The ARM9 Family - High Performance Microprocessors for Embedded Applications

Simon Segars, Manager CPU Development, ARM Ltd.

Abstract

Portable applications such as mobile phones, pagers, and PDAs are continually growing in sophistication. This places an increasing burden on the embedded microprocessor to provide high performance while retaining low power consumption and small die size.

The ARM7TDMI microprocessor has been highly successful in these application areas. However, as products grow in complexity more processing power is required while the expectation on battery life also increases. This has lead to the introduction of the ARM9 family, a range of high performance low power embedded microprocessors targeted at next generation embedded applications.

This paper focuses on the implementation of 2 members of the ARM9 family, the ARM9TDMI integer core and the ARM940T cached processor. These offer performance in excess of 150 MIPS while retaining low power consumption. The evolution from the ARM7 to the ARM9 microarchitecture is described and the trade offs between low power consumption and high performance discussed.

Introduction

ARM designs high performance, low power microprocessors targeted at embedded applications. To date most of the ARM design wins have been with the ARM7TDMI [1,2] processor. This product incorporates the Thumb instruction set [3], providing industry leading code density and typically achieves around 60MHz and only 1.5 mW/MHz power consumption on a 0.35µm process. Coupled with a small die size and integral debug features, this product is ideal for many medium-performance embedded applications.

ARM7TDMI has been successful in many portable applications. Examples include GSM mobile phones such as the Panasonic G650 [4]. ARM7 based cores have also been integrated with cache memories and peripherals in ASSPs such as the ARM7100 [5] as used in the PSION 5 PDA [6]. The ARM7 family owes its success to the combination of low power, low cost and high performance.

However, as applications become more complex and integrate more and more functionality, the processor is required to provide more and more performance. A classic example of such an application is the so-called 'Smart Phone'. This is a cellular phone and PDA rolled into one. Initial smart phones have used multiple processors in order to meet the performance needs - one to run the PDA, another to run the cellular protocol stack and a DSP to process the data traffic.

Applications such as this epitomize Moore's law and have lead ARM to develop the ARM9 family of microprocessors [7]. These devices build on the architecture of the ARM7 family and provide higher levels of performance. ARM9 processors are specifically targeted to meet the needs of the next generation of highly integrated portable applications while at the same time keeping power consumption and die size to a minimum.

While the ARM9 family rises to meet this challenge, the ARM7 family will live on servicing the needs of low-end applications.

The ARM9TDMI embedded core

The first member of the ARM9 family is the ARM9TDMI integer core. The goal of this product was to produce a high performance Thumb compatible processor, to providing a performance upgrade path from the ARM7TDMI. The processor was specified to be used either stand alone, or within a cached processor such as the ARM940T.

Higher performance has been achieved by increasing the depth of the pipeline from 3 stages as in the ARM7TDMI to 5 stages. This allows the device to be clocked at a higher rate than the ARM7TDMI. Forwarding paths have also been introduced to the pipeline in order to reduce the number of interlock cases and hence reduce the average number of clocks per instruction, CPI.

Load and store operations account for around 25% of all instructions in the ARM instruction flow (a more detailed breakdown of ARM instruction distribution is shown in Table 1). In ARM7TDMI a basic load takes 3 cycles and a store takes 2 cycles and these contribute significantly to the average overall CPI. It is therefore important to optimise for these instructions in high performance processors. This has been achieved in the ARM9TDMI processor core by adopting a Harvard memory architecture (ARM7TDMI used a Von Neumman architecture for simplicity) thereby allowing instruction fetches to occur in parallel with data accesses.

The ARM9TDMI pipeline consists of the stages fetch, decode, execute, memory access and write-back. The main operations performed in each stage are described in Figure 1 which compares the pipelines of the ARM7TDMI and ARM9TDMI.

The extra stages allow the work performed in the somewhat congested execute state of the ARM7TDMI to be spread more evenly, thereby permitting a higher maximum operating frequency. ARM7TDMI performs a Thumb to ARM instruction format conversion during the first phase of the decode cycle. In the 5 stage pipeline of ARM9TDMI, ARM and Thumb instructions are decoded in parallel. ARM9TDMI performs register address decode in the first half of the decode cycle and register reads in the second half. This means that there is no spare phase in which to perform a Thumb to ARM instruction conversion as there was in the ARM7TDMI. This leads to two parallel decode units, one which is only active when the processor is in ARM state and the other active only when the processor is in Thumb state, in order to save power.

