ

The FIQ (Fast Interrupt reQuest) exception is externally generated by taking the Nfiq input LOW. This input can accept asynchronous transitions, and is delayed by one clock cycle for synchronisation before it can affect the processor execution flow. It is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications (thus minimising the overhead of context switching). The FIQ exception may be disabled by setting the F flag in the CPSR (but note that this is not possible from User mode). If the F flag is clear, ARM6 checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

When a FIQ is detected, ARM6 performs the following:

(1) Saves the address of the next instruction to be executed plus 4 in R14_fiq; saves CPSR in SPSR_fiq

(2) Forces M[4:0]=%10001 (FIQ mode) and sets the F and I bits in the CPSR

(3) Forces the PC to fetch the next instruction from address &1C

To return normally from FIQ, use SUBS PC, R14_fiq,#4 which will restore both the PC (from R14) and the CPSR (from SPSR_fiq) and resume execution of the interrupted code.

3.2: IRQ

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by a LOW level on the Nirq input. It has a lower priority than FIQ, and is masked out when a FIQ sequence is entered. Its effect may be masked out at any time by setting the I bit in the CPSR (but note that this is not possible from User mode). If the I flag is clear, ARM6 checks for a LOW level on the output of the IRQ synchroniser at the end of each instruction. When an IRQ is detected, ARM6 performs the following:

(1) Saves the address of the next instruction to be executed plus 4 in R14_irq; saves CPSR in SPSR_irq

(2) Forces M[4:0]=%10010 (IRQ mode) and sets the I bit in the CPSR

(3) Forces the PC to fetch from address &18

To return normally from IRQ, use SUBS PC,R14_irq,#4 which will restore both the PC and the CPSR and resume execution of the interrupted code.

3.3: Abort

The abort input comes from an external Memory Management system, and indicates that the current memory access cannot be completed. For instance, in a virtual memory system the data corresponding to the current address may have been moved out of memory onto a disc, and considerable processor activity may be required to recover the data before the access can be performed successfully. ARM6 checks for abort during memory access (N and S) cycles. When successfully aborted ARM6 will respond in one of two ways:

(1) If the abort occurred during an instruction prefetch (a *Prefetch Abort*), the prefetched instruction is marked as invalid but the abort exception does not occur immediately. If the instruction is not executed, for example as a result of a branch being taken while it is in the pipeline, no abort will occur. An abort will take place if the instruction reaches the head of the pipeline and is about to be executed.

(2) If the abort occurred during a data access (a *Data Abort*), the action depends on the instruction type.

(a) Single data transfer instructions (LDR, STR) are aborted as though the instruction had not executed if the processor is configured for Early Abort. When configured for Late Abort, these instructions are able to write back modified base registers and the Abort handler must be aware of this.

(b) The swap instruction is aborted as if it had not executed.

(c) Block data transfer instructions (LDM, STM) complete, and if write-back is set, the base is updated. If the instruction would normally have overwritten the base with data (i.e. LDM with the base in the transfer list), this overwriting is prevented. All register overwriting is prevented after the Abort is indicated, which means in particular that R15 (which is always last to be transferred) is preserved in an aborted LDM instruction.

When either a prefetch or data abort occurs, ARM6 performs the following:

(1) Saves the address of the aborted instruction plus 4 (for prefetch aborts) or 8 (for data aborts) in R14_abt; saves CPSR in SPSR_abt

(2) Forces M[4:0]=%10111 (Abort mode) and sets the I bit in the CPSR .

(3) Forces the PC to fetch the next instruction from either address &0C (prefetch abort) or address &10 (data abort)

To return after fixing the reason for the abort, use SUBS PC,R14_abt,#4 (for a prefetch abort) or SUBS PC,R14_abt,#8 (for a data abort). This will restore both the PC and the CPSR and retry the aborted instruction.

3.4: Software interrupt

The software interrupt instruction (SWI) is used for getting into Supervisor mode, usually to request a particular supervisor function. When a SWI is executed, ARM6 performs the following:

(1) Saves the address of the SWI instruction plus 4 in R14_svc; saves CPSR in SPSR_svc

(2) Forces M[4:0]=%10011 (Supervisor mode) and sets the I bit in the CPSR

(3) Forces the PC to fetch the next instruction from address &08

To return from a SWI, use MOVS PC,R14_svc. This will restore the PC and CPSR and return to the instruction following the SWI.

3.5: Undefined instruction trap

When the ARM6 comes across a coprocessor instruction which it cannot handle then the ARM6 will take the undefined instruction trap.

The trap may be used for software emulation of a coprocessor in a system which does not have the coprocessor hardware, or for general purpose instruction set extension by software emulation.

When ARM6 takes the undefined instruction trap it performs the following:

(1) Saves the address of the Undefined or coprocessor instruction plus 4 in R14_und; saves CPSR in SPSR_und.

