

Application Note 216

Implementing sleep control for low-power Cortex-M1 systems

Document number: ARM DAI 0216A

Issued: 8th October, 2008

Copyright ARM Limited 2008

The ARM logo is located in the bottom right corner of the page. It consists of the letters 'ARM' in a bold, sans-serif font, with a registered trademark symbol (®) to the upper right of the 'M'.

Application Note 216

Implementing sleep control for low-power Cortex-M1 systems

Copyright © 2008 ARM Limited. All rights reserved.

Release information

Change history

Date	Issue	Change
October 2008	A	First release

Proprietary notice

Words and logos marked with © and ™ are registered trademarks owned by ARM Limited, 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 Limited 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.

Confidentiality status

This document is Non-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.

Feedback on this Application Note

If you have any comments on this Application Note, please send email to errata@arm.com giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

ARM web address

<http://www.arm.com>

Table of Contents

1. Introduction	1-1
2. Block diagram	2-1
3. Hardware description	3-1
3.1 Signal descriptions	3-1
3.2 Programmer's model	3-2
3.3 Sleep mode	3-2
3.4 Wakeup events	3-4
3.5 Debug response state machine.....	3-5
3.6 Clock gating.....	3-6
3.7 Instantiating the sleep controller.....	3-7
4. Software description	4-1
4.1 Macro definitions	4-1
4.2 Programming the sleep controller	4-1
4.3 Requesting sleep entry.....	4-2
5. Considerations	5-1
5.1 Synthesis and layout	5-1
5.2 Level-sensitive and pulse interrupts	5-1
5.3 Interrupt priority	5-1
5.4 NVIC consistency	5-1
5.5 Multi-layer buses	5-1
5.6 Code compatibility	5-2
6. References	6-1

1. Introduction

The ARM Cortex-M1 processor implements the ARMv6-M architecture, which includes a Wait For Interrupt (WFI) instruction. The ARMv6-M architecture defines the WFI instruction as a NOP-compatible hint, which means that an implementation can execute it as a No Operation (NOP) instruction.

When a processor that supports WFI executes the WFI instruction, it suspends execution and enters a low-power state. When a processor that does not support WFI executes the WFI instruction, it has no effect because it executes as NOP. This ensures code compatibility between different processors.

The ARM Cortex-M1 processor always executes the WFI instruction as a NOP. However, if your application requires power management support it is possible to implement an external sleep controller that puts the processor into a similar low-power state. This Application Note describes how you can implement an external sleep controller for the ARM Cortex-M1 processor.

Before reading this Application Note, you should make sure that you are familiar with:

- the ARM Cortex-M1 processor;
- the AMBA3 AHB-Lite protocol.

2. Block diagram

Figure 2-1 shows a block diagram of the external sleep controller in an ARM Cortex-M1 based system.

