ARM V8 FAMILY PROCESSORS

Silvia González Rodríguez
Miguel Enrique Játiva Jiménez
Miguel Sánchez de la Rosa

Computer Architecture

December 2019

Contents

List of Figures

List of Tables

1 Historic Context

ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Arm Holdings develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures including systems on chips (SoC) and systems on modules (SoM) that incorporate memory, interfaces, radios, etc. It also designs cores that implement this instruction set and licenses these designs to a number of companies that incorporate those core desings into their own products.[1]

The British computer manufacturer Acorn Computers first developed the Acorn RISC Machine architecture (ARM) in the 1980s to use in its personal computers. Its first ARM-based products were coprocessor modules for the BBC Micro series of computers. After the successful BBC Micro computer, Acorn Computers considered how to move on from the relatively simple MOS Technology 6502 processor to address business markets like the one that was soon dominated by the IBM PC, launched in 1981. The Acorn Business Computer (ABC) plan required that a number of second processors be made to work with the BBC Micro platform, but processors such as the Motorola 68000 and National Semiconductor 32016 were considered unsuitable, and the 6502 was not powerful enough for a graphics-based user interface.[1]

The official Acorn RISC Machine project started in October 1983. They chose VLSI Technology as the silicon partner, as they were a source of ROMs and custom chips for Acorn. Wilson and Furber led the design. They implemented it with a similar efficiency ethos as the 6502. A key design goal was achieving low-latency input/output (interrupt) handling like the 6502. The 6502's memory access architecture had let developers produce fast machines without costly direct memory access (DMA) hardware. The first samples of ARM silicon worked properly when first received and tested on 26 April 1985.[1]

The ARM2 featured a 32-bit data bus, 26-bit address space and 27 32-bit registers. Eight bits from the program counter register were available for other purposes; the top six bits (available because of the 26-bit address space) served as status flags, and the bottom two bits (available because the program counter was always word-aligned) were used for setting modes. The address bus was extended to 32 bits in the ARM6, but program code still had to lie within the first 64 MB of memory in 26-bit compatibility mode, due to the reserved bits for the status flags. The ARM2 had a transistor count of just 30,000, compared to Motorola's six-year-older 68000 model with around 40,000. Much of this simplicity came from the lack of microcode (which represents about one-quarter to one-third of the 68000) and from (like most CPUs of the day) not including any cache. This simplicity enabled low power consumption, yet better performance than the Intel 80286. A successor, ARM3, was produced with a 4 KB cache, which further improved performance.[1]

In the late 1980s, Apple Computer and VLSI Technology started working with Acorn on newer versions of the ARM core. In 1990, Acorn spun off the design team into a new company named Advanced RISC Machines Ltd., which became ARM Ltd when its parent company, Arm Holdings plc, floated on the London Stock Exchange and NASDAQ in 1998. The new Apple-ARM work would eventually evolve into the ARM6, first released in early 1992. Apple used the ARM6-based ARM610 as the basis for their Apple Newton PDA.[1]

In 1994, Acorn used the ARM610 as the main central processing unit (CPU) in their RiscPC computers. DEC licensed the ARMv4 architecture and produced the StrongARM. At 233 MHz, this CPU drew only one watt (newer versions draw far less). This work was later passed to Intel as part of a lawsuit settlement, and Intel took the opportunity to supplement their i960 line with the StrongARM. Intel later developed its own high performance implementation named XScale, which it has since sold to Marvell. Transistor count of the ARM core remained essentially the same throughout these changes; ARM2 had 30,000 transistors, while ARM6 grew only to 35,000.[1]

In 2005, about 98% of all mobile phones sold used at least one ARM processor. In 2010, producers of chips based on ARM architectures reported shipments of 6.1 billion ARM-based processors, representing 95% of smartphones, 35% of digital televisions and set-top boxes and 10% of mobile computers. In 2011, the 32-bit ARM architecture was the most widely used architecture in mobile devices and the most popular 32-bit one in embedded systems. In 2013, 10 billion were produced and "ARM-based chips are found in nearly 60 percent of the world's mobile devices".[1]

2 General characteristics

The ARM Cortex-A72 is a microarchitecture implementing the ARMv8-A 64-bit instruction set designed by ARM Holdings' Austin design centre. The Cortex-A72 is a 3-wide decode out-of-order superscalar pipeline. It is available as SIP core to licensees, and its design makes it suitable for integration with other SIP cores (e.g. GPU, display controller, DSP, image processor, etc.) into one die constituting a system on a chip (SoC). The Cortex-A72 was announced in 2015 to serve as the successor of the Cortex-A57, and was designed to use 20% less power or offer 90% greater performance. It has one to four cores in a single processor device with L1 and L2 cache subsystems.[2] The following figure 1 shows an example block diagram of a Cortex-A72 processor configuration with four cores.

