Application Note

Sitara AM6442 RTI 看门狗调试

Rio Chan

摘要

本应用手册提供了有关如何在 TI Sitara™ AM6442 平台上使用 RTI 制作软件看门狗的信息。此技术仅适用于 SDK 8.4 和较旧的 8.x SDK 版本。SDK8.5 和更高的 SDK 版本不需要此功能。

内容

插图清单

表格清单

商标

1 引言

本文档展示了 U-Boot 软件看门狗的实现。所使用的技术是实时中断 (RTI)。

一般来说，有两种类型的看门狗：

- 外部看门狗 (硬件)
- 在内部看门狗片上系统 (SoC) 内，称为软件看门狗

RTI 是 AM6442 器件中包含的软件看门狗。这是一个数字窗口看门狗 (DDWD)。

在 MCU 域和主域中分配了七个 RTI 模块。表 1-1 来自 *AM64x/AM243x 技术参考手册* 中的 *RTI 概述* 部分。

表 1-1. 跨器件域的 RTI 分配

实例	域	
	MCU	主
MCU_RTI0	✓	-
RTI0	-	✓
RTI1	-	✓
RTI8	-	✓
RTI9	-	✓
RTI10	-	✓
RTI11	-	✓

MCU 域中的实例：

- MCU_RTI0 专用于锁步中的 MCU 群集 (MCU_M4FSS0)。解锁后，它用作 MCU 域 (MCU_M4FSS0_CORE0) 中第一个 M4F CPU 内核的窗口式看门狗。

MAIN 域中的实例：

这些实例旨在用作 CPU 内核的数字窗口式看门狗，并与以下项相关联：

- RTI0 专用于 A53 集群 (A53SS0_CORE0) 中的第一个 A53 CPU 内核
- RTI1 专用于 A53 集群 (A53SS0_CORE1) 中的第二个 A53 CPU 内核
- RTI8 专用于主域 (R5FSS0_CORE0) 中的第一个 R5F CPU 内核
- RTI9 专用于主域 (R5FSS0_CORE1) 中的第二个 R5F CPU 内核
- RTI10 专用于主域 (R5FSS1_CORE0) 中的第三个 R5F CPU 内核
- RTI11 专用于主域 (R5FSS1_CORE1) 中的第四个 R5F CPU 内核

U-Boot 代码用于控制 RTI0，因为 U-Boot 主要在 A53 core0 上运行。

本应用手册提供了以下信息：

- 如何从 U-Boot 提示符处开始运行测试命令
- 如何将这些命令转换为 C 代码
- 它在运行时是什么样子

有时，U-Boot 会卡住，开发人员需要在没有冷启动的情况下重置系统；届时需要看门狗。例如，根据 *AM64x/AM243x 处理器芯片版本 1.0、2.0 勘误表*，如果客户需要在 U-Boot 上启动 ETH，则 ETH 会卡住。发生这种情况时，RTI 会触发看门狗复位，然后 ETH 可以再次恢复。有关更多信息，请参阅本文档中随附的补丁，并参阅“eth_initialize()”函数。

本文档使用 AM64x SDK8.2 版本 (版本：08.02.00.17，发布日期：2022 年 4 月 26 日)。

2 所需的硬件和软件

- 所需硬件项：
  - TMDS64GPEVM

  您需要 12V/5A 直流适配器。
- 所需软件项：
  - SDK8.0 / 8.1 /8.2 /8.3 /8.4

备注

SDK v8.5 及更高版本不需要本应用手册中的技术。

3 AM6442 RTI 看门狗模块

根据 *AM64x/AM243x 技术参考手册* 中的 *RTI 特性* 部分，这些关键点的 RTI 如下：

- 这是一个软件复位
- AM642x 上有七个 RTI 模块
- 它是一个可配置的窗口计时器看门狗
- 只要在这个时间窗口之外尝试对看门狗进行维护，或者在这个时间窗口内未能对看门狗进行维护，都会导致看门狗向 CPU 生成一个复位或一个不可屏蔽中断。

3.1 RTI 如何在 U-Boot 中工作？

通过读取特定于器件的 TRM，U-Boot 提示符中有六条命令可以让 RTI 看门狗工作：

- **mw.l 0x43009008 0x68EF3490 1**
  （CTRL_MMR0，请参阅 *AM64x/AM243x 技术参考手册* ）
- **mw.l 0x4300900C 0xD172BC5A 1**
  （CTRL_MMR0，请参阅 TRM ）
- **mw.l 0x43008380 0x3 1**
  （CTRL_MMR0，请参阅 TRM ）
- **mw.l 0xe0000a4 0xa 1**
  （RTI0，请参阅 TRM ）
- **mw.l 0xe000094 0x23 1**
  （RTI0，请参阅 TRM ）
- **mw.l 0xe000090 0xA98559DA 1**

