WHITE PAPER

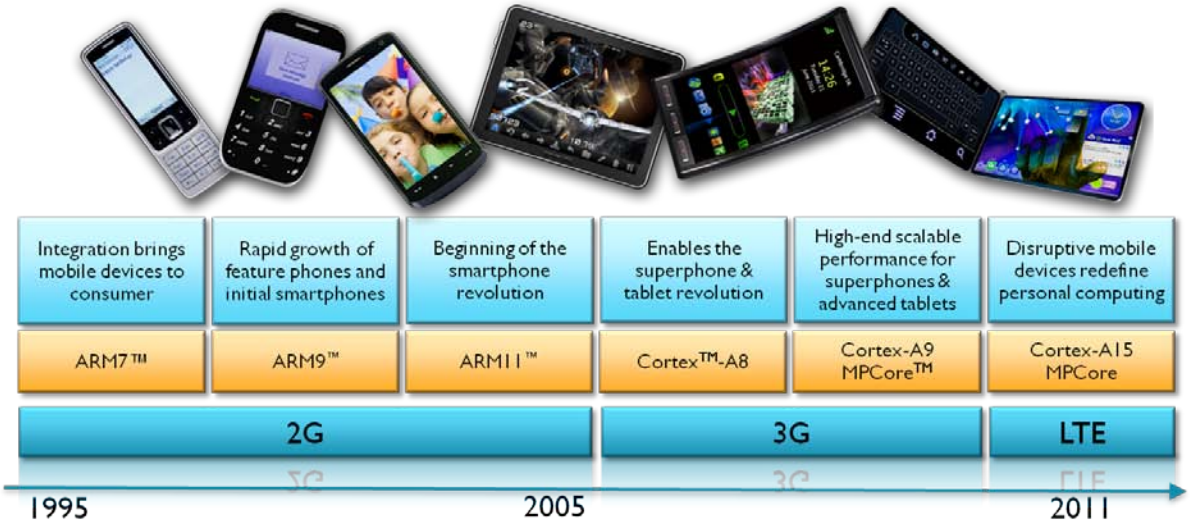
Technology Preview: The ARMv8 Architecture

John Goodacre, Director Program Management
ARM Processor Division, November 2011

Background: The ARM architecture

The ARM® RISC based architecture has evolved over the last 20 years and gained widespread recognition with the ARMv4 variant on the introduction of the ARM7™ family of processors. It was designed from the outset to be optimized for low power operation. At that time there was no floating point, no complex math instructions, no SIMD, not even the ability to divide two integer numbers. Fast forward to 2011 and now all the ARM Cortex™ family of processors are designed using the ARMv7 architecture

The market breadth of adoption of ARM processors had also started to extend well beyond just mobile devices and so the ARMv7 architecture, although fully consistent, has been split into three profiles that better match the specific requirements of the "Application" A-Profile; "Real-time" R-Profile and the "Microcontroller" M-Profile markets.



The Cortex processors have quickly become the primary choice throughout the industry. Within the A-profile, introduced by the Cortex-A8, the Cortex-A9 brought

SMP multiprocessing to the masses, and the Cortex-A5 brought it to the low cost, low-power internet connected ARM926-class devices. Cortex-A5 also started the opportunity to utilize the latest and richest ARM architecture features at the lowest Internet connected price point. Together these Cortex processors account for the majority of ARM's new business. In October 2011 ARM introduced the Cortex-A7 and the big.LITTLE energy- efficiency scheme to once more change the ability of the industry to deliver outstanding entry level smartphones with extended battery life.

It was clear to ARM from the level of adoption of the ARMv7 architecture and accompanying growth in the associated ecosystem and app stores, that any new enhancements to the architecture had to be fully backwards compatible. It was equally clear that to support any migration to a new architecture; the new architecture had to bring significant advantage over the existing architecture, even when running software that did not use any of the new features.

Historically in the computer industry, such enhancements had been accomplished by adding new functionality on top of the old. Attempts in the industry to remove legacy issues and start with a clean slate were all too common with a range of new architectures attempting to answer the problems of the old with a clean break. Thus, the critical foundation of ARMv8 had been set.

ARM processor-based devices had been seen as following the capabilities of the PC market by a few years. Recently however, the performance profile of a Cortex-A9 processor-based device, as used in almost all the tablet class of device, has unquestionably established that the performance of an ARM solution can deliver the experience that the vast majority of PC users require at much lower power consumption. This performance point means that ARM is being adopted into many other classes of device, most notably enterprise devices and servers where the power consumed per unit of performance is enhanced by ARM-based devices.