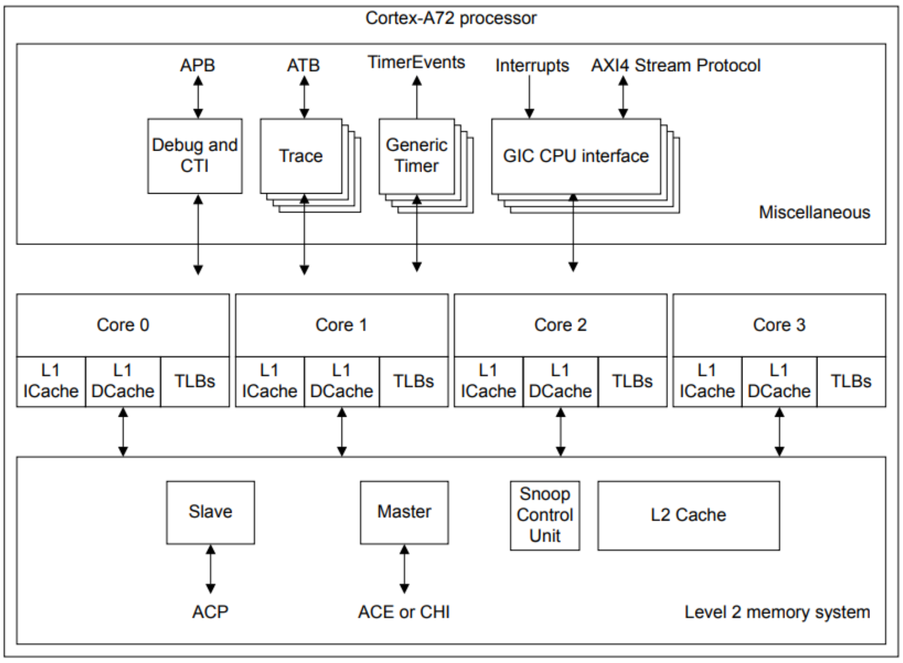

Figure 1: Example Cortex-A72 processor configuration

The Cortex-A72 boasts up to a 50 percent energy reduction when compared with the Cortex-A15 and a 20 percent saving compared with the A57, at the same clock speeds. Milliwatts per core have dropped from the A57, to around 700mW at 2.5GHz.[3] The design takes up 10 percent less area than the A57, which will also help save on costs. We can see this in figure 2.

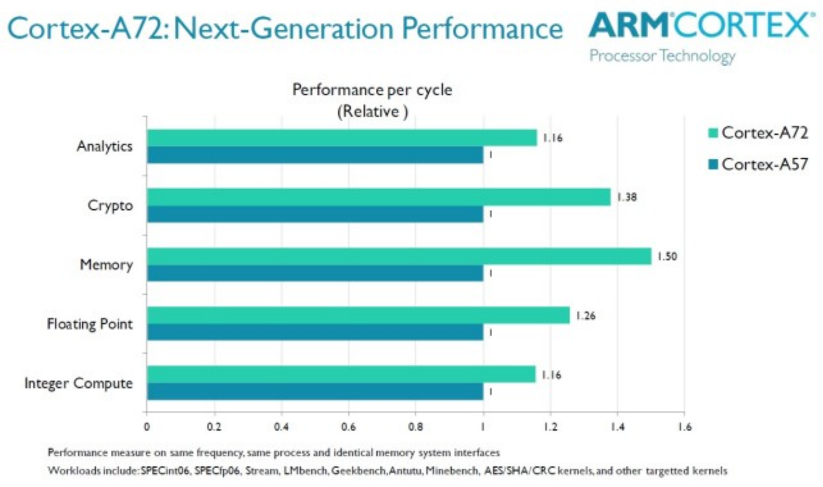

Figure 2: Cortex-A72 performance

As well as substantial energy savings, ARM reckons that the A72 will be able to sustain 2.5GHz clocks on the new 16nm process, whilst keeping within the limited smartphone power budget as seen in figure 3. It's the additional power efficiency and resulting lower heat profile that will really help the A72 achieve higher clock speeds than a 16nm A57.[3]