备注

用红色标记的值 0x23 表示可以为看门狗过期值配置的值。

流程如下：

1. 前四个命令用于解锁存储器映射寄存器 (MMR0)，以 MMR0 为例。根据表 3-1，您需要解锁 RTI0 的“MMR0 寄存器”。
2. 第 1 条和第 2 条命令用于 CTRL_MMR0 锁定或解锁。有关 CTRL_MMR0 的更多详细信息，请参阅 *AM64x/AM243x 技术参考手册* 内 *Kick 保护寄存器* 部分中的*分区解锁值* 表。

   ```
   mw.l 0x43009008 0x68EF3490 1
   mw.l 0x4300900C 0xD172BC5A 1
   ```
3. 第 3 个命令是设置 CTRLMMR_WWD0_CLKSEL 寄存器，其中 WWD0 是窗口看门狗 0 = RTI0。根据默认设置，值 0x3 是 32K 系统时钟源。有关更多信息，请参阅 *AM64x/AM243x 技术参考手册*。

   ```
   mw.l 0x43008380 0x3 1
   ```

4. 最后 3 个 (第 4 个、第 5 个、第 6 个) 命令用于配置 RTI0。
   a. 第 4 个命令是设置 RTI_WWDRXNCTR“数字窗口式看门狗响应”。该寄存器的值只能是 0x5 或 0xa。0xa 用于 RTI 生成不可屏蔽中断。

      ```
      mw.l 0xe0000a4 0xa 1
      ```

   b. 第 5 个命令用于设置预加载值，即计时窗口。这可以被视为“看门狗超时过期值”。

      ```
      mw.l 0xe000094 0x23 1
      ```

   c. 第 6 个命令用于启用 RTI 看门狗：
      i. 值 0x5312ACED 用于禁用 RTI。
      ii. 值 0xA98559DA 用于启用 RTI。

      ```
      mw.l 0xe000090 0xA98559DA 1
      ```

表 3-1. RTI 时钟和复位

模块实例	模块时钟输入	源时钟信号	吸电流	说明
RTI0	RTI0_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI0 接口时钟
	RTI0_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI0 功能时钟。有关 RTICLK0 多路复用器中时钟多路复用的更多信息，请参阅*控制模块 (CTRL_MMR)* 中的 CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL。
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		
RTI1	RTI1_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI1 接口时钟
	RTI1_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI1 功能时钟。有关 RTICLK1 多路复用器中时钟多路复用的更多信息，请参阅*控制模块 (CTRL_MMR)* 中的 CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL。
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		
RTI8	RTI8_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI8 接口时钟
	RTI8_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI8 功能时钟。有关 RTICLK8 多路复用器中时钟多路复用的更多信息，请参阅*控制模块 (CTRL_MMR)* 中的 CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL。
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		

4 关于 U-Boot 中的这六个命令

在 U-Boot 中开始实现六条命令之前，您需要知道安装 SDK 时在哪里可以找到需要修改的项。

U-Boot 路径：

/opt/ti-processor-sdk-linux-rt-am64xx-evm-08.02.00.14/board-support/U-Boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8

U-Boot dts 位于以下位置：

/opt/ti-processor-sdk-linux-rt-am64xx-evm-08.02.00.14/board-support/U-Boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/arch/arm/dts/**k3-am642-evm.dts**

U-Boot config 位于以下位置：

/opt/ti-processor-sdk-linux-rt-am64xx-evm-08.02.00.14/board-support/U-Boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/configs/**am64x_evm_r5_defconfig**

检查 ESM 模块是否在以下 U-Boot config 中默认设置为“y”：

CONFIG_ESM_K3=y

在 *AM64x/AM243x 技术参考手册*的 *MCU 域支持复位* 部分中，支持 MCU 域复位，如下所述：

本应用手册中使用 RTI 看门狗复位，因此选择 MCU ESM 错误复位。确保将这项设置为启用：CONFIG_ESM_K3。

- 上电复位
  - MCU_PORz 器件引脚
- 热复位
  - MCU_RESETz 器件引脚
  - MCU 域软件热复位
  - MCU 域 ESM 错误复位 ←
  - DMSC-L 冷复位
- 本地模块复位
  - MCU 域 LPSC 模块本地复位

