KE KEA watchdog can’t reset problem analysis

In the practical KE KEA usage, a lot of customers meet the watchdog can’t reset problems. Some customers find when they want to enable the watchdog, but can’t really enable the watchdog by set the EN bit in register WDOG_CS1; Some customers find when in debug mode, the EN bit WDOG_S1 register always be clear, but from the reference manual, this bit should be set after reset, even they check their code, and make sure they didn’t disable the watchdog; There also have some customers find when they use the KEXX_DRIVERS_V1.2.1_DEVD code, and set the timeout value register by themselves, but the watchdog can’t reset in the timeout value. Now according to these problems, this document will analyze it and give the recommendation to avoid these problems.

From the above problem description, we can get that there actually mainly 2 reasons caused these problems: 1, software configuration; 2, hardware usage

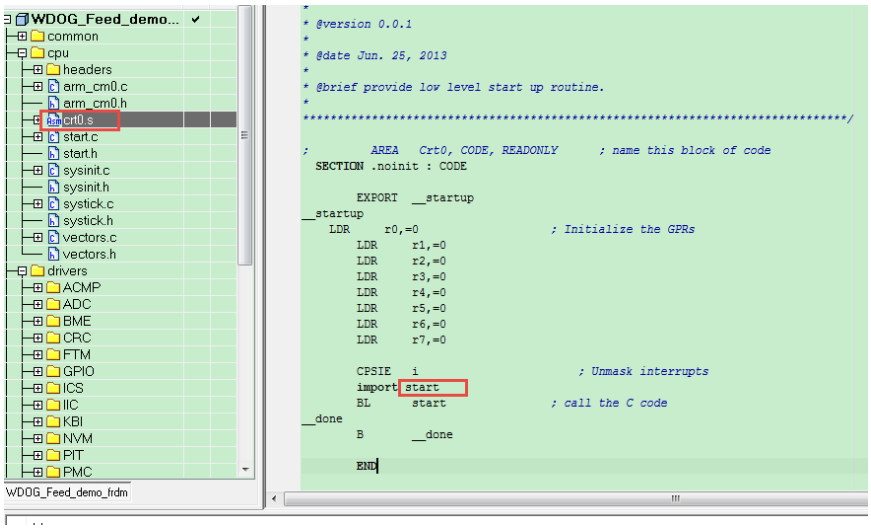
1. Software configuration

1) Start code disable the watchdog

In the KE KEA sample code, after reset, the chip will enter in the start code at first, the start code always disable the watchdog at first, if the watchdog is disabled, the watchdog can’t be enable just by set the EN bit in register WDOG_CS1, because bit EN in register WDOG_CS1 is the write-once bit after reset. It only can be modified when the UPDATE bit is set and with 128 bus clocks after performing the unlock write sequence.

Now how to find the disable code in the start code? Take KEXX_DRIVERS_V1.2.1_DEVD sample code as an example

IAR: from crt0.s, will find the watchdog disable code WDOG_DisableWDOGEnableUpdate(); in the start function.



The above IAR start picture is for KE, but in the KEA start file, you can’t see the start function in the KEA sample code which download from the freescale web, just find the __iar_program_start in cstartup_M_KEA128.s after the reset happens, but where is the __iar_program_start function, it can’t be searched in the whole project. Actually __iar_program_start is the default program entry function, it include the following function:

Call Graph Log ▾

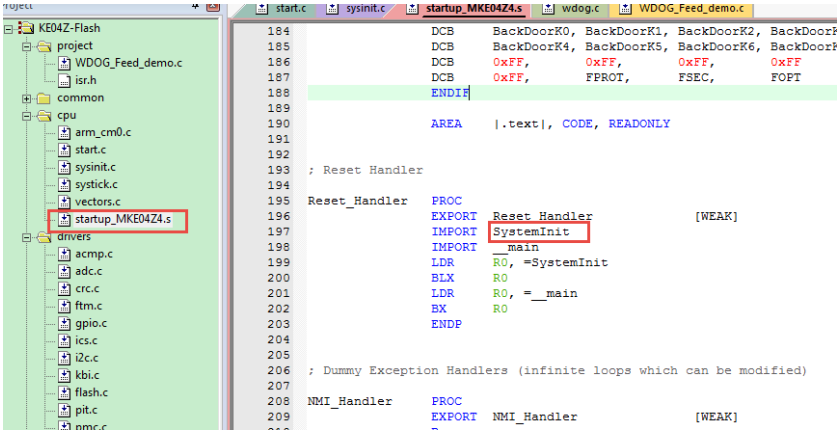
To help you interpret the results of the stack usage analysis, there is a log output option that produces a simple text representation of the call graph (--log call_graph).

