Application Note

Sitara AM6442 RTI Watchdog Debug

Rio Chan

ABSTRACT

This application note provides information on how to make a software Watchdog by using RTI on the TI Sitara™ AM6442 platform. This technique is only applicable to SDK 8.4 and older 8.x SDK releases. This is not needed for SDK8.5 and newer SDK releases.

Table of Contents

1 Introduction....2
2 Required Hardware and Software....3
3 AM6442 RTI Watchdog Modules....3
3.1 How the RTI Works in the U-Boot?....3
4 About Those Six Commands in the U-Boot....4
5 How to Turn These Commands Into C Code?....6
5.1 Entire RTI Patch for This Application Note....6
6 References....8

List of Figures

Figure 4-1. System Reboot After Issuing Six Commands....5

List of Tables

Table 1-1. RTI Allocation Across Device Domains....2
Table 3-1. RTI Clocks and Resets....4

Trademarks

Sitara™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

This document demonstrates the U-Boot software watchdog implementation. The technique being used is the Real Time Interrupt (RTI).

In general, there are two types of watchdogs:

- External watchdogs (hardware)
- Internal watchdogs System-on-Chip (SoC) inside, call it software watchdog

RTI is a software watchdog that is included in the AM6442 device. This a Digital Window Watchdog (DDWD).

There are seven RTI modules allocated in the MCU domain and Main domain. Table 1-1 is from the *RTI Overview* section in the *AM64x/AM243x Technical Reference Manual*.

Table 1-1. RTI Allocation Across Device Domains

Instance	Domain	
	MCU	Main
MCU_RTI0	✓	-
RTI0	-	✓
RTI1	-	✓
RTI8	-	✓
RTI9	-	✓
RTI10	-	✓
RTI11	-	✓

Instances in the MCU domain:

- MCU_RTI0 is dedicated to the MCU cluster (MCU_M4FSS0) in the lockstep. When unlocked, it serves as a Windowed Watchdog for the first M4F CPU core in the MCU domain (MCU_M4FSS0_CORE0).

Instances in the MAIN domain:

These are intended to function as a Digital Windowed Watchdog for the CPU core and are associated with the following items:

- RTI0 is dedicated to the first A53 CPU core in the A53 cluster (A53SS0_CORE0)
- RTI1 is dedicated to the second A53 CPU core in the A53 cluster (A53SS0_CORE1)
- RTI8 is dedicated to the first R5F CPU core in the Main domain (R5FSS0_CORE0)
- RTI9 is dedicated to the second R5F CPU core in the Main domain (R5FSS0_CORE1)
- RTI10 is dedicated to the third R5F CPU core in the Main domain (R5FSS1_CORE0)
- RTI11 is dedicated to the fourth R5F CPU core in the Main domain (R5FSS1_CORE1)

The U-Boot code is used to control the RTI0 due to fact that the U-Boot is mainly running on the A53 core0.

This application note provides information about:

- How to start the test commands running from U-Boot prompt
- How to make these commands turning into C code
- What it looks like when running

Sometimes, the U-Boot gets stuck and the developer needs to reset the system without cold boot; the watchdog is needed at that time. For example, according to the *AM64x/AM243x Processor Silicon Revision 1.0, 2.0 Errata*, if the customer needs to have the ETH boot on the U-Boot, then the ETH gets stuck. When this happens, the RTI triggers the watchdog reset, then the ETH can be recovered again. For more information, see the patch attached in this document, and see the “eth_initialize()” function.

The AM64x SDK8.2 release is used in this document (Version: 08.02.00.17 Release date: 26 Apr 2022).

2 Required Hardware and Software

- Hardware required item:
  - TMDS64GPEVM

  You need 12V/5A DC adapter.
- Software required item:
  - SDK8.0 / 8.1 /8.2 /8.3 /8.4

Note

The technology in this application note is not needed for SDK versions v8.5 and later.

3 AM6442 RTI Watchdog Modules

According to the *RTI Features* section in the *AM64x/AM243x Technical Reference Manual*, the RTI for those key points are as follows:

- It is a software reset
- There are seven RTI modules on AM642x
- It is a configurable window timer watchdog
- Any attempt to service the watchdog outside this time window, or a failure to service the watchdog in this time window, causes the watchdog to generate either a reset or a non-maskable interrupt to the CPU.

3.1 How the RTI Works in the U-Boot?

By reading the device-specific TRM, here are six commands in the U-Boot prompt to get the RTI watchdog working:

- **mw.l 0x43009008 0x68EF3490 1**
  (CTRL_MMR0, see *AM64x/AM243x Technical Reference Manual*)
- **mw.l 0x4300900C 0xD172BC5A 1**
  (CTRL_MMR0, see TRM)
- **mw.l 0x43008380 0x3 1**
  (CTRL_MMR0, see TRM)
- **mw.l 0xe0000a4 0xa 1**
  (RTI0, see TRM)
- **mw.l 0xe000094 0x23 1**
  (RTI0, see TRM)
- **mw.l 0xe000090 0xA98559DA 1**

Note

The value 0x23 marked in red indicates the value you can configure for the watchdog expire value.