There are also some basic trends that have fed requirements into the next architecture version. Systems on Chip (SoC) are becoming more and more integrated. The sub-nanometer fabrication technologies allow for an unimaginable level of integration on a single device. Such devices have the performance and the need to support richer and more complex software. The demand on the memory to hold these applications is continuously growing. The complexity of a single application is starting to need to address more memory than can be addressed by a 32-bit architecture alone.

The Cortex-A15 processor was released under the ARMv7 architecture; however a couple of optional extensions have been introduced to support key market

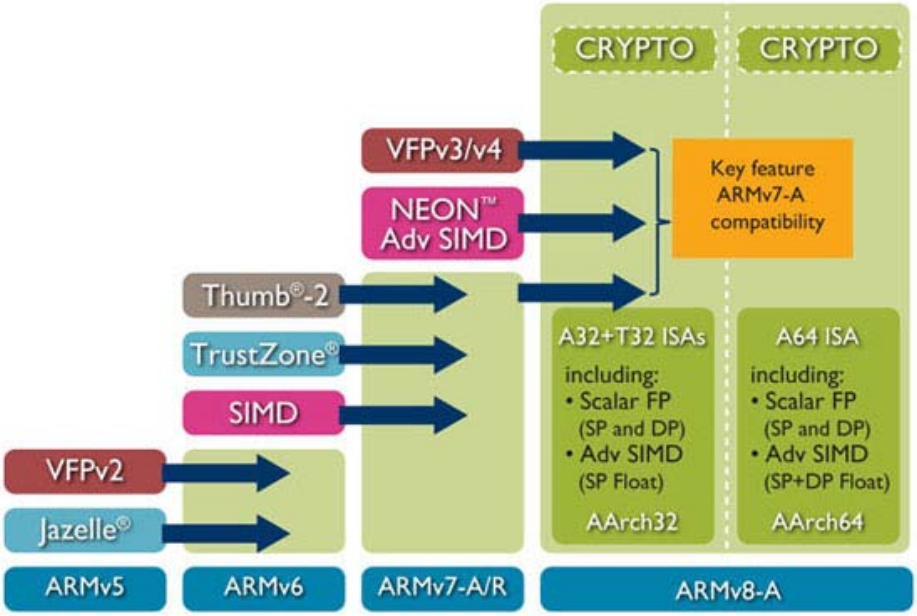
requirements; Large Physical Address Extensions (LPAE) and Virtualization. To support higher levels of device integration on a single memory space, it is necessary to place each device, and each device's communication (and potentially working memory) in a common address map. Unlike in the PC market, for example, where the GPU and the CPU have different address spaces, the lower power, and lower cost solution requires sharing the same memory space. The solution was LPAE; a new page table format supported by the Cortex-A15 that allows the mapping of multiple 32-bit virtual address spaces into up to a 40-bit physical address space using a 4KB page resolution. This fully addresses the need for the increased SoC integration and the address map restrictions being suffered by many of today's SoC devices. It also addresses the need for multiple applications to exist on a device at the same time and each having access to up to 4GB memory without the embedded device needing to include a hard disk on which to swap out application memory – something that perhaps most phone users would not appreciate. The Cortex-A15 also introduced full hardware accelerated virtualization. This supports the increased demands that software make of a device and to provide a technique to better partition the software into more manageable and potentially independent units.

So what has been changed in ARMv8, if ARMv7+LPAE solves the 4GB limitations? Does ARMv8 need to be considering a full 64-bit instruction set architecture with its larger virtual address space architecture or not? In terms of ARM's current markets, the need to support a single application that requires more than 4GB address space is rather small.

Trends. That's really what ARM has to look at when defining a new architecture. That is the nature of our business, we need to look a long way forward, and plan. ARMv8 as of today is one of the largest programs ever to be managed within ARM. The ability to innovate and change the world around us is done by software. The hardware, and in ARM's case, the processing beneath it are enablers; important, but still just enabling software to be able to deliver the experience and value to the consumer.

Fundamentals of the ARMv8 Architecture

The first thing to note about the ARMv8 architecture is that is has been defined for the Cortex A -profile processor markets only at this point. The ARMv7 architecture still has many generations of life left in it addressing the needs of the real-time and microcontroller markets. In fact, many of the A-profile markets needing extension beyond a 32-bit address space are being addressed by the Cortex-A15 with its LPAE capability. LPAE defines a page table format to enable mapping virtual pages into a 40-bit physical address space. This was done however in a manner that allows us to overlay both a larger virtual and physical address space. For ARMv8-A, this is defined as up to 48-bit sign-extended virtual address and up to 48-bit physical address in a manner that has a minimal impact to software developed today for the Cortex-A15.