如何从 U-Boot 提示符处开始运行测试命令？

1. 打开 EVM 的电源。
2. 连续按空格键，直到控制台显示 U-Boot 提示符。
3. 输入以下 6 个命令：

```
mw.l 0x43009008 0x68EF3490 1
mw.l 0x4300900C 0xD172BC5A 1
mw.l 0x43008380 0x3 1
mw.l 0xe0000a4 0xa 1
mw.l 0xe000094 0x23 1
mw.l 0xe000090 0xA98559DA 1
```

4. 通过设置 0x23 的值，您将看到 EVM 在大约 12 秒后重新启动。

图 4-1 展示了控制台日志的结果。输入 6 个命令后，系统将再次重启。

```
U-Boot SPL 2021.01-g44a87e3ab8 (Apr 10 2022 - 19:39:06 +0000)
SYSFW ABI: 3.1 (firmware rev 0x0016 '22.1.1--v2022.01 (Terrific Llam')
Trying to boot from MMC2


U-Boot 2021.01-g44a87e3ab8 (Apr 10 2022 - 19:39:06 +0000)

SoC:   AM64X SR1.0
Model: Texas Instruments AM642 EVM
Board: AM64-GPEVM rev E2
DRAM:  2 GiB
NAND:  0 MiB
MMC:   mmc@fa10000: 0, mmc@fa00000: 1
In:    serial@2800000
Out:   serial@2800000
Err:   serial@2800000
Net:   eth0: ethernet@8000000port@1
Hit any key to stop autoboot:  0
=> mw.l 0x43009008 0x68EF3490 1
=> mw.l 0x4300900C 0xD172BC5A 1
=> mw.l 0x43008380 0x3 1
=> mw.l 0xe0000a4 0xa 1
=> mw.l 0xe000094 0x23 1
=> mw.l 0xe000090 0xA98559DA 1
=>
=>
U-Boot SPL 2021.01-g44a87e3ab8 (Apr 10 2022 - 19:40:27 +0000)
SYSFW ABI: 3.1 (firmware rev 0x0016 '22.1.1--v2022.01 (Terrific Llam')
SPL initial stack usage: 13392 bytes
Trying to boot from MMC2
Starting ATF on ARM64 core...
```