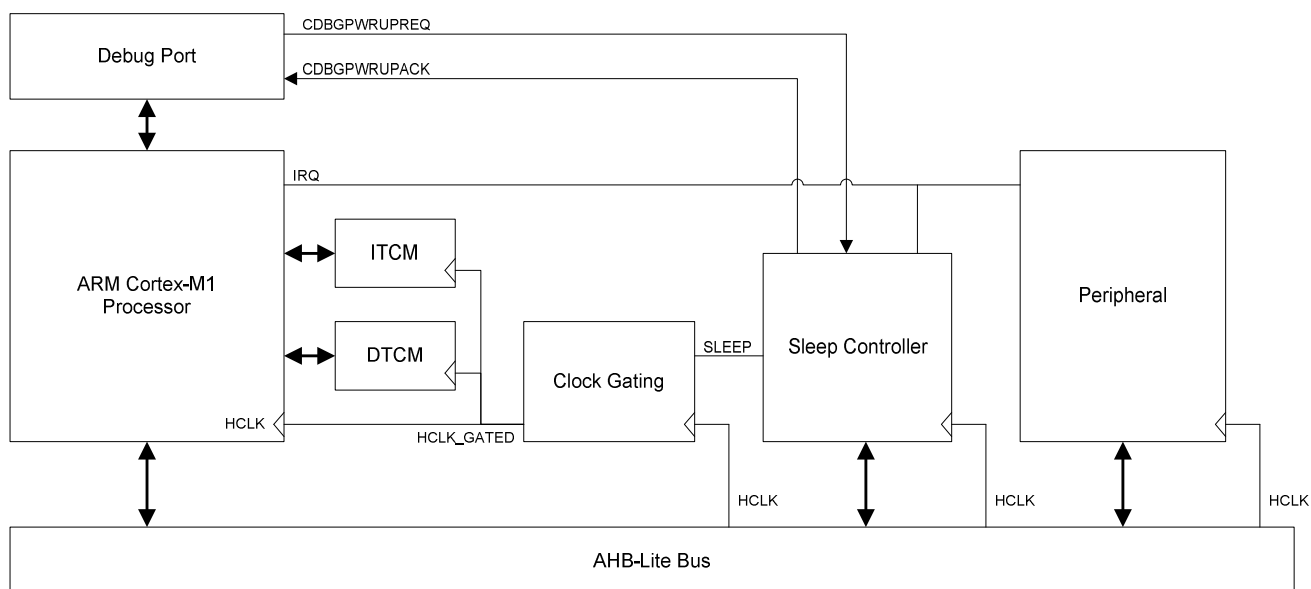


Figure 2-1 Block diagram of a sleep controller in an ARM Cortex-M1 based system

The components in the block diagram are described below.

Sleep Controller

The sleep controller is an AHB-Lite peripheral that controls clock gating logic to allow the processor to enter a low-power state. This Application Note contains guidelines and recommendations which you can use to implement your own sleep controller.

Clock gating logic

The clock gating logic gates the processor clock when the sleep controller asserts a **SLEEP** signal.

Peripheral

Any external peripheral can cause the processor to wake from sleep by asserting an interrupt. The interrupt signal is routed to both the processor and the sleep controller.

Debug Port

The Debug Port is the ARM CoreSight component that connects to an external debug interface. The ARM Cortex-M1 processor is supplied with a Serial Wire JTAG Debug Port (SWJ-DP). The debug port can wake the processor from a sleep state by using the power-up handshake signals that are part of the *ARM Debug Interface v5 Architecture Specification*.

3. Hardware description

This section describes a possible hardware implementation for a sleep controller that you can use to put the ARM Cortex-M1 processor into a low-power state.

The sleep controller peripheral is an AHB-Lite slave that the ARM Cortex-M1 processor can access to request entry into a low-power state. When the controller receives a sleep request it will put the bus into a waited state and gate the processor's clock until a wakeup event such as an interrupt is received. The effect of using the sleep controller is similar to the Wait For Interrupt (WFI) instruction in other ARM processors.

Note

The ARM Cortex-M1 processor does not include native power-management support, and executes the WFI instruction as an architected No Operation (NOP).

3.1 Signal descriptions

Table 3-1 lists the input and output signals on the sleep controller peripheral. The **HCLK** input to the sleep controller must be free-running.

Signal name	Direction	Clock domain	Description
HCLK	Input	HCLK	System clock
SYSRESETn	Input	HCLK	System reset
HSEL	Input	HCLK	AHB-Lite slave interface, see the <i>AMBA3 AHB-Lite Protocol Specification</i>
HADDR[31:0]	Input	HCLK	
HWRITE	Input	HCLK	
HTRANS[1:0]	Input	HCLK	
HSIZE[2:0]	Input	HCLK	
HBURST[2:0]	Input	HCLK	
HWDATA[31:0]	Input	HCLK	
HREADY	Input	HCLK	
HREADYOUT	Output	HCLK	
HRESP	Output	HCLK	
HRDATA[31:0]	Output	HCLK	
IRQ[31:0]	Input	HCLK	Interrupts
NMI	Input	HCLK	Non-maskable interrupt
CDBGPWRUPREQ	Input	SWCLKTCK ¹	Debug power-up request
CDBGPWRUPACK	Output	HCLK	Debug power-up acknowledge
SLEEP	Output	HCLK	Indicates that the core's clock is gated