Fundamental to ARMv8 has to be the new instruction set, known as A64; the encoding of instructions to enable an application to utilize a 64-bit machine. ARM took the decision to introduce 64-bit through a new instruction set rather than extension of an existing instruction set for many good reasons. Most notably, and probably as no surprise, because we could develop a new independent instruction set to execute code in a lower power manner than by adding instructions to the existing instruction set. Of course, for compatibility reasons, we still support the entire ARMv7 machine in the new ARMv8 architecture, but when running 64-bit software, this part of the machine is not being used, and the area of complex legacy it had built up does not need to be active when running in the 64-bit ISA, unlike other

architectures where 64-bit extension was simply added to the historical complexity and legacy of their 32-bit mode. The new ISA drew upon the years of experience of building different micro architecture implementations, so again it was defined so that these new processors can be more easily optimized for low power operation – an opportunity not really offered since the first ARMv4 machine that resulted in the now legendary low power ARM7 processors.

The new ARMv8-A architecture also offered the opportunity to optimize the software exception model and to improve the utilization of the silicon, again providing the opportunity to lower a system's power consumption.

Clearly this new architecture needed to enable software to support the 32-bit heritage, but unlike the earlier Thumb instruction sets, it was decided that transitions between 32-bit and 64-bit execution states would only occur on exception boundaries, known as inter-processing, and not through the traditional ARM to Thumb interworking. This permits a strict hierarchy between the various supported levels of execution privilege.

A64 Instruction Set

Each instruction in A64 is defined within a fixed length 32-bit instruction. For the hardware the original thoughts on adapting an existing decode table as a means of sharing the decoder between 32-bit and 64-bit instruction sets were rejected in favor of a clean decode structure with contiguous bit fields for operands and immediate values. This not only simplified the decode table in the hardware but also provided JIT compilers with important acceleration techniques which are key to high performance web browsing and other applications. The independent decode also permits some of the more advanced branch prediction techniques. The number of general purpose registers has also been increased. Although the virtual rename register pooling introduced in the Cortex-A9 provided hardware with an automatic way of unrolling small loops, it did not give the compiler the full advantage of more registers to provide improved scheduling options for the increasingly complex algorithms that are becoming common across various software codes. The A64 ISA therefore introduced thirty one 64-bit general purpose registers.

Another change in the ISA is actually one of simplification. Looking back to the original RISC goals of the ARM ISA, the new A64 ISA has removed the LDM/STM (load/store multiple) instructions that for a long time had cost complexity in the implementation of an efficient processor's memory system. There are fewer conditional instructions, since the implementation complexity did not provide a relative benefit.

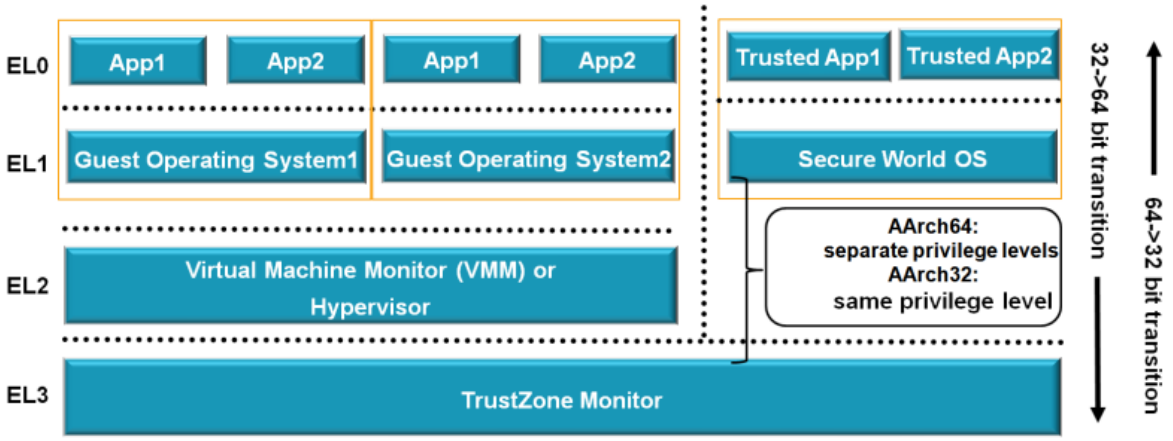
The final aspect of note in the instruction set is that the floating point unit will always be present. This will mean commodity software can assume FP is always available, and therefore will not require checking for its existence providing the software with underlying hardware consistency.

