Keywords: MAXQ1103,DRS diagnostic routine

APPLICATION NOTE 4431

Destructive Reset Diagnostic Routine for the DeepCover Secure Microcontroller (MAXQ1103)

Jul 14, 2009

Abstract: The DeepCover® Secure Microcontroller (MAXQ1103) erases sensitive data when any one of many tamper-detect inputs are triggered. After the destructive reset, the recovery vector allows diagnostic code to run and perform any additional actions required for proper logging or notification of the destructive event. This application note explores the various aspects of writing a diagnostic routine using the Rowley CrossWorks C compiler.

The source code used with this application note is available for download (ZIP).

Introduction

The DeepCover® Secure Microcontroller (MAXQ1103) implements many important features to deter physical tampering and the subsequent compromise of valuable data. One of these features is the Destructive Reset Source, or DRS, subsystem. The DRS feature allows any one of multiple self-destruct inputs (SDI) to cause near-instantaneous erasure of program and data decryption keys and internal static RAM. Assuming that the application has been stored in an encrypted region within the internal program flash memory, erasure of the program decryption key will render the microcontroller inert.

Previous Maxim products, such as the DS5250, incorporated this DRS feature. However, the MAXQ1103 adds the capability to execute an unencrypted diagnostic routine after a destructive reset. This diagnostic routine can execute any unencrypted internal code that does not require access to the external memory bus (which is disabled until the next power-on reset).

As an example, the diagnostic routine could be used to signal a maintenance alert through a modem to a central office and to display an "out of order" indication to the user. This routine also performs erasure and reprogramming of the internal flash memories.

Configuration of the DRS Diagnostic Routine

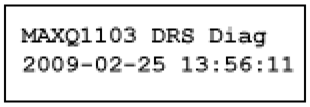
The DRS diagnostic routine is enabled with the DRSRS register bit location DIAE. The DRSRS register bits DIAS[3:0] specify the program code location to which the microcontroller will vector after the causative SDI is cleared. If the diagnostic vector location points to an encrypted memory region, the microcontroller's ROM will simply halt the processor after reset, which is the default action when DIAE=0 (diagnostic routine not enabled).

The DRSRS register may be written at any time during normal program execution. This register also holds flags which indicate the source of the destructive reset. These flags can be used by the diagnostic routine or logged to nonvolatile memory.

Example Application: Secure Clock

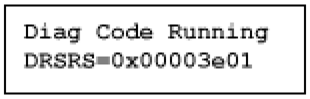
To demonstrate the DRS diagnostic routine on the MAXQ1103, a small application was written in C utilizing the Rowley CrossWorks compiler. This application implements a simple real-time clock (RTC) with the MAXQ1103 EV kit (Rev D).

The date and time are continually displayed on the EV kit's LCD with an update every second. Using the ENT key on the kit's numeric keypad, the user can enter the date and time directly; the cursor increments through the date and time fields automatically. A sample display is shown in **Figure 1**.



```
MAXQ1103 DRS Diag
2009-02-25 13:56:11
```

If any of the SDI input pins are pulled high (to VDDIO) and then released, the microcontroller will execute a destructive reset. This reset will erase the program decryption key and the main program code will no longer run. Instead, the LCD will display a message indicating that a DRS has occurred and showing the contents of the DRSRS register. See **Figure 2**.



```
Diag Code Running
DRSRS=0x00003e01
```

The flags in the DRSRS register are used to decode which SDI source caused the destructive reset. Additionally, the time and date of the last DRS are shown on the LCD display. This last feature is not strictly part of the DRS diagnostic routine, but rather of the DRS logger, which captures the value of the RTS (RTC Second Counter) register at the instant when the SDI triggered.

Implementation Details

The source code to implement the "main application" of an RTC is straightforward and will not be described in this application note. This source code is, however, heavily commented and may be reused as an RTC example. This section of code is stored in the memory segment called CODE, which begins at program memory address 0x000600. A preprogram load command contained in MAXQ30_Target.js script enables an encrypted area of 64kWord with the PMAC and PMSZ registers. This area is 3DES encrypted and will be the location where the CODE program section will reside.

To implement the DRS recovery vector, we must reserve some space which will not be used by the main application program. A memory section is declared in the MAXQ1103.xml file starting at 0x3C600 for 16kWord, and a memory segment is declared within this named RESERVE. This allows us to use the Rowley assembly directive CSEG RECOVERY to place our DRS diagnostic routine at the proper address. This address is dictated by the DIAS[3:0] bits within the DRSRS register.

The function `enable_drs_diag()` writes the DRSRS register to the value 0x00001E01. This value enables the diagnostic vector and selects the diagnostic vector location of 0x3C600.

The DRS diagnostic routine will be called by the ROM after the destructive reset (i.e., once the causative self-destruct input has been removed). This routine should not call any other code within an encrypted memory region (as defined by PMAC/PMSZ), as this code will be fetched encrypted and the resulting execution can cause undesired system operation.

The recovery routine is contained in the file drs.asm and demonstrates the use of the CSEG directive to locate this code in the RECOVERY segment.

While the diagnostic vector may call unencrypted C code in internal program memory, the programmer must set up the C runtime environment before calling this code. One can reference the crt0.asm file included with the Rowley compiler to determine what setup is needed.

Conclusion

The MAXQ1103 diagnostic routine feature gives the application programmer a method to execute code following a security-related destructive reset. It provides a clear separation between the secure application and the unsecured recovery code. The diagnostic routine can then log the security violation and take the appropriate post-destruction actions, such as alerting a remote location or performing further erasure of internal memories.

DeepCover is a registered trademark of Maxim Integrated Products, Inc.

Related Parts	
MAXQ1103	DeepCover Secure Microcontroller with Rapid Zeroization Technology and Cryptography
MAXQ1103-KIT	Evaluation Kit for the DeepCover Secure Microcontroller (MAXQ1103)