ARMv8-M Power management

Version 2.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
23 August 2016	0100	Non-Confidential	First release
01 March 22017	0200	Non-Confidential	Second release

Proprietary Notice

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-M Power management

Clocks, resets, and power control hardware are critical aspects of a system that the Secure code relies on for correct operation.

The level of, and type of protection these resources require depends on the attack profile being defended against, for example, remote software attack or basic hardware attack. As a minimum, any control registers that can influence clocks, resets, or power domains that can affect software running in the Secure state, or peripherals that are used by Secure software, must only be accessible from the Secure state.

To support use cases where the Secure state is disabled, the protection of these registers is configurable. System architects are also encouraged to consider the following extra protection measures.

Power on reset

Ensure that the power-on reset circuit can respond quickly enough to detect even short reductions in the power supply voltage.

The voltage threshold must also be set so that a reset is always generated before the voltage drops low enough to cause data corruption in registers, combinational logic, or SRAM.

As attackers are known to use unusual conditions, for example, very low temperatures, care must be taken to make sure the power-on reset either functions correctly at conditions outside the normal process corners, or exhibits fail-safe behavior, for example asserting reset.

Reset filtering

One possible way in which an attacker could cause data corruption is to introduce a short glitch on an external reset signal.

If the reset signal is not filtered correctly before being used internally it is possible that the glitch only partially resets the device, which could result in security vulnerabilities.

For immunity against cross talk and other types of noise most devices already contain reset filters, however in a similar way to the power-on reset the behavior of these circuits outside the normal operating conditions must be checked.

Clocks

Invalid behavior of external clock sources can also be a source of data corruption that can lead to security vulnerabilities.

The types of invalid behavior might include:-

- Excessive jitter.
- Extreme mark space ratios.
- Glitches.
- Clock frequencies outside the supported range.

  In addition to over or under clocking the main clock, attackers might try to exploit the system by using invalid ratios between two clocks, for example, the main clock source, and a synchronous peripheral interface, which could result in a buffer over or under flow condition.

You can use several methods to defend against these attacks:-

- Use trusted internal clock sources for critical operations.
- Clock conditioning circuitry that prevents critical digital logic being exposed to invalid clock behavior.
- Fail-safe behavior in the event of invalid clock signals, for example, asserting reset.

1.1 Sleep mode

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back into sleep mode.

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*.

The processor can have additional low-power states, with names such as STOP and Deep sleep. These refer to the ability for the hardware *Phase Locked Loop* (PLL) and voltage regulators to be controlled by power management software. Deep sleep mode stops the system clock and switches off the Phase Locked Loop (PLL) and flash memory.

The SLEEPDEEP bit of the *System Control Register* (SCR) selects which sleep mode is used.

Wait for interrupt

The wait for interrupt instruction, WFI, causes immediate entry to sleep mode unless the wakeup condition is true. When the processor executes a WFI instruction, it stops executing instructions and enters sleep mode.

Wait for event

The wait for event instruction, WFE, causes entry to sleep mode depending on the value of a one-bit event register.

When the processor executes a WFE instruction, it checks the value of the event register:

0 The processor stops executing instructions and enters sleep mode.

1 The processor clears the register to 0 and continues executing instructions without entering sleep mode.

If the event register is 1, it indicates that the processor must not enter sleep mode on execution of a `WFE` instruction. Typically, two events can cause this behavior:

- An external event signal is asserted.
- Another processor in the system has executed an `SEV` instruction.

Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handlers it returns to Thread mode and immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

Wakeup from sleep mode

The conditions for the processor to wake up depend on the mechanism that causes it to enter sleep mode.

Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry. Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler.

To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than the current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero.

Wakeup from WFE

Conditions which cause the processor to wake up from `WFE`.

The processor wakes up if:

- It detects an exception with sufficient priority to cause exception entry.
- It detects an external event signal.
- In a multiprocessor system, another processor in the system executes an SEV instruction.

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry.

The Wakeup Interrupt Controller

The *Wakeup Interrupt Controller* (WIC) is a peripheral that can detect an interrupt and wake the processor from deep sleep mode. The WIC is enabled only when the DEEPSLEEP bit in the SCR is set to 1.

The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

When the WIC is enabled and the processor enters deep sleep mode, the power management unit in the system can power down most of the processor. This has the side effect of stopping the SysTick timer. When the WIC receives an interrupt, it takes several clock cycles to wake up the processor and restore its state, before it can process the interrupt. This means interrupt latency is increased in deep sleep mode.

Note

If the processor detects a connection to a debugger, it disables the WIC.

The external event input

ARMv8-M processors provide an external event input signal. Peripherals can drive this signal, either to wake the processor from `WFE`, or to set the internal `WFE` event register to one to indicate that the processor must not enter sleep mode on a later `WFE` instruction.

Power management programming hints

CMSIS provides the following functions for these instructions:

```
void __WFE(void) // Wait for Event
void __WFI(void) // Wait for Interrupt
```