

ARM® and Low-Power Applications

Ernest Karjala, Technical Training Manager, Arrow Electronics
Teodor Neagoe, Sr. Field Applications Engineer, Arrow Electronics

Abstract

ARM®'s success to date has been largely due to its remarkable performance/power (MIPS/Watt) rating, and this will likely continue to be its most critical benchmark for future applications. This paper considers how attention to interdependent detail in cache controller design, conditionally executable instructions, intelligent voltage management, and interrupt control hardware contribute to the ARM architecture's high performance and low power operation.

Level 1 Cache Controller Design

As CPU clock rates go higher and higher, Level 1 caches, which are implemented as part of the processor core, are a requirement when high performance, measured in terms of clocks per instruction (CPI) must be maintained, even when frequent access must be made to SDRAM or Flash memory external to the core.

A cache, as illustrated in Figure 1 below, is a subsystem that includes two kinds of RAM and some logic that forms the cache controller. The cache is interposed between the CPU and slow system memory. The purpose of a cache is to make the slow memory system look like a fast memory system.

The data RAM size is measured in bytes, but internally, it is organized as lines of memory. A line represents the minimum number of sequential words (or bytes) that are copied into the cache when it is updated. Many ARM implementations use an eight-word (32-byte) line.

Each line in the Data RAM has an associated location in the TAG RAM, indexed to the same depth in the array, that provides the upper order bits of the address representing where in memory that line is from. Think of it as an address that can be changed on the fly. To satisfy a request for memory data, the cache controller compares the TAG in the TAG RAM at the same "address" as the line in memory. If there is a match, there is a hit.

The cache controller can usually satisfy requests for memory data from lines recently written into the cache. The cache provides the fastest (single bus cycle) memory access. For a miss, the controller holds off the CPU, reads slow memory, supplies the data requested to the CPU, and updates the line in the cache and the TAG. The line fill requires multiple bus cycles to slow memory.

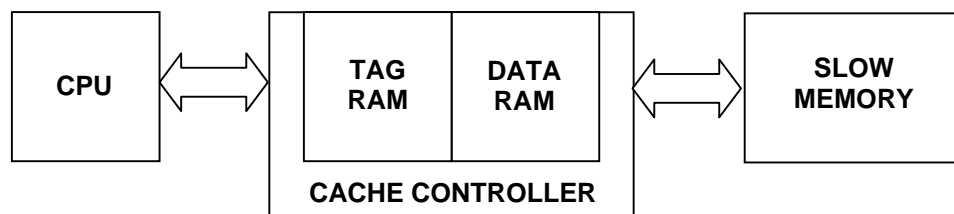


Figure 1: Generic Cache

Assume, for a moment, that the slow memory shown above requires ten bus cycles to access, and the data RAM in the cache can be accessed in one bus cycle. Assume, also, that 95% of the memory data requests can be satisfied by the cache. This is called a 95% hit rate. If all cache cycles were read cycles (which they are not), in 100 cycles, the cache would provide 95 single-cycle and five 10-cycle memory accesses, resulting in $95 + 50 = 145$ bus cycles. This is a 1.45 cycle memory system for reads. Thus, a memory system that would normally require ten bus cycles to access now looks to the processor like a 1.45 bus cycle system!

Using the same kind of arithmetic, a hit rate of 80% would provide a 2.8 bus cycle system, a 98% hit rate would provide a 1.18 bus cycle system, etc.

There are two fundamental ways to maximize cache hit rates:

1. Increase the cache size. The bigger the cache, the higher the hit rate.
2. Change the cache organization:
 - a. Direct Mapped: Easiest to implement, but the lowest hit rate for a given cache size.
 - b. Set Associative: Better hit rate for a given size cache. The more ways the better.
 - c. Fully Associative: Best for a given size cache, but expensive to implement.

