Reducing Power Consumption in the RC323xx Series

Application Note AN-290

Notes

By Upendra Kulkarni

Revision History

February 7, 2002: Initial publication.

October 18, 2002: Changed device designation to RC3233x which includes 3 devices: RC32334, RC32333, and RC32332.

April 2, 2003: Changed device designation to RC323xx to include the RC32355, RC32351, and RC32365 devices.

Introduction

The RC32334, RC32333, RC32332, RC32355, RC32351, and RC32365 integrated processors are based on the RISCore32300™ CPU core. This CPU core offers several methods of dramatically reducing the power consumption of the processor core itself and the SDRAM memory subsystem. One very effective method is the execution of the "WAIT" instruction.

The aim of this application note is to outline how the "WAIT" instruction can be used, provide some guidelines as to the steps a programmer should take as preparatory measures prior to issuing the "WAIT" instruction and how the low-power mode should be exited and normal mode operation be restored.

The WAIT Instruction

The "WAIT" instruction is a CP0 instruction. It is designed to signal the rest of the chip that execution and clocking should be halted. This helps to reduce system power consumption during idle periods.

When the "WAIT" instruction finishes the "W" pipe stage, if the address and data buses ("AD" bus) are detected to be idle, the chip is allowed to enter the "standby mode". In the standby mode, internal clocks will be shut down and the pipeline will be frozen. The PLL, internal timer, some of the input pin clocks (Int[5:0], NMI, Reset and ColdReset) will continue to run. Once in standby mode, any interrupt, including the internally generated timer interrupt, will cause the CPU to exit standby mode.

Note that if the "AD" bus is detected to be busy at the time when the "WAIT" instruction finishes execution, then the instruction is simply treated as a NOP. Therefore, in order to successfully enter the standby mode, a few basic things must happen. Firstly, the "WAIT" instruction must execute through instruction cache. Secondly, it should execute following several NOPs to guarantee the bus is idle when the "WAIT" instruction completes. Furthermore, IDT recommends that the code should ensure that all DMA activities have come to an end before embarking upon the standby entry process.

In addition to these basic considerations, care must be taken regarding several issues, in order to accomplish successful entry into and exit from the standby mode. For example, the caches should be in a known clean state before attempting to introduce the "WAIT" instructions into the I-cache. Additionally, the SDRAM subsystem, if any, must be brought to a known and acceptable state before entering the standby mode. The system controller module present on the RC323xx integrated processors must be conditioned and brought to a state from where it can be restored to operate in normal mode in a predictable and acceptable manner when the standby mode is exited. It is also important to design a proper exit strategy for bringing the CPU core back to the normal mode of operation. This strategy must include an appropriate interrupt handler subroutine located at the correct vector, so that the desired interrupt can re-awaken the system out of standby mode. Desirable interrupts must be enabled and others should be disabled before entering the standby mode. Failure to design this step properly will result in a scenario in which an NMI-generated interrupt will be the only way to get out of the standby mode. If the system includes a SDRAM

Notes

subsystem, code to initialize it must be executed as part of the interrupt handler before control is passed to normal code execution which will typically occur upon code being loaded into the SDRAM. Another issue to be addressed is that two instructions immediately following the "WAIT" instruction are always executed before a jump to the interrupt vector is taken when the system exits standby mode. Therefore, care must be taken to include only useful or NOP instructions in these two locations.

Sequence of Operation

Each application will have its own reasons for going into the standby mode at specific points in the life of the execution of its code. Therefore, it cannot be reasonably expected to come up with a generalized algorithm for addressing the task of successful implementation of the standby mode in all systems. However, a few steps, which can be reasonably assumed to be mandatory, are listed here.

- **Write and install the interrupt handler for the interrupt that will be bringing the standby mode to an end**
- **Initialize CP0 Status Register to have desirable interrupts enabled**
- **Initialize the instruction cache and possibly the data cache as well**
- **Pre-fill and lock the instruction cache with NOPs, "WAIT" instruction, followed by more NOPs.**
- **Suspend SDRAM by removing the CKE signal**
- **Sync and Flush write buffer**
- **Execute WAIT instruction(s) by jumping to the location in instruction cache set up above.**
- **Wait for an acceptable interrupt to occur so as to enable exiting the standby mode.**

Include code inside the interrupt handler, responsible for returning the system from standby mode, to unlock the instruction cache, initialize the system controller, the SDRAM subsystem, and other subsystems which are in either unknown states or known but undesirable states.

Actual Power Savings

IDT has conducted experiments on IDT79S334A and IDT79S332 evaluation boards with the aim of obtaining estimates of power consumption savings on the RC323xx processors, while in standby mode. Savings of between 42% and 46% were observed under various combinations of operating frequencies of the processor, the PCI bus, and the memory sub-system. As stated earlier, each application will be unique in terms of the power saving opportunities it may offer. Therefore, the numbers presented here ought to be used merely as simplistic guidelines.