Table 3-1 Sleep controller signal interface

¹ The SWCLKTCK clock domain assumes that an ARM SWJ-DP is used for the debug port

3.2 Programmer's model

The example sleep controller has several control registers that are accessed through the AHB-Lite slave interface. An example register set is described in Table 3-2, but you can modify this to suit your own application.

Address offset	Size	Name	Access type	Reset value	Description
0x0	32 bits	SLEEP	Read-only	0x00000000	The processor reads from this register to request entry into a sleep state.
0x4	32 bits	SETWAKE	Read/write	0x00000000	Write 1 to a bit to allow the corresponding IRQ input to wake the processor from sleep mode. Writing 0 to a bit has no effect. On reads, returns 1 in each bit position corresponding to an interrupt that will wake the processor from sleep.
0x8	32 bits	CLEARWAKE	Read/write	0x00000000	Write 1 to a bit to prevent the corresponding IRQ input from waking the processor. Writing 0 to a bit has no effect. On reads, returns 1 in each bit position corresponding to an interrupt that will wake the processor from sleep.

Table 3-2 Sleep controller registers

3.3 Sleep mode

3.3.1 Entry and exit

The peripheral asserts its **SLEEP** output when it puts the processor into a low-power sleep mode, and keeps it asserted until a wakeup event occurs. Sleep mode is entered when the processor reads from the SLEEP register, and there are no wakeup events active.

Note

ARM recommends that the sleep controller enters sleep mode following a read request from the SLEEP register, not a write. The ARM Cortex-M1 processor contains a write buffer that might cause a write to the AHB-Lite interface to be delayed. The sleep controller must be mapped into the peripheral region of the processor's memory space, which permits reads to have system side-effects. See Section 3.7 for more details.

A wakeup event can be:

- an incoming interrupt that has the corresponding bit set in the SETWAKE register;
- a non-maskable interrupt;
- a debug wakeup request.

Note

ARM recommends that the sleep controller unconditionally wakes the processor from sleep following a non-maskable interrupt (NMI). This is because NMIs are normally used to indicate critical system events that the processor must service immediately.

3.3.2 Sleep mode timing

Figure 3-1 shows the recommended signal timing for a sleep controller implementation.