Many implementations of the ARM Microarchitecture offer 32- and 64-way set associative caches. This has two benefits. First, 32- and 64-way associativity maximizes the hit rate of a small cache to enhance performance. At the same time, it reduces thrashing or cache contention to maximize cache performance. Second, the small cache size minimizes the amount of memory required for a high hit rate cache, reducing power consumption. The large number of ways by definition divides the cache memory into smaller blocks, which can be exploited to reduce power consumption even more.

The ARM 1176JZF-S™ microarchitecture provides a more subtle, but specific, example¹ of a technique for reducing cache power consumption. The microarchitecture takes advantage of the sequential nature of many cache operations to reduce overall power consumption. The idea is to reduce the number of full cache reads, especially for the instruction cache, as follows: On a cache read that is sequential to the previous cache read, only the data RAM set that was previously read is accessed, if the read is within the same cache line. This avoids having to access the TAG RAM at all during this sequential operation. Additionally, only the addressed words within a cache line are read at any time to reduce power consumption even further.

There is also the ARM11™ microarchitecture's *hit-under-miss* feature that allows up to three successive cache data misses to occur without an instruction pipeline stall, assuming no data dependencies occur after the first miss². This reduces the number of pipeline stalls and the subsequent, associated cache reloads.

In general then, a good L1 cache design reduces power consumption by reducing the number of transistors required to implement the cache memory and the number of clocks a CPU requires to complete an operation. It will also minimize the number of transistors or gates that must be enabled, activated, or caused to do state transitions for cache operations.

Intelligent Voltage Management

For current integrated circuits, the reduced feature size, the increased number of transistors, and increased transistor speed all collaborate together to present challenges in terms of power density and a growing gap between battery capacities and aggregate energy needs³. The three most commonly identified sources of power consumption in integrated circuits are:

1. Dynamic: Power required to charge and discharge capacitive loads.
2. Short circuit: Power dissipated in momentary short circuits during complementary PMOS/NMOS switch state transitions.
3. Static: Power lost due to leakage.

There is also an important di/dt contribution, including V_{DD} / GND bounce³.

Equation 1 (from Trevor Mudge⁴, the University of Michigan) allows us to estimate the contribution of each power consumption source to an IC's overall power dissipation:

$$P = ACV^2f + \tau AVI_{\text{short}}f + VI_{\text{leak}} \quad (1)$$

Where f is the frequency of the system's operation, A is the activity of the gates in the system to account for gates that do not switch every clock, I_{short} is the short circuit current and τ is its duration, I_{leak} is the static leakage, and V is the supply voltage.

The first term in Equation 1 is for dynamic power, the second term for short circuit power, and the third term for static power. All three types of power consumption are proportional to supply voltage and dynamic and short circuit power are proportional to frequency. Even worse, dynamic power is proportional to the *square* of the supply voltage. Clearly, frequency of operation and reduction or management of supply voltage are important parameters to manage. Dynamic power consumption has been dominant until recently for power aware design. With the advent of 90 nm technology, however, leakage power now roughly equals dynamic power in terms of chip power. The leakage power/dynamic power ratio is over one for 65 nm and 25 nm geometries. Thus, optimization of leakage power is significant in IC energy management.

Historically, a variety of techniques have been developed and applied to date to manage power. These include the use of clock gating, frequency scaling, activation only of logic that is needed for a specific operation, power gating using of V_{DD} - or V_{SS} -path isolation cells, and the selection of custom process options. More recently, such techniques have included voltage scaling and performance optimization using multicell libraries⁵.