Code Example

/* Algorithm:

1. Jump past exception handler memory space (bfc0_0380)
2. Initialize CP0 registers. Enable interrupts.
3. Setup memory controller channel 0, 1.
4. Turn off SDRAM CLK (output_clk).
5. Initialize I-cache.
6. Run test: Assume DMA is disabled.
7. Pre-fill and lock I-cache.
8. Sync and Flush write buffer.
9. Execute Wait instruction making sure that it is the 0x0, 0x1, or 0x2 instruction in a cache line and that the cache line and the subsequent cache line are cache hits.
10. Stall and wait for interrupt handler
12. Execute interrupt handler.

Notes

13. Unlock i-cache.

14. Exit

*/

/*

Notes:

1. Interrupts must be enabled for them to exit out of wait instruction stall. NMI is always enabled and thus can always exit out.
2. DMA activity will not block wait from stalling, however, once wait is in the register write-back pipeline stage, the internal BusGnt signal will be stuck at its current level (active or inactive).
3. On the RC32300 core, the I-cache does not require that it be pre-filled. In other words, instruction streaming does not occur. So, a cache miss on the current line and/or on the subsequent cache line will still invoke a WAIT stall, as long as the WAIT instruction is in word position 0x0, 0x1, or 0x2, but not 0x3 in the cache line.
4. The 2 instructions after the wait instruction get executed after the interrupt occurs, before the interrupt vector is taken.

*/

```
        .text
        .set noreorder

    bootcode:
    ; jump past exception handler
    la      r2, StartOfCode
    jr      r2
    nop     # branch delay slot is always executed

    ; exception handler
    ; put your exception handler code here
    ; make sure you make a clean orderly exit out of the standby mode through this code
    ; .align is used here as an example of how it works on Solaris native GNU compiler,
    : use appropriate methods to place this code at the reset vectors.

    align   9           # align to 0xbfc00200
    nop
    .align  8           # align to 0xbfc00300
    nop
    .align  7           # align to 0xbfc00380
    b       unlock_icache
    nop
```

Notes

```
; start of boot code
StartOfCode:
        li      r2,PCACHE_I | PCACHE_D | WR_BACK
        mtc0    r2,C0_CONFIG# only lower bits are writable
        mtc0    r0,C0_INX
        mtc0    r0,C0_CAUSE
        mtc0    r0,C0_COUNT
        mtc0    r0,C0_COMPARE
        mtc0    r0,C0_EPC
        mtc0    r0,C0_TAGLO

        li      r2,SR_BEV|SR_CU1|SR_DE|SR_IMASK|SR_IE
                        # exception in bfc00200 area
                        # no cache parity checking
                        # all interrupt masks enabled
                        # global enable interrupts
        mtc0    r2,C0_SR

INIT_TEST:
; use r2, r3 as index-able pointers for everything else
        lui     r2, 0xb800      # physical addr 1800xxxx
        lui     r3, 0xbfc0      # physical addr 1fc0xxxx

; setup memory controller
        li      r4, 0x00002863      # IP Write to MemReg00
                                    # mem_type, 245, wrprotoff,32bit,
                                        # 3WrWS,3RdWS
        sw      r4, 0x0200(r2)      # bank0

; Setup SDRAM Control Reg by turning SDRAM CLK (output_clk) off (bit14).
        li r4, ((0x5b7f00ff) + (0x80000000 * (1)))
        sw      r4, 0x300(r2)
        lw      r0, 0x300(r2)

; initialize and prefill cache while locking it into cache
; initialize and prefill cache
; insert code for I-cache flush here
; icache lock setup, part 1 of 2
        mfc0    r4, C0_SR
```

Notes

```
        or      r4, r4, 0x00080000# sr_set_il
        mtc0    r4, C0_SR
        nop
        nop
        nop

; lock these lines using cache instruction here

; icache lock setup, part 2 of 2
        mfc0    r4, C0_SR
        and     r4, r4, ~0x00080000
        mtc0    r4, C0_SR
        nop
        nop
        nop
; lock these lines…

; jump to cached space
        la      r4, Subtest1
        and     r4, r4, 0x1fffffff
        or      r4, r4, 0x90000000
        jr      r4
        nop

      .align  4
Subtest1:
      ; flush write buffer
        sync                    # to make sure previous store
        lw      r0,0x200(r2)    # gets complete.
        nop
        nop

; perform wait instruction
; note: the 2 instructions after the wait will execute after the interrupt is taken!
        .align  4

        wait
        nop     # executed after interrupt
        nop     # executed after interrupt
```

Notes

```
        nop

; make sure next cache line doesn't create bus activity
        nop
        nop
        nop
        nop

; shouldn't get here unless wait doesn't take
        la      r4, Fail
        jr      r4
        nop
        nop

; interrupt handler jumps here
; unlock locked icache locations

unlock_icache:
        li      r4, 0x80000000
        mtc0    r0, C0_TAGLO        # set TagLo CP0 Reg 28 to 0
        nop
        nop
        nop

# Prefill i-cache space
.space 0x2000
```

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.