The SIMD data engine's instruction set is also now part of the base architecture and has been revised for operation in the new 64-bit world. It introduces double precision floating data processing to the SIMD capability while also undergoing simplification to better address the algorithms to which it has been targeted in line with the latest IEEE 754-2008 standard.

Exception Levels

The exception levels in AArch64 are the same as those in the Cortex-A15 through the addition of the HYP mode for hypervisor support inserted between the OS kernel mode and the TrustZone® monitor mode. The secure world still only supports a single OS instance for reasons of simplicity associated with approvals for secure software.

The exception hierarchy also defines a strict set of rules for what transitions are valid between 32-bit and 64-bit operation. As execution elevates or takes exception at an increased level of privilege, operation can only be either at the same width, or an increased width of ISA. It is not possible, for example, for a 32-bit hypervisor to support a 64-bit operating system while executing in the HYP mode.

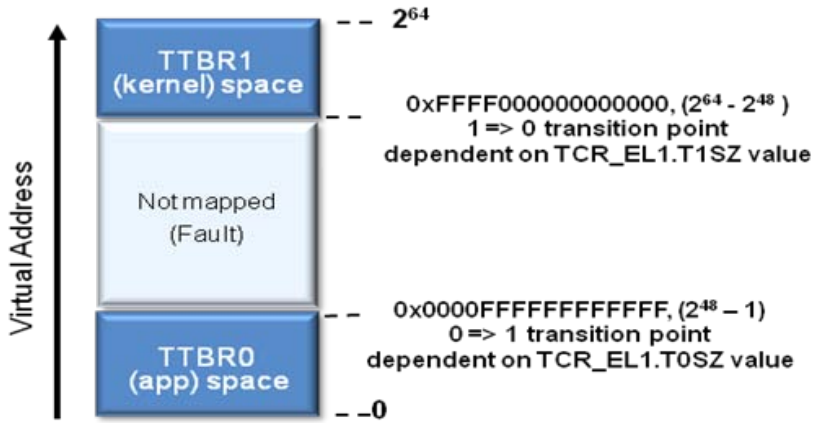


To help support this new exception model, A64 also introduces a dedicated exception link register (ELR) that is written on all exception entry. When moving from a 32-bit mode to enter a 64-bit exception it will also be automatically zero extended. Interrupt masks are also set automatically on exception entry. Each exception level

has its own vector base address registers and each vector is distinguished by type; synchronous, IRQ, FIQ or Error. The origin of the exception is also available from the vector and additional details about the exception are supplied in a syndrome register. This is a specifically useful feature to enable the virtualization of IO devices where any virtual machine's access to the device is trapped to an exception in the hypervisor, and as such the hypervisor can simply read this information to evaluate which operation is to be virtualized.

Memory Management Unit

The memory management unit defined by AArch64 is fundamentally the same as that used in the Cortex-A15 except for the ability to now support both 48-bits of virtual and physical address. Bounding support to 48-bit has the advantage that we could again simplify the hardware such that they are required to only support up to four levels of page table when required to decode an address, while also more importantly limiting the scope of complexity for validation. In fact, AArch64 also now natively supports a 64KB minimum page size in addition to the more familiar 4KB page and as such can reduce the required walk from four to two levels where a 42-bit address is sufficient.



Any 32-bit code will of course be limited to operating in the first 4GB of address space, and as such the hardware will automatically zero-extend the virtual address into any elevated 64-bit call. To provide a basic memory map, the architecture also provides two base addresses from which the virtual addresses used for access to the OS services and the application data can grow. Virtual address spaces from 2^{32} to 2^{48} bytes in size are supported from the top and bottom of the 64-bit address space.

As with the Cortex-A15, the ARMv8 memory management unit provides two stage translations from an application virtual address (VA), to the intermediate physical address (IPA) used by any hypervisor, and then through to the actual physical address (PA) placed on the memory bus. The IPA and PA are implementation defined to support between 32 and 48-bits of address space.

Program Registers

One of the main differences any assembler or compiler writer will notice with the A64 instruction set is the access to 30 general purpose registers. Each register is 64-bit wide and in the assembler language are known as X0 through X30, which of course is 31 registers and that is because X30 is not strictly general purpose since it is used as the Procedure Link Register (PLR).