An example of practical power aware design is the recent development of a general-purpose power optimization methodology for synthesis-based digital designs as applied to the realization of an ARM1136 JF-S microprocessor⁵. It demonstrates a variety of power-aware design techniques. To start with, there is the use of two voltage domains (1.0-V and 0.8-V), allocating 97.5% and 8.5% of the standard cells in the 1.0-V and 0.8-V domains, respectively, as the high-type. It turns out that the faster Low V_T cells have about six times the leakage current of the high- V_T cells in this geometry. Low V_T cells were used in critical timing paths. There was also application of pin swapping to match high transition inputs to low capacitance pins, buffering inputs to reduce PMOS/NMOS transition short circuit currents, logic restructuring to reduce switching nodes and logic required for given functions, careful placement of voltage level shifting (VLS) cells to achieve optimal routing, and clock gating. The bottom line is an approximate 40% reduction in power dissipation and a 46% reduction in leakage power.

In addition to this kind of approach, and even beyond instruction set architecture and microarchitecture, ARM also supports a systems approach to power aware design. This could form the basis for techniques that address a chip's static (leakage) power consumption⁶. It is called the Intelligent Energy Manager (IEM)⁷. IEM uses predictive algorithms embedded in an operating system kernel to monitor all processes to reduce power consumption. It provides continuous predictive monitoring of the CPU workload and attempts to run the clock frequency at the lowest available value while still completing the work prior to its deadline.

The basic idea is simple: completing a task before its deadline and then idling is less energy efficient than running the task more slowly to begin with, and meeting its deadline exactly⁸. To implement this, frequency and voltage scaling must be possible. Maximum clock rate is proportional to supply voltage and dynamic power is a function of V_{DD}^2 . Thus, adjusting the operating voltage to the minimum required for the slowest successful clock rate produces the lowest power operation. A major advantage is that implementation of this kind of dynamic power management requires no modification of user programs

IEM has two components, a software component, IEM, and a hardware component called the Intelligent Energy Controller (IEC). The IEC is an Advanced Microcontroller Bus Architecture (AMBA) bus compliant peripheral which provides a number of essential support functions to the IEM software. It also measures the work done in the system to ensure that the software deadlines are not going to be missed. Additionally, the IEC supports a "maximum performance" hardware request feature.

The IEC drives other hardware. It uses the SoC specific on-chip clock management unit to control frequency changes and then depends on the Adaptive Power Control (APC) macrocell developed by National Semiconductor Corporation to control voltage. The IEC and APC adaptively set the minimum required power supply voltage provided by external compliant power management chips, such as the National Semiconductor PowerWise compliant power management chips using the PowerWise Interface (PWI), an open-standard interface dedicated to power management. The IEM, APC, and PowerWise components can be used to reduce core CPU power consumption by 75%.

The APC product is provided as soft IP (10,000 gates) with RTL, synthesis scripts, test benches, implementation and programming documentation. The APC product is also configurable to support up to eight performance levels and includes a PWI compliant master. The APC product supports deployment of both table-based Dynamic Voltage Scaling and PowerWise Adaptive Voltage Scaling (AVS). APC technology is available from ARM. For further information, please visit www.powerwise.national.com, www.arm.com, or www.pwistandard.org.

Instruction Set

ARM is a RISC (Reduced Instruction Set Computer) machine also called a Load/Store type architecture. All instructions are 32 bits long and most of them execute in one clock cycle. We are not going to analyze in detail the ARM instruction set, just flag the differences that contribute to the increase of code density and faster execution.

First, all ARM instructions can be made to execute conditionally by post-fixing them with the appropriate condition code; this can increase code density and increase performance by reducing the number of forward branches.

CMP	r0, r1	;r0-r1, compare r0 with r1 and set flags
ADDGT	r2, r2, #1	;if > r2=r2+1 flags remain unchanged
ADDLE	r3, r3, #1	;if <= r3=r3+1 flags remain unchanged

The following examples are relevant to this concept:

C source code	ARM instructions	
	unconditional	conditional
<pre>if (r0 == 0) { r1 = r1 + 1; } else { r2 = r2 + 1; }</pre>	<pre>CMP r0, #0 BNE else ADD r1, r1, #1 B end else ADD r2, r2, #1 end ...</pre>	<pre>CMP r0, #0 ADDEQ r1, r1, #1 ADDNE r2, r2, #1 ...</pre>
	<ul style="list-style-type: none"> ▪ 5 instructions ▪ 5 words ▪ 5 or 6 cycles 	<ul style="list-style-type: none"> ▪ 3 instructions ▪ 3 words ▪ 3 cycles

C source code	ARM instructions	
	unconditional	conditional
<pre>if (r0 != 5) { r1 = r1+r0-r2; }</pre>	<pre>CMP r0, #5 BEQ BYPASS ADD r1, r1, r0 SUB r1, r1, r2 BYPASS</pre>	<pre>CMP r0, #5 ADDNE r1, r1, r0 SUBNE r1, r1, r2 ...</pre>
	<ul style="list-style-type: none"> ▪ 4 instructions ▪ 5 words ▪ 5 or 6 cycles 	<ul style="list-style-type: none"> ▪ 3 instructions ▪ 3 words ▪ 3 cycles

Another specific of the ARM architecture instruction set is that data processing instructions do not affect the condition flags but this can be achieved by postfixing the instruction (and any condition code with an "S"):

```
loop
    ADD    r2, r2, r3          ;r2=r3+r3
    SUBS   r1, r1, #0x01      ;decrement r1 and set flags
    BNE    loop
```

As with conditional execution, this feature, when used judiciously, leads to increased code density, increased performance, and consequently reduced power consumption.

Interrupt System

An exception handler lies at the heart of embedded systems. It is responsible for handling errors, interrupts, and other events generated by the internal core or external peripherals. Efficient handlers can dramatically improve system performance and reduce overall power consumption. A comparison of the traditional way of handling exceptions and, more precisely, interrupts in classical systems as opposed to the way the ARM systems do it, should illustrate how and why this is so.

In many microprocessor architectures, the interrupt system implements a priority scheme that requires signal handshaking. In such a scheme, it is the responsibility of the requesting peripheral to provide an interrupt vector number or address to the interrupt system. Figure 2 represents such a system.

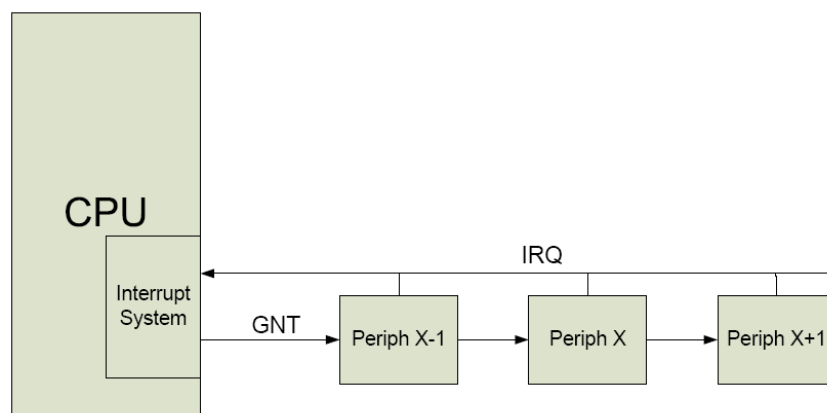


Figure 2: Classic Interrupt System Signal Handshaking

Suppose peripheral X needs to exchange data with the processor:

1. It pulls down the IRQ (Interrupt ReQuest) signal.
2. At the first interruptible point, the processor saves the context, Program Status (PS), and Program Counter (PC) in the stack.
3. Then it sends back an acknowledge, commonly called GNT (Interrupt Grant).
4. This signal is daisy chained to all peripherals on the same interrupt level and the first one in the chain that needs the bus sends its interrupt vector.
5. The processor reads the interrupt vector that represents an address.
6. From corresponding address, it reads the address where the associated Interrupt Service Routine is located. This address is loaded in the PC (Program Counter).
7. It fetches the first instruction from the ISR.