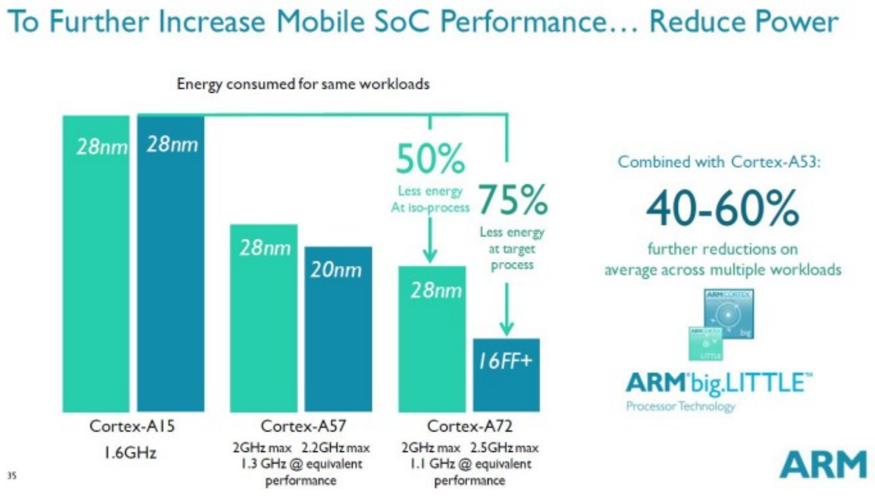

Figure 3: Cortex-A72 power reduction

ARM is looking to differentiate its high performing designs from their lower energy counterparts. The A53 and A57 are quite different in their design and intended applications, so switching the more powerful cores over to the A7x naming scheme should help avoid any confusion in the future.[3]

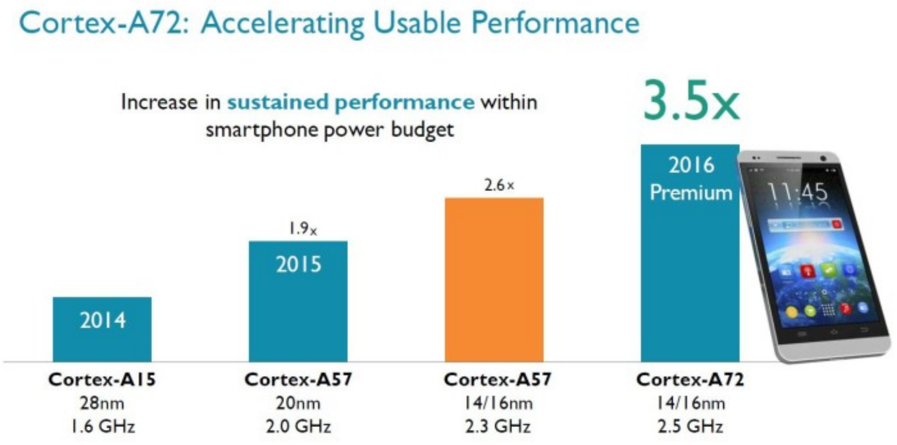

Figure 4: Cortex-A72 usable performance

The key point to take-away is that ARM has focused heavily on improving power and area efficiency with the A72, which is always welcome in mobile products. This also has the added benefit of allowing the chip to run cooler and to be clocked slightly higher than its predecessor.[3]

3 Architectural characteristics

4 Internal organization/structure (datapath)

The internal structure is the one pictured in Figure 1. And the details of the blocks are described in section 2.1.1 of source [2].

Fetch, decode and dispatch units

Instruction Fetch

Up to 3 instructions are fetched from the L1 instruction memory. The cache has 48KB of size in a 3-way associativity configuration with 64-bits-long cache lines; and optionally, parity bits for error-checking in the data and tag. In order to facilitate address prediction and translation, the architecture provides this features:

- For fast virtual address translation, a Translation Lookaside Buffer (TLB) is with support for pages of sizes 4KB, 64KB, and 1MB.
- 2-level dynamic Branch Target Buffer (BTB). A static predictor is also provided.

