ARMv8-A Power management

Version 1.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
08 March 2017	0100	Non-Confidential	First release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents

1 ARMv8-A Power Management

Many ARM systems are mobile devices and powered by batteries. In such systems, optimization of power use, and total energy use, is a key design constraint. Programmers often spend significant amounts of time trying to save battery life in such systems. Built into ARM cores are many hardware design methods that are aimed at reducing power use.

Power-saving can also be of concern even in systems that do not use batteries. For example, you might want to minimize energy use for reduction of electricity costs to the consumer, for environmental reasons, or to minimize the heat that the device generates.

Energy use can be divided into two components:

Static Static power consumption, also often called leakage, occurs whenever the core logic or RAM blocks have power applied to them. In general terms, the leakage currents are proportional to the total silicon area, meaning that the bigger the chip, the higher the leakage. The proportion of power consumption from leakage gets higher as you move to smaller fabrication geometries.

Dynamic Dynamic power consumption occurs because of transistor switching and is a function of the core clock speed and the numbers of transistors that change state per cycle. Clearly, higher clock speeds and more complex cores consume more power.

Power management-aware operating systems dynamically change the power states of cores, balancing the available compute capacity to the current workload, while attempting to use the minimum amount of power. Some of these techniques dynamically switch cores on and off, or place them into quiescent states, where they no longer perform computation. This means that they consume little power.

Note

The types of power management described here are not architectural in origin. The methods described here are examples of common practice in software methods of power management.

2 Idle management

Idle management is normally under the control of the operating system. In such a case, when a core is idle, *Operating System Power Management* (OSPM) moves it into a low-power state. Typically, a choice of states is available, with different entry and exit latencies, and different levels of power consumption associated with each state. The state that is used typically depends on how quickly the core is required again. The power states that can be used at any one time might also depend on the activity of other components in a SoC, beside the cores. Each state is defined by the set of components that are clock-gated or power-gated when the state is entered.

The time that is required to move from a low-power state to a running state, which is known as the wakeup latency, is longer in deeper states. Although idle power management is driven by thread behavior on a core, the OSPM can place the platform into states that affect many other components beyond the core itself. If the last core in a cluster becomes idle, the OSPM can target power states that affect the whole cluster. Equally, if the last core in a SoC becomes idle, the OSPM can target power states that affect the whole SoC. The choice is also driven by the use of other components in the system. A typical example is placing memory in self-refresh when all cores, and any other bus masters, are idle.

The OSPM has to provide the necessary power management software infrastructure to determine the correct choice of state. In idle management, when a core or cluster has been placed into a low-power state, it can be reactivated at any time by a core wakeup event. That is, an event that can wake up a core from a low-power state, such as an interrupt. No explicit command is required by the OSPM to bring the core or cluster back into operation. The OSPM considers the affected core or cores to be always available even if they are currently in a low-power state.

2.1 Power and clocking

One way that can reduce energy use is to remove power, which removes both dynamic and static currents (sometimes called *power-gating*), or to stop the clock of the core which removes dynamic power consumption only and can be referred to as *clock-gating*.

ARM cores typically support several levels of power management, as follows:

- Standby.
- Retention.
- Powerdown.
- Dormant mode.
- Hotplug.

For certain operations, there is a requirement to save and restore the state before and after removing power. Both the time taken to do the save and restore, and the power consumed by this extra work can be an important factor in software selection of the appropriate power management activity.

The SoC device that includes the core can have additional named low-power states, such as *STOP* and *Deep sleep*. These refer to the ability of the hardware Phase Locked Loop (PLL) and voltage regulators to be controlled by power management software.

Standby

In the standby mode of operation, the core is left powered-up, but most of its clocks are stopped, or clock-gated. This means that almost all parts of the core are in a static state and the only power drawn is because of leakage currents and the clocking of the small amount of logic that looks out for the wake-up condition.

This mode is entered using either the `WFI` (Wait For Interrupt) or `WFE` (Wait For Event) instructions. ARM recommends the use of a Data Synchronization Barrier (`DSB`) instruction before `WFI` or `WFE`, to ensure that pending memory transactions complete before changing state.

If a debug channel is active, it remains active. The core stops execution until a wakeup event is detected. The wakeup condition is dependent on the entry instruction. For WFI, an interrupt or external debug request wakes the core. For `WFE`, several specified events exist, including another core in the cluster executing the `SEV` instruction.

A request from the Snoop Control Unit (SCU) can also wake up the clock for a cache coherency operation in a cluster. This means that the cache of a core that is in standby state remains coherent with caches of other cores (but the core in standby does not necessarily execute the next instruction). A core reset always forces the core to exit from the standby condition.

Various forms of dynamic clock gating can also be implemented in hardware. For example, the SCU, GIC, timers, instruction pipeline, or NEON blocks can be automatically clock gated when an idle condition is detected, to save power.

Standby mode can be entered and exited quickly (typically in two-clock-cycles). It therefore has an almost negligible effect on the latency and responsiveness of the core.

To an OSPM, a standby state is mostly indistinguishable from a retention state. The difference is evident to an external debugger and in hardware implementation, but not evident to the idle management subsystem of an operating system.

Retention

The core state, including the debug settings, is preserved in low-power structures, enabling the core to be at least partially turned off. Changing from low-power retention to running operation does not require a reset of the core. The saved core state is restored on changing from low-power retention state to running operation. From an operating system point of view, as stated, there is no difference between a retention state and standby state, other than method of entry, latency, and use-related constraints. However, from an external debugger point of view, the states differ as External Debug Request debug events stay pending and debug registers in the core power domain cannot be accessed.