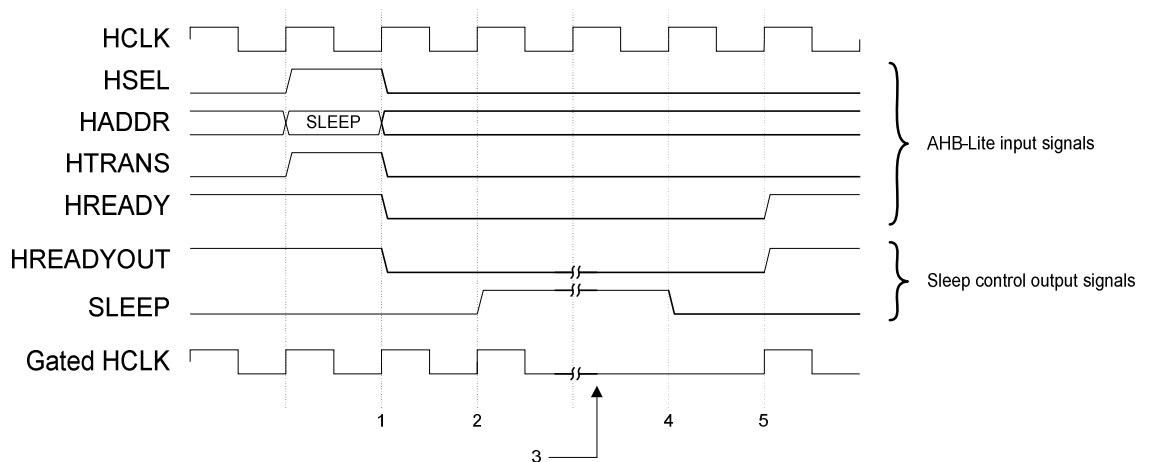


Figure 3-1 Sleep controller sleep signal timing

The **HREADYOUT** and **SLEEP** signals in Figure 3-1 are generated by the sleep controller. Every AHB-Lite slave has a **HREADYOUT** signal that signals when the slave is ready. When entering the sleep state in response to a valid bus transaction, the sleep controller:

- de-asserts **HREADYOUT** to stall all other transactions on the bus;
- enables clock gating once the bus is stalled.

In detail, the sleep controller responds to a sleep request as follows. The numbers refer to the labels in Figure 3-1.

1. The software running on the processor reads from the **SLEEP** register to request that the processor enters a low-power sleep state. As with all AHB-Lite slaves, the transaction to this peripheral is indicated with the asserted **HSEL** and **HTRANS** signals, and is qualified by the main **HREADY** signal.
2. When there are no active wakeup events, such as an active debug connection, the sleep controller grants the sleep request. It does this by stalling the AHB-Lite transaction with a low **HREADYOUT** in the data phase of the transaction.

On the next clock cycle, when the core has sampled the **HREADYOUT** signal, the sleep controller asserts the **SLEEP** output to the clock gating logic.

3. While the processor is sleeping, a wakeup event such as an interrupt or a debugger connection is detected by the sleep controller. The sleep controller starts the wakeup process.
4. When waking the processor from sleep, the sleep controller first de-asserts the **SLEEP** signal so that the clock gating logic enables the processor clock.
5. One cycle later, when the core has sampled the low **HREADYOUT** signal, the sleep controller asserts the **HREADYOUT** signal to complete the data phase of the AHB-Lite transaction.

Note

Figure 3-1 does not show all AHB-Lite signals. You must ensure that the sleep controller complies with the AHB-Lite specification, as documented in the *AMBA3 AHB-Lite Protocol Specification*. The specification requires that a peripheral drives data on the enabled byte lanes in **HRDATA** during the data phase of a read transaction, so the sleep controller must drive **HRDATA** when reading from the **SLEEP** register. Software can discard this data.

3.4 Wakeup events

When the sleep controller has put the processor into sleep mode, a wakeup event must cause it to exit sleep mode. As mentioned in Section 3.3, a wakeup event can be:

- an interrupt that has the corresponding bit set in the SETWAKE/CLRWAKE registers;
- a non-maskable interrupt;
- a debug wakeup request.

A wakeup request signal can be formed as the logical OR of these three events. Interrupt requests are detected from the **IRQ[31:0]** and **NMI** pins, and a debug wakeup request is detected through a debug handshake state machine as described in Section 3.5.

The sleep controller described in this Application Note will only work with peripherals using level-sensitive interrupts. This means that peripherals requesting an interrupt must keep their interrupt signal asserted until the software in the interrupt service routine running on the processor acknowledges the interrupt. You can extend this scheme to support pulse interrupts by:

- latching peripheral interrupts in the sleep controller when the processor is sleeping;
- creating interrupt and NMI output ports on the sleep controller that connect to the processor's **IRQ[31:0]** and **NMI** inputs, respectively;
- converting pulse interrupts from peripherals to level-sensitive interrupts on the processor's input port, using an extra register in the sleep controller that the processor can write to in order to clear an interrupt.

Note

If you extend the sleep controller to support pulse interrupts, you must also consider what changes are required to your software. Under a normal pulse interrupt scheme, the software running on the processor does not need to clear or acknowledge the interrupt. However, if you use the sleep controller to provide level-sensitive versions of the interrupts to the processor, you must add extra code in your interrupt service routines to clear the interrupts in the sleep controller.