Instruction decode

Types of instructions able to be decoded by the CPU (including)[2]:

- A32: 32-bit wide, aligned on 4-byte boundaries. Both A-profile and R-profile are supported (application and real-time applications respectively) It is the most supported ISA. [4]
- T32: mixed 32 and 16-bit length instruction set, aiming at lost memory footprint and cost. It's the only ISA supported by type M instructions (specific to microcontrollers)[6]
- A64. Instructions semantics similar to A32 and T32, with 31 64-bit general purpose registers; independent from Program Counter and Stack Pointer, and hard-coded zero register. [5]

  This unit performs register renaming to facilitate out-of-order execution; thus removing both WAW and WAR hazards.

Instruction dispatch

The dispatch unit controls when the instructions that are decoded are ready to be issued, and afterwards, when the the results are retired. [2]

Functional units for execution and memory access

This set of functional units is composed by the units for operating with integers, floating point values, and memory accesses. [2]

Integer execute

- 2 ALU pipelines.
- Integer multiply-accumulate and ALU pipelines.
- Iterative integer dividing unit.
- Unit for resolution and prediction of branches.
- Result forwarding routes.

Load/Store unit

This unit is responsible of accessing L1 memory cache and service memory coherency requests with the second cache level (L2).

L1 is a 32KB 2-way associative cache with 64-bits-long lines, and can optionally implement Error Correction Code (ECC)[1] for every 32 bits. Apart from the aforementioned TLB (see Instruction Fetch), this unit also uses automatic pre-fetching targeting the L1 data cache and L2. [2]

Level-2 of cache

Services the both the data and instruction L1 in case of cache miss, and keeps coherence. For that, it keeps a duplicate of the tag fields of each L1 cache. Its size is either 12KB, 1MB, 2MB, or 4MB; with a 16-way associativity with optional ECC data per 64 bits; and a 1024-entry TLB **for each processor**. This cache, as explained in the section 7.2 of [2] has a *dirty* bit per line of cache (That is, per 64 bits).

Advanced SIMD and floating-point

It gives support to ARMv8 advanced SIMD (Simple Input, multiple data) and floating-point operations. Optionally, it is up to the implementation to include an optional cryptography engine, albeit with contractual rights. [2]

GIC CPU interface

This unit delivers interrupts to the processor. Examples of interruptions: device signals (device ready, read/write completed, error; battery and power), clock IRQ (for scheduling), exceptions. [2]

Interrupts and debug

Generic Timer

Provides with the ability to trigger interrupts and trigger events. [2]

[1]The difference between using parity bits in memories and using ECC memory is that, while parity bits helps **detecting** errors; ECC memory can both **detect** and **repair** those errors

Debug and trace

- ARMv8 debug architecture with slave architecture with debug registers. It provides the programmer with registers for discerning the interrupts **source**, **behavior** to apply, and **routing** to other processors.
- Performance Monitor Unit: creates statistic data from **runtime** status of the CPU and the memory
- Embedded Trace Macrocells: performance of real-time instruction flow tracing.

  [2]

5 Cache memory

Cache level	Sizes available	Associativity	Cache line length
Level 1 - Data	32KB **per core**	2-way set assoc.	64 bits
Level 1 - Instruction	48KBs	3-way set assoc.	64 bits
Level 2	12KB, 1MB, 2MB, or 4MB	16-way associativity	64 bits

Writes to main memory use as policy **Write-back-write-allocate**. This means that updating MM only takes place when cache is *dirty* and prompt to be discarded, and not on MM directly; and upon a miss in cache, the block is brought to the upper levels and allocated there for it to be read or written. [2] 6.4.1

6 I/O support

References

[1] Wikipedia. *ARM architecture.*

[2] *ARM® Cortex®-A72 MPCore Processor Technical Reference Manual.*

[3] Robert Triggs. *A closer look at the ARM Cortex-A72.*

[4] *Arm Developer - A32 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/a32

[5] *Arm Developer - A64 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/a64

[6] *Arm Developer - T32 instruction set* - last access: 12/12/2019. Available at https://developer.arm.com/architectures/instruction-sets/base-isas/t32