The datapath of the ARM9TDMI is shown in Figure 2. The register bank has 3 read ports and two write ports. The A and B read ports feed the execution units in the datapath. The C port is used exclusively for reading store data. Store data is read during the execute stage of the pipeline. This reduces the number of forwarding paths and also removes the need for holding latches which would be required if the data was read during decode along with the other registers.

The shifter and ALU perform the same functions as those found in ARM7TDMI. The main difference in the ALU is that the arithmetic and logic units are separated so that during an instruction only the required functional unit is activated. It has been found that the ALU contributes significantly to the power consumption of the ARM7TDMI. In that device the simple nature of the ALU means that both an arithmetic and a logic result are calculated each cycle and the required result is then selected. This is inefficient from a power consumption perspective and so the two units have been partitioned in the ARM9TDMI design.

The forwarding paths in the ARM9TDMI allow back to back data processing instructions to execute in the pipeline without stall cycles. Load data, which becomes available at the end of the memory cycle, is also forwarded into the pipeline. If the data from the load is required in the very next cycle then there is a one cycle interlock, since the data is not returned until the end of the memory cycle, and the load instruction occupies 2 execute cycles in the datapath. However, if the data is not required until the next but one instruction then the data is forwarded, there is no interlock and the instruction has only occupied one execute cycle in the datapath. These two cases are depicted in Figure3 (a) and (b).

There is a third case where a load data specifies a rotation or sign extension as it is fetched and here the forwarding paths cannot be used. The loaded data must be passed through the Byte Rotator block and written back to the register bank before being used by subsequent instructions. Therefore, instructions such as these can cause up to two cycles of interlock depending on when the data is required.

The ARM9TDMI microarchitecture described above results in an average CPI of 1.5. A breakdown of the number of cycles for each instruction class is shown in Table 1 along with that of ARM7TDMI for comparison. The new microarchitecture results in a 21% increase in

ARM7TDMI Pipeline Operation

Fetch	Decode		Execute	
Instruction Fetch	Convert Thumb to ARM	Main Decode Register Address Decode	Register Read	Shifter ALU Writeback

ARM9TDMI Pipeline Operation

Fetch	Decode	Execute		Memory	Writeback
Instruction Fetch	ARM Decode Reg. Address Decode Register Read Thumb Decode Reg. Address Decode Register Read	Shifter	ALU	Memory Data access	ALU Result and / or Load data Writeback

Figure 1 : ARM7TDMI and ARM9TDMI Pipelines

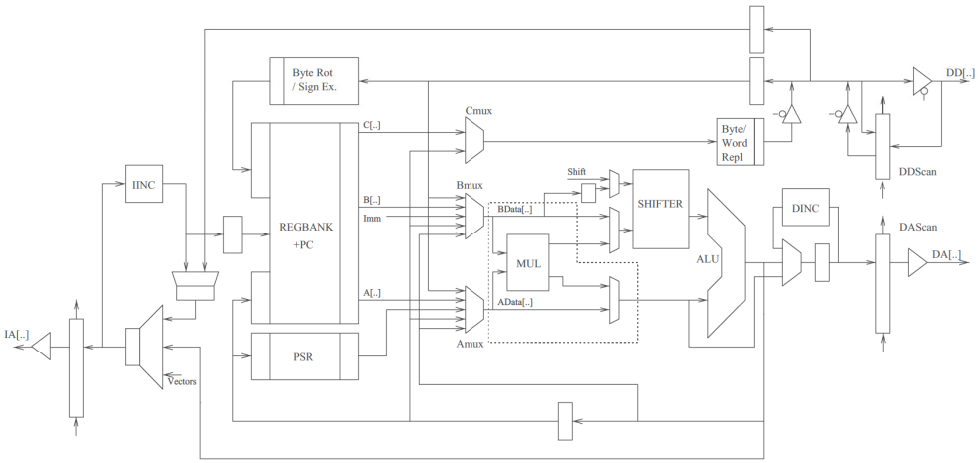
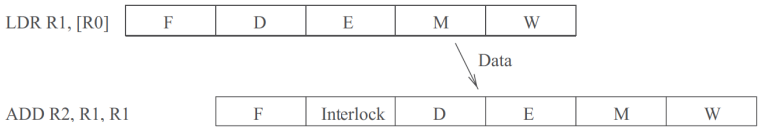
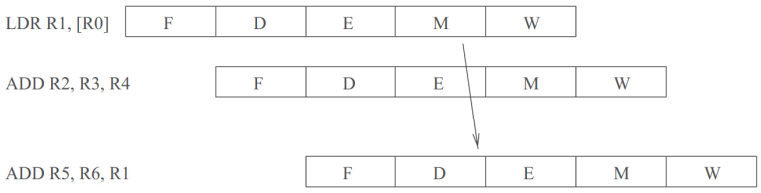