Power down

In this state, as its name implies, the core is powered off. Software on the device must save all core state, so that it can be preserved over the powerdown. Changing from powerdown to running operation must include:

- A reset of the core, after the power level has been restored.
- Restoring the saved core state.

The defining characteristic of power down states is that they are destructive of context. This means that all data, operating conditions and operating states are lost. This affects all the components that are switched off in a given state, including the core, and in deeper states other components of the system such as the GIC or platform-specific IP. Depending on how debug and trace power domains are organized, one or both of debug and trace context might be lost in some powerdown states. Mechanisms must be provided to enable the operating system to perform the relevant context saving and restoring for each given state. Resumption of execution starts at the reset vector, and after this each OS must restore its context.

Dormant mode

Dormant mode is an implementation of a powerdown state. In dormant mode, the core logic is powered down, but the cache RAMs are left powered up. Often the RAMs are held in a low-power retention state where they hold their contents but are not otherwise functional. This provides a far faster restart than complete shutdown, as live data and code persists in the caches. Again, in a cluster, individual cores can be placed in dormant mode.

In a cluster that permits individual cores to go into dormant mode, there is no scope for maintaining coherency while the core has its power removed. Such cores must therefore first isolate themselves from the coherence domain. They clean all dirty data before doing this and are typically woken up using another core signaling the external logic to re-apply power.

The woken core must then restore the original core state before rejoining the coherency domain. Because the memory state might have changed while the core was in dormant mode, it might have to invalidate the caches anyway. Dormant mode is therefore much more likely to be useful in a single core environment rather than in a cluster. This is because of the additional expense of leaving and rejoining the coherency domain. In a cluster, dormant mode is typically likely to be used only by the last core when the other cores have already been shut down.

Hotplug

Hotplug is a technique that can dynamically switch cores on or off. Hotplug can be used by the OSPM to change available compute capacity based on current compute requirements. Hotplug is also sometimes used for reliability reasons.

There are several differences between hotplug and use of a powerdown state for idle:

- When a core is hot unplugged, the supervisory software stops all use of that core in interrupt and thread processing. The core is no longer considered to be available by the calling OS.
- The OSPM has to issue an explicit command to bring a core back online, that is, hotplug a core. The appropriate supervisory software only starts scheduling on or enabling interrupts to that core after this command.

Operating systems typically perform much of the kernel boot process on one primary core, bringing secondary cores online at a later stage. Secondary boot behaves similarly to hotplugging a core into the system. The operations in both cases are almost identical.

3 Dynamic voltage and frequency scaling

Many systems operate under conditions where their workload is variable. Therefore it is useful to be able to reduce or increase the core performance to match the expected core workload. Clocking the core more slowly reduces dynamic power consumption.

Dynamic Voltage and Frequency Scaling (DVFS) is an energy saving technique that exploits:

- The linear relationship between power consumption and operational frequency.
- The quadratic relationship between power consumption and operational voltage. This relationship is given as:

  $P = C \times V^2 \times f$

  Where:

  P Is the dynamic power.

  C Is the switching capacitance of the logic circuit in question.

  V Is the operational voltage.

  f Is the operational frequency.

Power savings are achieved by adjusting the frequency of a core clock.

At lower frequencies, the core can also operate at lower voltages. The advantage of reducing supply voltage is that it reduces both dynamic and static power.

There is an IMPLEMENTATION SPECIFIC relationship between the operational voltage for a given circuit and the range of frequencies that circuit can safely operate at. A given frequency of operation together with its corresponding operational voltage is expressed as a tuple and is known as an *Operating Performance Point* (OPP). For a given system, the range of attainable OPPs is collectively termed as the system DVFS curve.

Operating systems use DVFS to save energy and, where necessary, keep within thermal limits. The OS provides DVFS policies to manage the power consumed and the required performance. A policy that is aimed at high performance selects higher frequencies and uses more energy. A policy that is aimed at saving energy selects lower frequencies and therefore results in lower performance.

4 Assembly language power instructions

ARM assembly language includes instructions that can be used to place the core in a low-power state. The architecture defines these instructions as hints, meaning that the core is not required to take any specific action when it executes them. In the Cortex-A processor family, however, these instructions are implemented in a way that shuts down the clock to almost all parts of the core. This means that the power consumption of the core is reduced so that only static leakage currents are drawn, and there is no dynamic power consumption.

Use of the `WFI` instruction suspends execution until the core is woken up by one of the following conditions:

- An IRQ interrupt, even if the PSTATE I-bit is set.
- An FIQ interrupt, even if the PSTATE F-bit is set.
- An asynchronous abort.

If the core is woken by an interrupt when the relevant PSTATE interrupt flag is disabled, the core implements the next instruction after `WFI`.

The `WFI` instruction is widely used in systems that are battery powered. Mobile telephones, for example, can place the core in standby mode many times a second, while waiting for you to press a button.

`WFE` is similar to `WFI`. It suspends execution until an event occurs. This can be one the event conditions listed or an event signaled by another core in a cluster. Other cores can signal events by executing the `SEV` instruction. `SEV` signals an event to all cores. The generic timer can also be programmed to trigger periodic events that wake up a core from `WFE`.

5 Power State Coordination Interface

The *Power State Coordination* Interface (PSCI) provides an OS agnostic method for implementing power management use cases where cores can be powered up or down. This includes:

- Core idle management.
- Dynamic addition and removal of cores (hotplug), and secondary core boot.
- big.LITTLE® migration.
- System shutdown and reset.

The messages sent using this interface are received by all relevant levels of execution. That is, if EL2 and EL3 are implemented, a message sent by the OS executing in a guest must be received by a hypervisor. If the hypervisor sends it, the message must be received by the secure firmware that then coordinates with a Trusted OS. This allows each operating system to determine whether context saving is required.