These steps, depending on the implementation, represent somewhere between eight and twelve clock cycles of delay.

There are two fundamental differences in ARM systems. First, there is no signal handshaking and second, the entries in the vector table are not the addresses of Interrupt Service Routines (ISRs), but instructions that branch to the ISRs. The vector table has seven entries: Reset, Undefined Instruction, Software Interrupt, Prefetch Abort, Data Abort, Interrupt ReQuest (IRQ) and Fast Interrupt reQuest (FIQ), as shown in Figure 3.

0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Figure 3: ARM Vector table

FIQ was defined at the top of the vector table on purpose, in order for the Interrupt Service Routine to be entirely located at this address.

When an exception occurs, the ARM:

1. Saves the context, copies the CPSR into SPSR_<mode>.
2. Sets the appropriate CPSR bits (ARM state if in Thumb, Mode field bits if appropriate, Interrupt diable bits is necessary).
3. Stores the return address in LR_<mode>.
4. Sets the PC to vector address.

All this process takes between four to six clock cycles.

It should be obvious that ARM's interrupt system has a much faster response time than the typical architecture, which in turn translates into reduced power consumption.

References

- 1** ARM1176JZF-S™ Technical Reference Manual, Revision: r0p6, © 2004-2007 ARM Limited. All rights reserved. ARM DDI 0301F, p7-3
- 2** David Cormie, "The ARM11™ Microarchitecture," ARM Ltd, April 2002
http://www.arm.com/support/White_Papers
- 3** Professor David Brooks, lecture 1 notes from course CS246: Advanced Computer Architecture, Spring 2007, Harvard University, <http://www.eecs.harvard.edu/~dbrooks/cs246/#Course%20Readings>
- 4** T. Mudge, "Power: A First-Class Architectural Design Constraint," IEEE Computer, 2001
- 5** Aurangzeb Khan, Senior Member, IEEE, Philip Watson, George Kuo, Due Le, Trung Nguyen, Steven Yang, Peter Bennett, Pokai Huang, Jaspal Gill, Chris Hawkins, John Goodenough, Demin Wang, Irfan Ahmed, Peter Tran, Helder Mak, Oanh Kim, Frank Martin, Yimu Fan, David Ge, Joseph Kung, Member, IEEE, and Vincent Shek, "A 90-nm Power Optimization Methodology with Application to the ARM 1136JF-S Microprocessor", IEEE Journal of Solid-State Circuits, Vol. 41, No. 8, August 2006
- 6** Krisztián Flautner, David Flynn (ARM Limited), and Mark Rives (National Semiconductor Corporation), DesignCon 2003 System-on-Chip and ASIC Design Conference, "A Combined Hardware-Software Approach for Low-Power SoCs: Applying Adaptive Voltage Scaling and Intelligent Energy Management Software"
- 7** IQ On-linePartner News. 15 June 2004 "National Semiconductor's APC Now Licensing From ARM," <http://www.arm.com/iqonline/news/partnernews/6569.html>
- 8** Krisztián Flautner, Steven K. Reinhardt, Trevor N. Mudge: "Automatic Performance Setting for Dynamic Voltage Scaling May 30, 2001" MOBICOM 2001 Proceedings and University of Michigan, 1301 Beal Ave. Ann Arbor, MI 48109-2122
- 9** ARM System-on-Chip Architecture (2nd Edition) by Steve Furber, Published by Addison-Wesley Professional (August 25, 2000)
- 10** ARM Architecture Reference Manual (2nd Edition) by David Seal, Published by Addison-Wesley Professional (December 27, 2000)
- 11** ARM System Developer's Guide: Designing and Optimizing System Software by Andrew Sloss, Dominic Symes and Chris Wright, Published by Morgan Kaufmann (March 25, 2004)
- 12** 034v14_InstOverview.ppt and 063v11-Exception_Handling.ppt, ARM Training material