Figure 2 : The ARM9TDMI Datapath



(a) Single Cycle Interlock



(b) No Interlock

Figure 3 : ARM9TDMI Load Behaviour

instruction throughput relative to ARM7TDMI and 1.1 MIPS/MHz, compared to 0.9 MIPS/MHz.

The re-pipelining allows the clock rate to be increased significantly compared to ARM7TDMI. On the same process, ARM9TDMI may be clocked at twice the rate of ARM7TDMI. The increase in complexity required to achieve the increase in performance requires around 50% more transistors and the area has increased by almost 90% This area increase is accounted for by an increase in the number of routing channels in the datapath (to permit the additional forwarding paths) and a relative increase in the standard cell control logic to custom datapath ratio. A comparative summary between the two processor cores is shown in Table 2.

The ARM940T cached processor

The ARM9TDMI processor core has been integrated with caches, a write buffer and a protection unit in the ARM940T processor. This system has many advantages for the system designer. Firstly, it allows the processor to operate at its maximum frequency since memory accesses are to the local, high performance cache. Secondly, since main memory is accessed infrequently, system power is reduced. Also, the main memory system may now be used for other tasks, such as DMA, while the

Instruction	% Taken	% Skipped	ARM7TDMI	ARM9TDMI
Data processing	49	4	1	1
Data processing with PC	3	0	3	3
Branch/Branch with link	11	4	3	3
Load register	14	1	3	1-2
Store register	8	1	2	1
Load multiple registers	1	0	7	5
Store multiple registers	2	0	7	6
CPI			1.9	1.5

Table 1 : ARM9TDMI vs ARM7TDMI CPI Analysis

processor is executing from its caches. A block diagram of the ARM940T design is shown below in Figure 4.

The Harvard caches in the ARM940T are both 4KB in size in the first implementation. The caches are constructed in a modular manner using 1KB cache blocks. Through the use of these blocks the size of the cache can easily be varied with minimal impact on the rest of the design. The cache blocks are built using a CAM-RAM structure comprising 64 lines each with 4 words of data. Each cache block has 64 way associativity and the CAMs are designed to compare a maximum of 27 address bits. A cache read involves 3 basic steps.

- Firstly, the 32 bit address from the processor is decoded to determine which of the 4 segments the addressed data might be in. In the 4KB ARM940T design, bits 5:4 of the address are used for the segment decode.
- Secondly, the upper 26 bits of the address are then passed into the CAM where they are compared with the CAM contents.
- Finally, if there is a CAM match, then a data access in the cache RAM occurs. Each RAM line is 4 words long and bits 3:2 of the address are used to select the desired word. If the cache lookup fails then the processor is stalled by the cache control logic and an external access occurs to fetch the required data.

Although the high associativity helps little with cache hit rates, the design has a number of advantages. These include low power, short cycle time and simple modularity allowing the ARM940T to be extended to have larger caches by utilising more cache segments. For example, if the caches were 8KB in size, then an additional bit must be used for the segment decode and one less bit passed into the CAM for address look-up. In this case, bits 6:4 would be used for segment decode and bits 31:7 of the address would be passed into the CAMs. The unused column of the CAM would simply be tied off. In fact, the CAMs are designed to cope with cache sizes down to 2KB and in the initial design the least significant CAM column is tied off.