(2) Forces M[4:0]=%11011 (Undefined mode) and sets the I bit in the CPSR

(3) Forces the PC to fetch from address &04

To return from this trap after emulating the failed instruction, use MOVS PC,R14_und. This will restore the CPSR and return to the instruction following the undefined instruction.

3.6: Reset

When the Nreset signal goes LOW, ARM6 abandons the currently executing instruction and then continues to fetch instructions from memory which it interprets as NOPs. When Nreset goes HIGH again, ARM6 does the following:

(1) Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and CPSR is not defined.

(2) Forces M[4:0]=%10011 (Supervisor mode) and sets the I and F bits in the CPSR.

(3) Forces the PC to fetch the next instruction from address &00

4: ARM610 cache

ARM610 contains a 4kByte mixed instruction and data cache; the IDC has 256 lines of 16 bytes (4 words), organised as 4 blocks of 64 lines (making it 64-way set associative), and uses the virtual addresses generated by the processor core. The IDC is always reloaded a line at a time (four words). It may be enabled or disabled via the ARM610 Control Register and is disabled on RESET. The operation of the cache is further controlled by two bits: **Cacheable** and **Updateable**, which are stored in the Memory Management Page Tables. For this reason, in order to use the IDC, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register.

4.1: The cacheable bit

The Cacheable bit determines whether data being read may be placed in the IDC and used for subsequent read operations. Typically main memory will be marked as Cacheable to improve system performance, and I/O space as Non-cacheable to stop the data being stored in ARM610's cache. [For example if the processor is polling a hardware flag in I/O space, it is important that the processor is forced to read data from the external peripheral, and not a copy of initial data held in the cache]. The Cacheable bit can be configured for both pages and sections.

4.2: The updateable bit

The Updateable bit determines whether the data in the cache should be updated during a write operation to maintain consistency with the external memory. [In certain cases automatic updating of cached data is not required: for instance, when using the MEMC1a memory manager, a read operation in the address space between 3400000H - 3FFFFFFH would access the ROMs, but a write operation in the same address space would change a MEMC register, and should not affect the cached ROM data]. The Updateable bit can only be configured by the Level One descriptor i.e. that is an entire section or all the pages for a single Level One descriptor share the same configuration.

5: ARM610 write buffer

The ARM610 Write Buffer is provided to improve system performance. It can buffer up to 8 words of data, and 2 independent addresses. It may be enabled or disabled via the W bit (bit 3) in the ARM610 Control Register and the buffer is disabled and flushed on RESET. The operation of the Write Buffer is further controlled by one bit, B, or Bufferable, which is stored in the Memory Management Page Tables. For this reason, in order to use the Write Buffer, the MMU must be enabled. The two functions may however be enabled simultaneously, with a single write to the Control Register. For a write to use the Write Buffer, both the W bit in the Control Register, and the B bit in the corresponding page table must be set.

5.1: Bufferable bit

This bit controls whether a write operation may or may not use the Write Buffer. [Typically main memory will be bufferable and I/O space unbufferable]. The Bufferable bit can be configured for both pages and sections.

Note that a single write requires one address slot and one data slot in the write buffer; a sequential write of n words requires one address slot and n data slots. The total of 8 data slots in the buffer may be used as required. So for instance there could be one non-sequential write and one sequential write of 7 words in the buffer, and the processor could continue as normal: a third write or an eighth word in the second write would stall the processor until the first write had completed.

6: ARM610 memory management unit

The MMU performs two primary functions: it translates virtual addresses into physical addresses, and it controls memory access permissions. The MMU hardware required to perform these functions consists of a Translation Look-aside Buffer (TLB), access control logic, and translation table walking logic.

The MMU supports memory accesses based on Sections or Pages. Sections are comprised of 1MB blocks of memory. Two different page sizes are supported: Small Pages consist of 4kB blocks of memory and Large Pages consist of 64kB blocks of memory. (Large Pages are supported to allow mapping of a large region of memory while using only a single entry in the TLB). Additional access control mechanisms are extended within Small Pages to 1kB Sub-Pages and within Large Pages to 16kB Sub-Pages.

The MMU also supports the concept of domains - areas of memory that can be defined to possess individual access rights. The Domain Access Control Register is used to specify access rights for up to 16 separate domains.

The TLB caches 32 translated entries. During most memory accesses, the TLB provides the translation information to the access control logic.

If the TLB contains a translated entry for the virtual address, the access control logic determines whether access is permitted. If access is permitted, the MMU outputs the appropriate physical address corresponding to the virtual address. If access is not permitted, the MMU signals the CPU to abort.

If the TLB misses (it does not contain a translated entry for the virtual address), the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. Once retrieved, the translation information is placed into the TLB, possibly overwriting an existing value. The entry to be overwritten is chosen cyclically.

When the MMU is turned off (as happens on RESET), the virtual address is output directly onto the physical address bus.

6.1: MMU faults and CPU aborts

The MMU generates four types of faults:

Alignment Fault.

Translation Fault.

Domain Fault.

Permission Fault.

In addition, an external abort may be raised on external data access.