An example of how to generate a wakeup signal, **wake_request**, using a level-sensitive interrupt scheme is shown in Figure 3-2. In the diagram, **wake_reg[31:0]** is the 32-bit internal register in the sleep controller containing the wakeup-enable bits. This is the internal register that is modified using the SETWAKE and CLEARWAKE registers in the programmer's model.

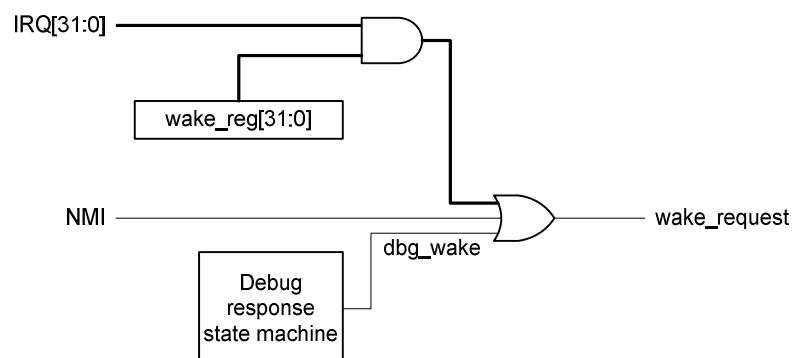


Figure 3-2 Wakeup signal generation

3.5 Debug response state machine

The sleep controller must wake the processor from sleep when a debugger attempts to connect, and must not allow the processor to sleep when a debugger is connected. Otherwise, the debug session will be unsuccessful.

The *ARM CoreSight Debug Interface v5 Architecture Specification* describes power-up handshake signals that you can use to power up system components when a debugger connects. You can use these signals in the sleep controller to wake the processor.

Figure 3-3 shows the required handshaking protocol for the two signals, **CDBGPWRUPREQ** and **CDBGPWRUPACK**, that you can use to wake the processor from sleep.

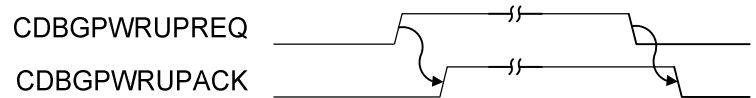


Figure 3-3 Debug power-up request and acknowledge signal timing

The **CDBGPWRUPREQ** and **CDBGPWRUPACK** signals are asynchronous to one another:

- **CDBGPWRUPREQ** is an output from the Debug Port and is in the debug port's clock domain, for example **SWCLKTCK**;
- **CDBGPWRUPACK** is an output from the sleep controller and is in the sleep controller's clock domain, **HCLK**.

You must synchronize the **CDBGPWRUPREQ** signal into the local **HCLK** domain in the sleep controller. You must also ensure that the sleep controller:

- wakes the processor from a sleep state when the **CDBGPWRUPREQ** signal is asserted;
- asserts the **CDBGPWRUPACK** signal in response once the processor has been taken out of sleep mode;
- keeps the **CDBGPWRUPACK** signal asserted until the **CDBGPWRUPREQ** signal is de-asserted;
- does not allow the processor to sleep until the handshake sequence has completed.

You can implement this functionality using a state machine with three states:

IDLE	No handshake in progress
WAKE	Wake the processor following a CDBGPWRUPREQ
ACK	Acknowledge a CDBGPWRUPREQ

Figure 3-4 shows the state transitions in this state machine.