	ARM7TDMI	ARM9TDMI
Area (mm^2, 0.35µm)	2.2	4.15
Transistor count	74k	112k
Pipeline stages	3	5
CPI	1.9	1.5
MIPS/MHz	0.9	1.1
Typical Max Clock rate (0.35µm)	60	120
Power (mW/MHz @ 3.0V)	1.5	1.8

Table 2 : ARM9TDMI vs ARM7TDMI Comparison Sumary

In order to increase performance, the ARM940T contains a write buffer. Without this, then any store operation to main memory would have to stall the processor until the memory bus was free. The write buffer allows the processor to be isolated from the traffic on the system memory bus and so stall cycles are minimised. The write buffer allows storage for up to 8 words of data and 4 address values.

As a further power and bus activity saving feature, the ARM940T data cache may be operated in a

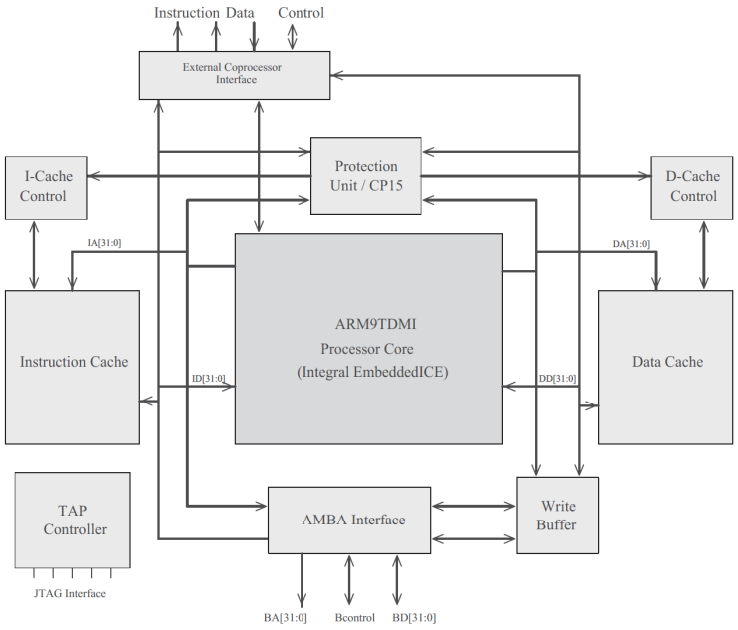


Figure 4 : The ARM940T Processor

write-back mode. In this mode of operation, whenever a store occurs which hits in the cache, the cache is updated but the write is not passed to the external memory system. At this time the cache and main memory have lost coherency and the cache line containing the incoherent data is said to be 'dirty'. Subsequently, if as a result of a later cache miss the dirty line is selected to be overwritten with new data, the dirty data must be written back to main memory. When this occurs, the processor is stalled while the dirty data is copied from the cache into the write buffer. The linefill of the new data is then performed and written into the cache. At that point processor execution is resumed. The data in the write buffer is written back to memory when the system bus is free and before any further read operations, ensuring memory coherency.

The benefit of a write-back cache is that many store operations may occur to the cache before they are copied to the main memory. Therefore the total number of main memory accesses and hence system power, is reduced. This is especially useful for data regions where program variables are to be stored. There are cases when main memory and the cache must be kept coherent at all times, for a video frame buffer. Consequently ARM940T also supports a write-through mode of operation where all cache updates are written through to memory as they happen.

ARM940T is targeted at a class of embedded applications referred to as closed applications. Closed applications are where all the software the processor will execute is present in the system when it is shipped by the OEM. Since the software can be considered reliable and safe, the memory protection provided by the processor may be minimised. Consequently, ARM940T does not support virtual memory and does not contain an MMU or TLB. Instead, a simple Protection Unit, PU, is provided. The PU is programmed via accesses to the system coprocessor, CP15.

The protection unit allows the system designer to partition memory into 16 regions, 8 on the instruction side and another 8 unique regions on the data side. Each region is specified by a base address pointer and a size field. The size can be anything, in powers of 2, from 4GB to 4KB. The address of the start of the region must be multiple of the region's size. Each region has a number of properties associated with it specifying how the cache and write buffer behave in that region, eg. cacheable, non-cacheable, write-through or write-back, and also what type of access, eg. supervisor only, can occur within the region.

The regions are labeled 0-7 and may be programmed such that they overlap. If a memory access occurs which corresponds to 2 or more regions then the attributes for the highest numbered region are used (ie. region 7 has the highest priority and region 0 the lowest).

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.