Unlike in the AArch32 environment where each operational state has a number of banked registers, in AArch64 only a Stack Pointer (SP), the Exception Link Register (ELR) and the storage for the Saved Process State (SPSR) is banked. Unlike AArch32 for the current processor state, known as PState in AArch64, the current process status is accessed independently by category, for example the flag bits, so that atomic read/write/modify of a bit field within a register is not required.

	X0-X7	X8-X15	X16-X23	X24-X30
0	R0	R8_usr	R14_irq	R8_fiq
1	R1	R9_usr	R13_irq	R9_fiq
2	R2	R10_usr	R14_svc	R10_fiq
3	R3	R11_usr	R13_svc	R11_fiq
4	R4	R12_usr	R14_abt	R12_fiq
5	R5	R13_usr	R13_abt	R13_fiq
6	R6	R14_usr	R14_und	R14_fiq
7	R7	R13_hyp	R13_und	No Register

So that a 64-bit OS or hypervisor can manipulate a 32-bit environment, there is also an architected relationship between the registers. There is a simple mapping of all the 32-bit registers directly into the bottom 32-bits of each of the 64-bit registers.

Debug

The concept and philosophy used between the ARMv7 and ARMv8 debug architecture is the same. You can still run instructions which have been sent directly from the debugger and the processor has an alternative channel to extract the executed instructions. There are still the two types of invasive debug through self-hosting and halted mode debug. The ARMv8 debug is similar to that presented in the

ARMv7 devices except that all break points and watch points have grown to support the 64-bit addressing and the instructions that can be directly executed in debug state are limited to a small subset of those available.

However, it should be noted that the halting debug view is no longer compatible with tools that today support ARMv7 halted debug even if the processor is running only 32-bit code. For this to work it will be necessary to have the debugger updated to support ARMv8. Self hosted AArch32 debug used by an OS however does not change.

Ecosystem evolution

Clearly the main challenge of introducing a new instruction set is not the development of the hardware, but the migration and support for the tools and software in the ecosystem. ARM has an enviable ecosystem, with a richness and depth commensurate with the trend that predicts that in the next twelve months, 10 billion more ARM-powered devices will be delivered to market.

For this reason, the ARM processors will run today's 32-bit software without alteration and as such no impact to the ecosystem. Only key targeted areas of software will initially need to consider operating within A64 instruction set. The first is likely to be a hypervisor or any secure monitor code in such a system. Since the new hardware accelerated virtualization was first introduced in the Cortex-A15, which already supported much of the new page table format, updating such a hypervisor is manageable and many vendors have already started.

The next area of software that will need to consider supporting 64-bit operation will be the operating system. ARMv8 introduction is focused at the A- profile devices running open platforms, which limits the number of such OSs. ARM, along with its key partners, has been working for many months on adding 64-bit support for ARM in Linux and all the associated tools. The Linux work includes support of 32-bit and 64-bit applications stored within the same file system, known as multi-architecture support. This too, is a manageable task since in many cases the OS has already made the migration to supporting a 64-bit architecture, and therefore porting to the ARMv8 64-bit instruction set is less daunting.

That leaves the vast majority of the ecosystem unchanged. The new 64-bit OS can easily and efficiently support those existing 32-bit applications – additionally, they can now also cleanly resolve the address map congestion and physical RAM limitation. If an application does like the idea of a little more performance, the ability to address more RAM, and operate in the most power efficient manner, they can of

course recompile, and where necessary, port to the new A64 ISA. The schedule of which will be dependent on the market need.

Closing Remarks

So why is ARM saying anything about its intent to support 64-bit computing? As has been described many times, the time between the architectural update, chips which support this architecture and final devices in market using the new feature can be several years. In the Open Source world, it can also take many years for changes to the Linux and associated code base to be submitted and flow into the ecosystem. So that there is software available when devices start to be manufactured, it is necessary to start to move this code into the ecosystem before silicon devices are available – and hence the disclosure of the ARMv8 architecture. While we continue to prepare the solid foundation of specification, tools, models and the first phases of OS porting, we could have chosen to simply expose the new code, or provide some technology preview of this new code. However, with so much interest in the topic, we have decided it would answer a number of questions if we provided a technology preview of the new architecture while also enabling the required public discussion and disclosures of the technology.

Unless you are an open platform operating system vendor, or a hypervisor vendor, at this stage you probably don't need to do anything other than consider whether a move to 64-bit computing is in your future and is meaningful.

The work is currently with ARM and across a number of key ARM partners, developing processor implementations of the ARMv8 architecture. Targeting various different markets, some very new to ARM, others much more traditional, one thing is for sure however, the future keeps moving forward, and with ARM you have the largest support and ecosystem of any technology ready to support any change, as and when it will be required.