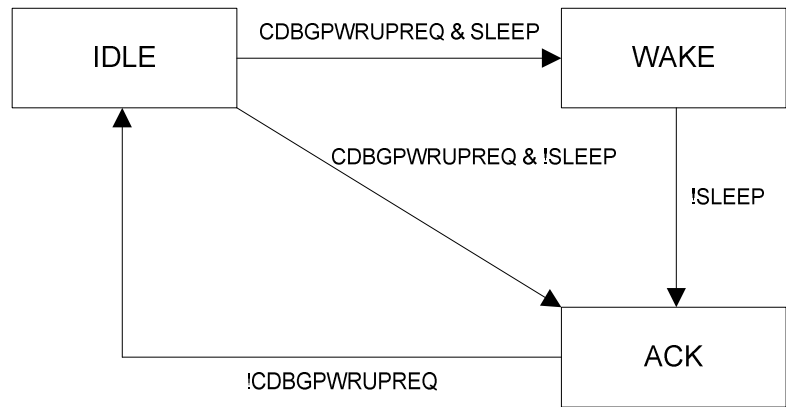


Figure 3-4 Debug state machine transitions

The state machine must assert a wakeup signal when it is in either the WAKE state or the ACK state. Asserting the signal in the ACK state ensures that the processor does not sleep before the debug port de-asserts **CDBGPWRUPREQ**.

Note

Figure 3-4 uses **CDBGPWRUPREQ** for clarity, but in the hardware implementation you must first synchronize this signal to **HCLK**.

3.6 Clock gating

The **SLEEP** output from the sleep peripheral controls a processor clock enable term in the clock gating logic. When **SLEEP** is high, the processor clock is disabled. When **SLEEP** is low, the processor clock is enabled and fully synchronous to **HCLK**.

You must ensure that your clock gating logic:

- does not cause any glitches on the processor's clock;
- does not affect the duty cycle of the processor's clock.

Figure 3-5 shows an example of how you can implement clock gating. In this scheme, a transparent latch is used to ensure that transitions of the **SLEEP** signal in the first phase of the clock do not cause glitches.

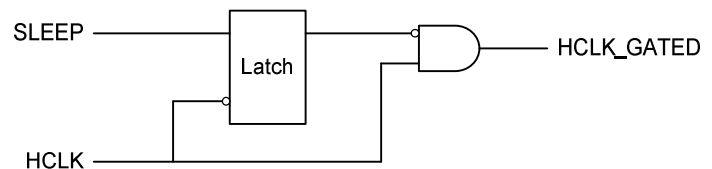


Figure 3-5 Clock gating using the SLEEP signal

3.7 Instantiating the sleep controller

When you instantiate the sleep controller in your top-level hardware design, you must connect:

- the AHB-Lite interface to your AHB-Lite interconnect;
- the **SLEEP** signal to the clock-gating logic;
- the gated **HCLK** signal to the processor's **HCLK** input.

These connections are illustrated in Figure 2-1.

If your processor configuration includes debug support, you must also connect **CDBGPWRUPREQ** and **CDBGPWRUPACK** to the CoreSight Debug Port. If your processor configuration does not include debug support, you must drive the **CDBGPWRUPREQ** input to the sleep controller low.

Note

If you have other devices that require a connection to the Debug Port's **CDBGPWRUPREQ** and **CDBGPWRUPACK** ports, you must connect:

- the Debug Port's **CDBGPWRUPREQ** output to the **CDBGPWRUPREQ** input of each device;
- the logical AND of each device's **CDBGPWRUPACK** outputs to the Debug Port's **CDBGPWRUPACK** input.

You must configure the address decoder of your AHB-Lite interconnect such that the sleep controller is only accessible from the Peripheral region of the ARM Cortex-M1 processor's address map, 0x40000000-0x5FFFFFFF. For more information about the address map in the ARM Cortex-M1 processor, refer to the *ARM Cortex-M1 Technical Reference Manual*.

Note

The Peripheral memory region in the processor's memory map defines accesses using a *Device* memory type. This means that both reads and writes to this memory region can have system side-effects, and the processor will not re-order or repeat accesses. See the *ARMv6-M Architecture Reference Manual* for more information about memory types.

4. Software description

This section provides examples of how to write software that uses the external sleep controller. It includes C code examples that you can adapt and use in your own software projects.

Some of the examples in this section use syntax that is specific to the ARM RealView Compiler Tools. If you are using a different compiler toolchain, you can adapt the examples to suit your own environment.