Example output:

```
Program entry:
0   __iar_program_start [168]
  0 __cmain [168]
    0 __iar_data_init3 [16]
      8 __iar_zero_init3 [8]
        16 - [0]
      8 __iar_copy_init3 [8]
        16 - [0]
    0 __low_level_init [0]
    0 main [168]
      8 printf [160]
        32 _PrintfTiny [136]
          88 _Prout [80]
            104 putchar [64]
              120 __write [48]
                120 __dwrite [48]
                  120 __iar_sh_stdout [48]
                    144 __iar_get_ttio [24]
                      168 __iar_lookup_ttioh [0]
                  120 __iar_sh_write [24]
                    144 - [0]
          88 __aeabi_uidiv [0]
            88 __aeabi_idiv0 [0]
          88 strlen [0]
    0 exit [8]
      0 _exit [8]
        0 __exit [8]
          0 __iar_close_ttio [8]
            8 __iar_lookup_ttioh [0] ***
    0 __exit [8] ***
```

You can find it will enter __low_level_init function, the watchdog disable code is just in __low_level_init function.

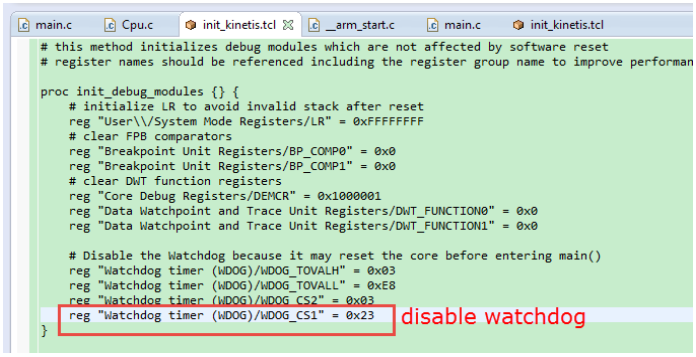
MDK: From startup_MK0XZ4.s will find the watchdog disable code in the SystemInit function.



Codewarrior: From __arm_start.c file, will find the watchdog disable code in __init_hardware function.

2) Codewarrior script init_kinetis.tcl disable the watchdog

To the Codewarrior, just comment the disable watchdog code in the __arm_start.c file is not enough to check the watchdog enable after reset, because in the codewarrior connect script init_kinetis.tcl, there also have the watchdog disable code.



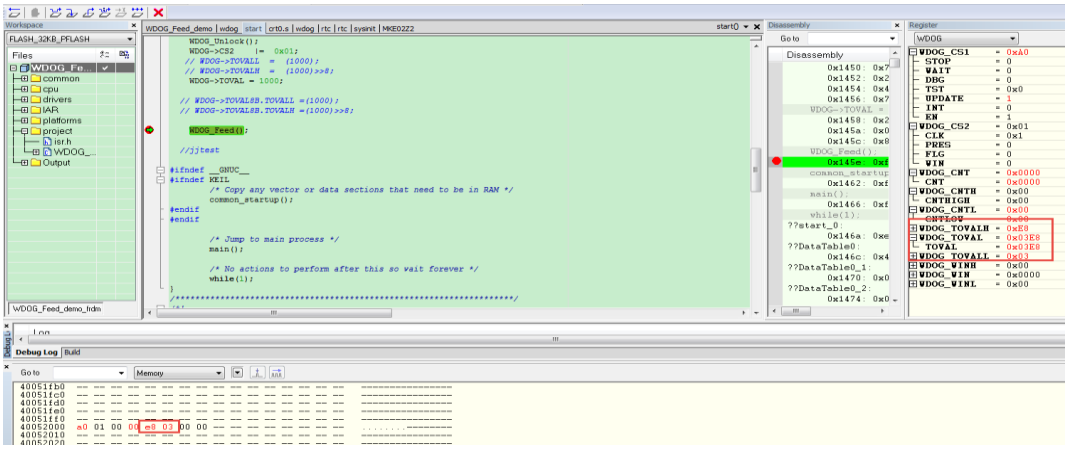
If you want to find the state of EN bit in register WDOG_S1 after reset, you must disable all these watchdog disable code.

3) Timeout register configuration incorrect

From the header file MKE02Z2.h, we can find the time out register define like this:

```
union {                     /* offset: 0x4 */
  __IO uint16_t TOVAL;              /**< WDOG_TOVAL register., offset: 0x4 */
  struct {                  /* offset: 0x4 */
    __IO uint8_t TOVALH;              /**< Watchdog Timeout Value Register: High, offset: 0x4 */
    __IO uint8_t TOVALL;              /**< Watchdog Timeout Value Register: Low, offset: 0x5 */
  } TOVAL8B;
```

This structure means that customer can define the watchdog timeout value by separated unit8 TOVALH, TOVALL or just defined it with unint16 TOVAL. But actually in the IAR project usage, take an example, use 1khz as the clock source for watchdog, then want to set the timeout value as 1s, it means the timeout value should be 1000=0x03e8, so one of the customers configure it like this:

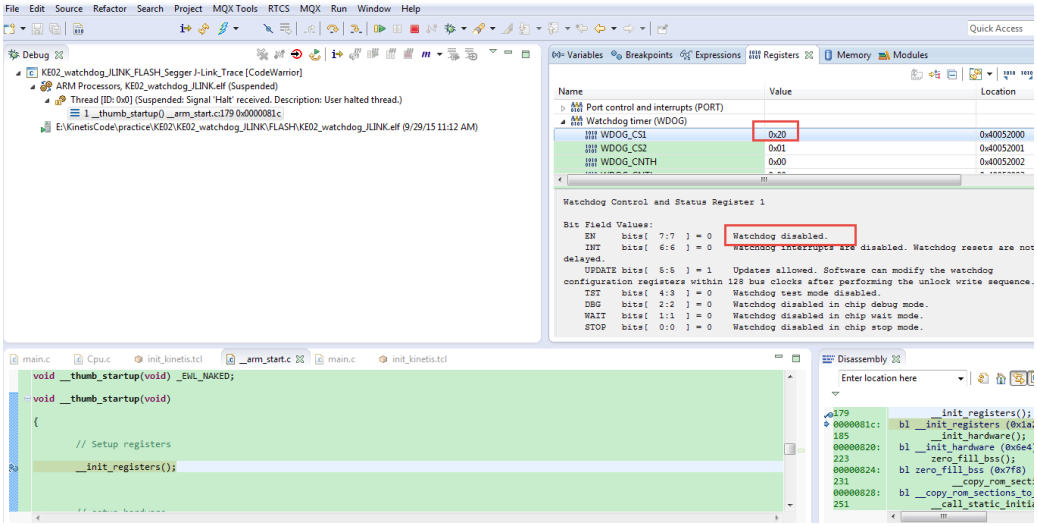


You can find, we need the TOVALL= 0XE8, TOVALH=0X03, but from the test result, the register is TOVALL= 0X03, TOVALH=0Xe8, this will cause the timeout value is much larger than 1000, that is why customer can't reset the mcu after 1s, because the register configuration is not correct. It is caused by the IAR int16 store endian mode, the default IAR endian mode is little endian mode. So in the practical usage, it is recommended to use the separated time out value definition.

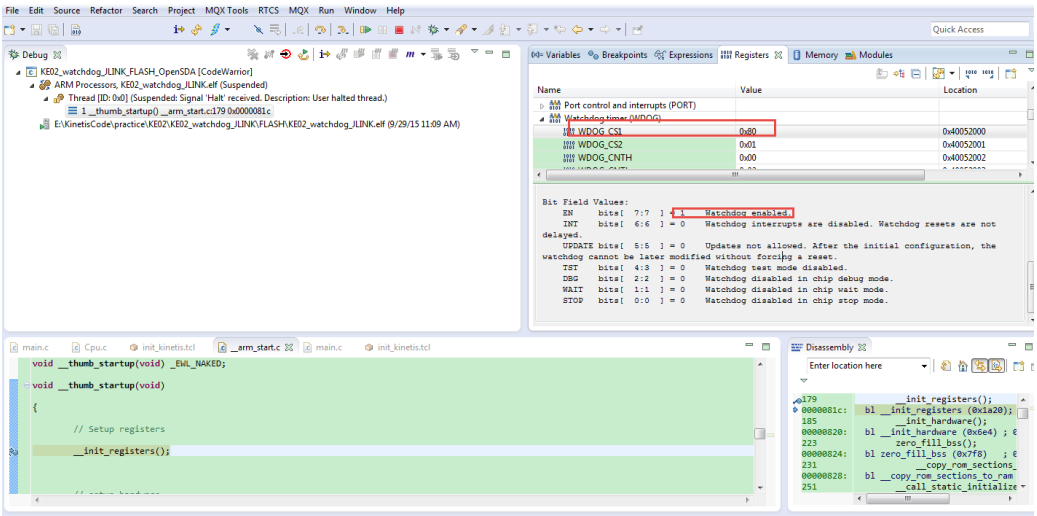
2. Hardware usage

When in debug mode with IDE, some customers find even they comment all the watchdog disable code, they still can't reset the MCU by the watchdog. After check the register WDOG_S1, bit EN is 0, it means the watchdog is disabled. But from the reference manual, we get that after reset, the EN bit should be 1. What caused this? After test, we find this actually caused by the debugger, the debugger hardware which you are using. Eg, in the same project which already comment all the watchdog disable code, SEGGER JLINK will still disable the watchdog, but the PE opensda or PE multilink won't do this, the EN bit is enabled by default, the following is the test picture, take codewarrior as an example:

1) JLINK



2) PE Opensda or PE multilink



So, if you want to test the watchdog in debug mode, and want the EN is set after reset, you can choose PE debugger tool instead of JLINK, but this JLINK feature is just influence the debug mode, after you download the code to the chip flash, and after reset, the EN bit in WDOG_S1 will still be set.

Wish this document will help you get out the problem of watchdog can't be reset.