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as the result of a memory access, the MMU will abort the access and signal the fault condition to the CPU. The MMU is also capable of retaining status and address information about the abort. The CPU recognises two types of abort: data aborts and prefetch aborts, and these are treated differently by the MMU.

If the MMU detects an access violation, it will do so before the external memory access takes place, and it will therefore inhibit the access. External aborts will not necessarily inhibit the external access, as described in the section on external aborts.

6.2: Fault checking sequence

The sequence by which the MMU checks for access faults is slightly different for Sections and Pages. Figure 5 illustrates the sequence for both types of accesses. The sections and figures that follow describe the conditions that generate each of the faults.

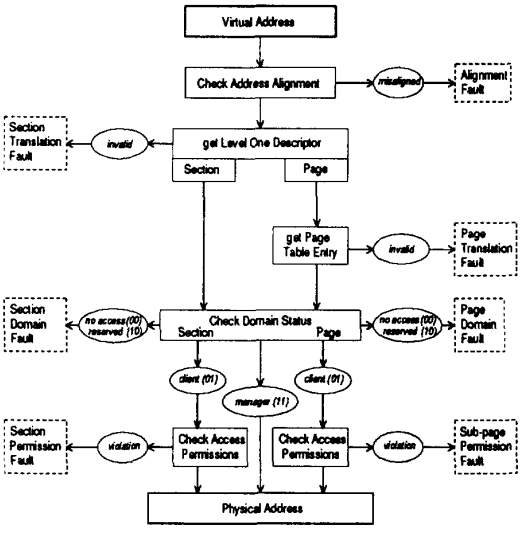

Figure 5: Sequence for Checking Faults

6.3: Alignment fault

If Alignment fault is enabled (bit 1 in Control Register set), the MMU will generate an alignment fault on any data word access the address of which is not word-aligned irrespective of whether the MMU is enabled or not; in other words, if either of virtual address bits [1:0] are not 0. Alignment fault will not be generated on any instruction fetch, nor on any byte access. Note that if the access generates an alignment fault, the access sequence will abort without reference to further permission checks.

6.4: Translation fault

There are two types of translation fault: section & page.

A Section Translation Fault is generated if the Level One descriptor is marked as invalid. This happens if bits[1:0] of the descriptor are both 0 or both 1.

A Page Translation Fault is generated if the Page Table Entry is marked as invalid. This happens if bits[1:0] of the entry are both 0 or both 1.

6.5: Domain fault

There are two types of domain fault: section & page. In both cases the Level One descriptor holds the 4-bit Domain field which selects one of the sixteen 2-bit domains in the Domain Access Control Register. The two bits of the specified domain are then checked for access permissions as detailed in Table 7. In the case of a section, the domain is checked once the Level One descriptor is returned, and in the case of a page, the domain is checked once the Page Table Entry is returned.

If the specified access is either No Access (00) or Reserved (10) then either a Section Domain Fault or Page Domain Fault occurs.

6.6: Permission fault

There are two types of permission fault: section & sub-page. Permission fault is checked at the same time as Domain fault. If the 2-bit domain field returns client (01), then the permission access check is invoked as follows:

section:

If the Level One descriptor defines a section-mapped access, then the ap bits of the descriptor define whether or not the access is allowed.Their interpretation is dependent upon the setting of the S bit (Control Register bit 8). If the access is not allowed, then a Section Permission fault is generated.

sub-page:

If the Level One descriptor defines a page-mapped access, then the Level Two descriptor specifies four access permission fields (ap3..ap0) each corresponding to one quarter of the page. Hence for small pages, ap3 is selected by the top 1kB of the page, and ap0 is selected by the bottom 1kB of the page; for large pages, ap3 is selected by the top 6kB of the page, and ap0 is selected by the bottom 16kB of the page. The selected ap bits are then interpreted in exactly the same way as for a section (see table 1 below), the only difference being that the fault generated is a sub-page permission fault

ap	S	Supervisor	User	Notes
00	0	No Access	No Access	Any access generates a permission fault
00	1	Read Only	No Access	Supervisor read only permitted
01	x	Read/Write	No Access	Supervisor read or write only permitted
10	x	Read/Write	Read Only	Writes in User mode cause permission fault
11	x	Read/Write	Read/ Write	All access types permitted in both modes

Table 1: Interpreting Access Permission (ap) Bits

6.7: External Aborts

In addition to the MMU-generated aborts, ARM610 has an external abort pin which may be used to flag an error on an external memory access.

Conclusion

ARM achieves this small size, low power consumption and high code density by using RISC and CISC design techniques as well as careful attention to VLSI implementation. The flexibility of a CPU macrocell that can be used in ASIC applications as well as full custom designs is an important advantage. These features are essential for the emerging market of portable consumer products and the demands of high volume high performance embedded control.

[1] " VLSI RISC Architecture and Organisation"

Stephen B. Furber. Marcel Deckker, Inc.

ISBN 0-8247-8151-1