4.1 Macro definitions

The software running on the processor programs the sleep controller by reading and writing to its memory-mapped registers. Example 4-1 defines some peripheral addresses and provides some example C macro definitions that are useful when programming memory-mapped devices. You can use these assignments in a C header file.

```

/* Sleep controller registers */
#define SLEEP_CTRL_SLEEP      0x40000000
#define SLEEP_CTRL_SETWAKE   0x40000004
#define SLEEP_CTRL_CLRWAKE   0x40000008

/* Cortex-M1 NVIC registers */
#define NVIC_IRQ_SETENA       0xE000E100
#define NVIC_IRQ_CLRENA      0xE000E180

/* Read from a 32-bit memory-mapped register */
#define GET_UINT32(address, data_ptr) (*(data_ptr) = *((volatile unsigned int *) (address)))

/* Write to a 32-bit memory-mapped register */
#define PUT_UINT32(address, data) (*(volatile unsigned int *) (address)) = (data)

```

Example 4-1 C code definitions

Example 4-1 assumes that the sleep controller is instantiated with a base address of 0x40000000 in the ARM Cortex-M1 processor's memory map. You must modify the addresses in the example code to match your own system.

Two macros, `GET_UINT32` and `PUT_UINT32`, are defined to read from and write to 32-bit memory-mapped registers. These macros interpret the data at the given address as a volatile unsigned integer.

The `volatile` keyword instructs the C compiler that the data can change independently of the code running on the processor. ARM recommends that you use this keyword when you access peripheral registers from your C code. This is because some peripheral registers, such as timers, are expected to change between reads and the compiler should not optimize them. Registers in external peripherals might also be modified by other masters in a multi-layer AHB-Lite system.

4.2 Programming the sleep controller

The software running on the processor must configure the sleep controller to choose which interrupts will wake the processor from sleep. This must be done in addition to enabling the interrupts in the Nested Vectored Interrupt Controller (NVIC) in the ARM Cortex-M1 processor.

Note

ARM recommends that you keep the NVIC and sleep controller in a consistent state by enabling the same interrupts in each. Always change the sleep controller's configuration after changing the NVIC configuration, and only request entry into the low-power sleep mode from the processor's base priority level. This helps to maintain software compatibility with other sleep implementations.

Example 4-2 shows an example of how you can enable interrupts in the ARM Cortex-M1 processor's NVIC and program them to wake the processor from sleep. The example enables interrupts 0 to 3 in the NVIC and sleep controller.

```
/* Enable interrupts 0 to 3 in the NVIC by writing 1s to bits [3:0] in */
/* the NVIC SETPEND register, then set them to wake the processor by */
/* writing to the sleep controller's SETWAKE register. */
PUT_UINT32(NVIC_IRQ_SETENA, 0x0000000F);
PUT_UINT32(SLEEP_CTRL_SETWAKE, 0x0000000F);
```

Example 4-2 Enabling interrupts in the NVIC and sleep controller**Note**

You must define an interrupt service routine for each interrupt that you enable, and create a pointer to each interrupt service routine in the processor's vector table.

4.3 Requesting sleep entry

To request entry into the low-power sleep state, the processor can read from the sleep controller's SLEEP register.

Note

To increase compatibility with other sleep implementations, ARM recommends that you define a Wait For Interrupt macro that expands to a C function that accesses the sleep controller.

Example 4-3 shows an example `wait_for_interrupt()` macro definition that you can include in your header files. Your application code should use this macro to request entry into the low-power sleep state.

```
/* Define a generic macro alias that invokes a sleep implementation */
#define wait_for_interrupt() sleep_controller_read()
```

Example 4-3 Wait For Interrupt macro definition

Defining a macro like the one in Example 4-3 makes it easier to port software to processors that use a different Wait For Interrupt mechanism. You can re-define the macro to use the available sleep mechanism.