图 4-1. 系统在发出六个命令后重启

5 如何将这些命令转换为 C 代码？

本节分为两个部分：

- 为这六个命令创建一个 API
- 将此 API 放入 U-Boot 中的目标 C 文件。

5.1 本应用手册的完整 RTI 补丁

这是整个补丁。它会影响以下文件：

- common/autoboot.c
- common/board_r.c

```
diff --git a/common/autoboot.c b/common/autoboot.c
index e628baffb8..fcfed76438 100644
--- a/common/autoboot.c
+++ b/common/autoboot.c
@@ -4,6 +4,13 @@
  * Wolfgang Denk, DENX Software Engineering, wd@denx.de.
  */

+#define TI_RTI_WATCHDOG_PATCH
+
+#ifdef TI_RTI_WATCHDOG_PATCH
+#include <asm/io.h>
+#include <asm/arch/hardware.h>
+#endif
+
 #include <common.h>
 #include <autoboot.h>
 #include <bootretry.h>
@@ -246,13 +253,55 @@ static int abortboot_key_sequence(int bootdelay)
     return abort;
 }

+#ifdef TI_RTI_WATCHDOG_PATCH
+static CTRL_and_RTI_Clock_Counter_Enabled(void)
+{
+    int checkreg = 0;
+        /* TI: The below 2 lines are to unlock the MMR register. */
+        /* TI: Lock2/Kick0, Proxy physical 0. = 9008, to write the register 0x68EF3490 is
"unlock". */
+        writel(0x68EF3490,0x43009008);
+        /* TI: Lock2/Kick1, Proxy physical 0. = 900C, to write the register 0xD172BC5A is
"unlock". */
+        writel(0xD172BC5A,0x4300900C);
+        /* TI: Check the clk source of WWD0 before writing. */
+        checkreg = readl(0x43008380);
+        printf("TI : RTI check before writing 0x43008380 clk_src reg == 0x%x\n", checkreg);
+        /* TI: Set the clk src as 32k, IE: WWD0 Clock select, Proxy physical 0, the register is
0x43008380. Write value 0x3 is to set the clk src as 32KHz.) */
+        writel(0x3,0x43008380);
+        /* TI: double check the clk source of WWD0 value again. */
+        checkreg = readl(0x43008380);
+        printf("TI : RTI check after writing 0x43008380 clk_src reg == 0x%x\n", checkreg);
+        /* TI: The RTI_WWDRXNCTR is 0xE0000a4, Digital Windowed Watchdog Reaction. */
+        /* TI: Ah = The windowed watchdog will generate a non-maskable interrupt to the CPU if the
watchdog is serviced outside the time window
+        defined by the configuration, or if the watchdog is not serviced at all. Writing any other
value will cause a system reset if the watchdog is
+        serviced outside the time window defined by the configuration, or if the watchdog is not
serviced at all. */
+        writel(0xa,0xe0000a4);
+        /* TI: double Check the RTI_WWDRXNCTR 0xE0000a4 value. */
+        checkreg = readl(0xE0000a4);
+        printf("TI : RTI check 0xe0000a4 reg == 0x%x\n", checkreg);
+        /* TI: Check the RTI_DWDPRL 0xE000094 value. */
+        checkreg = readl(0xE000094);
+        printf("TI : RTI check before writing 0xe000094 timing window reg == 0x%x\n", checkreg);
+        /* TI: Write the timeing window to the register RTI_DWDPRL, 0x9 is about 4~5 seonds, this
just fit the uboot complete,
+        in this example, we expect the WDT takes effect after 4~5 seconds.) */
+        writel(0x9  ,0xE000094);
+        checkreg = readl(0xE000094);
```

```
+        printf("TI : RTI check after writing 0xe000094 timing window reg == 0x%x\n", checkreg);
+        /* TI: WDT CLock coutner register is RTI_DWDCTRL, set this value: A98559DA to let the
Watchdog counter enabled. */
+        writel(0xA98559DA,0xE000090); /* TI: Clock counter enabled. */
+        udelay(4000); /* TI: Adding this delay for letting this function to take effect. */
+}
+
+#endif
+
 static int abortboot_single_key(int bootdelay)
 {
     int abort = 0;
     unsigned long ts;
-
+#ifdef TI_RTI_WATCHDOG_PATCH
+    /* TI: Set the boot delay as 0 to accelerate the next system reset faster, user can adjust
this value by himself. */
+    bootdelay = 0;
+#endif
     printf("Hit any key to stop autoboot: %2d ", bootdelay);
-
     /*
      * Check if key already pressed
      */
@@ -262,6 +311,10 @@ static int abortboot_single_key(int bootdelay)
         abort = 1;    /* don't auto boot    */
     }

+#ifdef TI_RTI_WATCHDOG_PATCH
+    printf("TI : abort:0x%x\n", abort);
+#endif
+
     while ((bootdelay > 0) && (!abort)) {
         --bootdelay;
         /* delay 1000 ms */
@@ -283,6 +336,21 @@ static int abortboot_single_key(int bootdelay)
         printf("\b\b\b%2d ", bootdelay);
     }

+#ifdef TI_RTI_WATCHDOG_PATCH
+    if(abort !=1)
+    {
+        int temp32 = 0;
+        temp32 = readl(0x04518170);
+        printf("TI : This bit should be zero, check 0x04518170 reg == 0x%x\n", temp32);
+        printf("TI : in autoboot.c Enable RTI Clock.\n");
+        CTRL_and_RTI_Clock_Counter_Enabled();
+        /* TI: Disable the eth_initialize initialized in common/board_r.c, put to here. */
+        /* do the CTRL_and_RTI_Clock_Counter_Enabled first, then, do the eth_initialize in order
to prevent the ETH stuck. */
+        puts("Net:   ");
+        eth_initialize();
+    }
+#endif
+
     putc('\n');

     return abort;
@@ -386,3 +454,4 @@ void autoboot_command(const char *s)
             run_command_list(s, -1, 0);
     }
 }
+
diff --git a/common/board_r.c b/common/board_r.c
index 29dd7d26d9..b4e249056d 100644
--- a/common/board_r.c
+++ b/common/board_r.c
@@ -622,11 +622,18 @@ static int initr_bbmii(void)
 }
 #endif

+#define TI_RTI_WATCHDOG_PATCH
+
 #ifdef CONFIG_CMD_NET
 static int initr_net(void)
 {
+#ifdef TI_RTI_WATCHDOG_PATCH
+    /* TI: moved this eth_initialize to autoboot.c for RTI_WATCHDOG_PATCH. */
+#else
```

```
      puts("Net:   ");
      eth_initialize();
 +#endif
 +
  #if defined(CONFIG_RESET_PHY_R)
      debug("Reset Ethernet PHY\n");
      reset_phy();
```

6 参考文献

- *AM64x/AM243x GP EVM 用户指南*
- *AM64x/AM243x 技术参考手册*
- AM64 SDK 文档：https://software-dl.ti.com/processor-sdk-linux-rt/esd/AM64X/08_02_00_17/exports/docs/devices/AM64X/Overview.html

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265