The flow is as follows:

1. The first four commands are unlocking the Memory Mapped Register (MMR0) - MMR0 is used as an example. According to Table 3-1, you need to unlock the "MMR0 registers" for the RTI0.
2. The 1st and 2nd commands are used for CTRL_MMR0 lock or unlock. For more details on CTRL_MMR0, see the *Partition Unlock Values* table in the *Kick Protection Registers* section in the *AM64x/AM243x Technical Reference Manual*.

   ```
   mw.l 0x43009008 0x68EF3490 1
   mw.l 0x4300900C 0xD172BC5A 1
   ```
3. The 3rd cmd is to set the CTRLMMR_WWD0_CLKSEL Register, in which WWD0 is the window watchdog 0 = RTI0. The Value 0x3 is the 32K system clock source, according to the default setting. For more information, see the *AM64x/AM243x Technical Reference Manual*.

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

4. The last 3 (4th, 5th, 6th) commands are used for configuring the RTI0.
   a. The 4th cmd is to set the RTI_WWDRXNCTR, Digital Windowed Watchdog Reaction. The value of this reg can only be either 0x5 or 0xa. 0xa was used for RTI generating non-maskable interrupt.

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

   b. The 5th cmd is to set the pre-load value, which is the timing window. This can be treated as an "Watchdog timeout expire value".

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

   c. The 6th cmd is to enable the RTI watchdog:
      i. Value 0x5312ACED is to disable the RTI.
      ii. Value 0xA98559DA is to enable the RTI.

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

Table 3-1. RTI Clocks and Resets

Module Instance	Module Clock Input	Source Clock Signal	Source	Description
RTI0	RTI0_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI0 Interface Clock
	RTI0_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI0 Functional Clock. For more information about clock multiplexing in RTICLK0 MUX, see CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL in *Control Module (CTRL_MMR)*.
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		
RTI1	RTI1_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI1 Interface Clock
	RTI1_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI1 Functional Clock. For more information about clock multiplexing in RTICLK1 MUX, see CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL in *Control Module (CTRL_MMR)*.
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		
RTI8	RTI8_ICLK	MAIN_SYSCLK0/4	PLLCTRL0	RTI8 Interface Clock
	RTI8_FCLK	MCU_HFOSC0_CLKOUT	MCU_HFOSC0	RTI8 Functional Clock. For more information about clock multiplexing in RTICLK8 MUX, see CTRLMMR_WWD0_CLKSEL [1:0] CLK_SEL in *Control Module (CTRL_MMR)*.
		MCU_HFOSC0_CLKOUT_32K		
		MCU_CLK_12M_RC	MCU_RC_OSC_12M	
		CLK_32K		

4 About Those Six Commands in the U-Boot

Before starting the implementation for six commands in the U-Boot, you need to know where the items that need to be modified can be found within the SDK installation.

The **U-Boot path:**

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

U-Boot dts is here:

/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 is here:

/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**

Check that the ESM module is set by default to "y" in the following U-Boot config:

CONFIG_ESM_K3=y

In the *MCU Domain Supported Reset* section in the *AM64x/AM243x Technical Reference Manual*, the MCU domain reset are supported as described below:

In this application note, the RTI watchdog reset is used, so choose the MCU ESM error reset. Make sure to set this enabled: CONFIG_ESM_K3.

- Power-On-Resets
  - MCU_PORz device pin
- Warm Resets
  - MCU_RESETz device pin
  - MCU domain software warm reset
  - MCU domain ESM error reset ←
  - DMSC-L cold reset
- Local Module Resets
  - MCU domain LPSC module local reset

How to start the test commands running from U-Boot prompt?

1. Power on the EVM.
2. Press “Space” key continuously until the console shows the U-Boot prompt.
3. Input those 6 cmds:

```
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. By setting the value of 0x23, you will see the EVM reboot after about approximately 12 seconds.

Figure 4-1 shows the results of the console log. After inputting the 6 cmd, the system will reboot again.

```
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...
```

Figure 4-1. System Reboot After Issuing Six Commands

5 How to Turn These Commands Into C Code?

This section is separated into two parts:

- Making an API for those six commands
- Place this API to the target C file in the U-Boot.

5.1 Entire RTI Patch for This Application Note

This is the entire patch. It affects the following files:

- 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 References

- *AM64x/AM243x GP EVM User's Guide*
- *AM64x/AM243x Technical Reference Manual*
- AM64 SDK documentation: https://software-dl.ti.com/processor-sdk-linux-rt/esd/AM64X/08_02_00_17/exports/docs/devices/AM64X/Overview.html

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 AND 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 skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265