Example 4-4 shows how you can implement the `sleep_controller_read()` function that implements the Wait For Interrupt feature using the sleep controller that is described in this Application Note. The function uses the `__inline` operator, which provides a hint to the ARM RealView Compiler Tools that the function should be inlined if possible. This prevents the overhead of function entry and exit that is caused by register stacking.

```
/* Request entry into the low-power sleep state using the external sleep controller */
__inline void sleep_controller_read (void)
{
    int temp;                                /* temporary storage for the read access */
    GET_UINT32(SLEEP_CTRL_SLEEP, &temp); /* sleep */

    return;
}
```

Example 4-4 Sleep function definition

5. Considerations

There are a number of considerations that you must make if you use the external sleep controller that is described in this Application Note. The major considerations are described in this section.

5.1 Synthesis and layout

You must constrain your design so that the global **HCLK** signal and the gated **HCLK** signal, which is denoted **HCLK_GATED** in Figure 2-1, have matched insertion delays. This is to ensure that AHB-Lite transactions operate synchronously.

ARM also recommends that you perform a hold-time analysis of your design to guarantee correct synchronous operation between the **HCLK** and **HCLK_GATED** clock domains.

5.2 Level-sensitive and pulse interrupts

The sleep controller described in this Application Note only works with level-sensitive interrupts. Do not use peripherals that generate pulse interrupts, otherwise the interrupt will not be serviced when the processor comes out of sleep.

If you want to use peripherals that generate pulse interrupts, you must add extra functionality to the sleep controller that registers the interrupts and presents a level-sensitive interface to the processor.

5.3 Interrupt priority

The external sleep controller does not have visibility of the processor's current execution priority. ARM recommends that you only request entry into the low-power sleep state from the processor's base level of priority, otherwise an interrupt that has a lower priority than the processor's current execution priority might wake the processor from sleep.

Alternatively, you can handle the interrupt priorities in software. If you want to activate the low-power sleep mode from execution priorities that are higher than the base priority, then before reading the sleep controller's SETWAKE register you must:

- read the NVIC priority registers for each enabled interrupt to find their priority settings;
- use the sleep controller's CLRWAKE register to clear the wake-on-interrupt bits for all interrupts that have a lower priority than the current execution priority.

5.4 NVIC consistency

You must ensure that the NVIC's interrupt enable registers and the sleep controller's wake-on-interrupt registers are kept in a consistent state, based on the processor's current execution priority. Otherwise, an interrupt that is disabled in the NVIC might wake the processor from sleep mode.

5.5 Multi-layer buses

If you are using a multi-layer AHB-Lite subsystem, do not allow any master other than the ARM Cortex-M1 processor to access the sleep controller peripheral. Failure to do so may cause undesirable results.

If your system contains more than one processor that requires sleep support, you must use a separate sleep controller for each processor. This is required to ensure consistency with each processor's NVIC and to allow each processor to sleep independently.

5.6 Code compatibility

Some ARM processors, such as the ARM Cortex-M3 processor, include native power-management support and a Wait For Interrupt (WFI) instruction. If you intend to migrate your Cortex-M1 software to a Cortex-M3 system that supports the WFI instruction, you must consider power-management support.

When you port the software to another processor that supports WFI, you must:

- remove accesses to the sleep controller's initialization registers;
- replace accesses to the sleep controller's SLEEP register with the WFI instruction.

You can simplify this transition by keeping the NVIC and sleep controller in a consistent state and defining general sleep macros, as recommended in this Application Note.

6. References

This Application Note refers to the following ARM documentation:

- *ARM Cortex-M1 Technical Reference Manual* (ARM DDI 0413)
- *ARMv6-M Architecture Reference Manual* (ARM DDI 0419)
- *AMBA3 AHB-Lite Protocol Specification* (ARM IHI 0033)
- *ARM Debug Interface v5 Architecture Specification* (ARM IHI 0031)

You can also refer to the ARM InfoCenter at <http://infocenter.arm.com> for access to all Non-Confidential ARM documentation.