针对RAM防护的二维多比特报警法

孙瑞一[1,2*]，张钦宇[1]，梁松海[2]，梁允萍[2]

(1. 哈尔滨工业大学(深圳)电子与信息工程学院 广东 深圳 518055；2. 国民技术股份有限公司 广东 深圳 518057)

【摘要】针对安全芯片中RAM故障对系统安全性的危害，提出了一种基于二维纠错码的RAM防护技术，即二维多比特纠正报警法(TDMBAM)。该技术设计了一种可以有效识别、纠正RAM故障类型的二维纠错码，并可对二维纠错码的纠错能力和RAM的故障类型进行评估。TDMBAM对RAM进行了分区防护，将RAM分为程序存储区和数据存储区，并根据存储区的各自特点设计相异的纠错报警电路。对TDMBAM进行算法仿真，结果表明TDMBAM纠正了给定宽度内所有的连续和非连续故障，并对超过识别能力的故障给出了报警。报警后的系统有99.147 7%没有产生系统故障，并按照系统预定状态工作。

关 键 词 容错；存储器；安全芯片；二维纠错码

中图分类号 TP302.8 **文献标志码** A doi:10.3969/j.issn.1001-0548.2019.02.017

A Novel Two Dimensional Multi-Bits Alarm Method for RAM Protection

SUN Rui-yi[1,2*], ZHANG Qin-yu[1], LIANG Song-hai[2], and LIANG Yun-ping[2]

(1. School of Electronic and Information Engineering, Harbin Institute of Technology (Shenzhen) Shenzhen Guangdong 518055;
2. Nationz Technologies Co., Ltd. Shenzhen Guangdong 518057)

Abstract The two-dimensional multi-bits alarm method (TDMBAM) is proposed to reduce the damage of system security caused by RAM faults. The proposed method is based on two-dimensional error correction code. TDMBAM is able to correct multiple bit faults of RAM with any possible width and evaluate the error correcting capacity of two-dimensional code and RAM faults type. TDMBAM is designed to have partitioned protection for RAM with program memory block and data memory block, and has two different error correcting alarm circuits according to the respective characteristics of two memory blocks. TDMBAM is simulated and the results show that it is able to correct all continuous and discontinuous faults within the given width, and able to alarm faults beyond identification. Up to 99.147 7% alarmed systems are able to work in schedule without system faults.

Key words fault tolerant; memory; security chip; two dimensional error correction code

如今，安全芯片已经广泛的应用在电子商务、税务、通信和国防等领域[1]。安全芯片基于特定的密码算法，可以为敏感信息提供机密性与完整新的保护[2]。敏感信息广泛多样，如银行卡的PIN码，SIM卡的身份签名(subscribe identities)和生物特征电子护照(biometric E-passports)中的指纹[3-4]。这些敏感信息非常有价值。安全芯片在信息安全保护方面的重要作用使其容易受到攻击。

研究表明系统中绝大部分的失效是由RAM引起的[5]。在安全芯片受到攻击时，RAM容易受到攻击，产生意想不到的故障，如数据错误、指令错误、逻辑错误。攻击者利用这些错误，扰乱系统对程序权限的控制和管理，窃取系统特权，并利用系统级的优先级获得安全芯片中只有系统程序可以访问的PIN码、签名和指纹等。针对RAM的防护设计提高安全芯片的可靠性和安全性，保护系统正常运行，防止系统特权被攻击者窃取，是安全芯片防护设计的关键技术之一。

目前针对RAM的防护研究已经较为深入。其中，三模冗余是应用最广泛的硬件冗余防护方法[6-7]，优点是速度快，且可以迅速提高系统的可靠性，缺点是面积和功耗都很大；刷洗定期对RAM重新写一遍原来的数据[8-10]，可以保证RAM内容的可靠性且实现简单，但是会对系统性能产生一定影响；纠错码是一种检测、纠正数据在存储、传输过程中发生改变的强大技术[11-13]，纠错码的纠错能力有限，只能纠正固定位数[14-15]和故障类型的故障，无法识别和纠正超过纠错能力的故障类型。

收稿日期：2017－09－09；修回日期：2017－11－01

作者简介：孙瑞一(1980－)，女，博士，主要从事安全芯片系统容错、系统可靠性及系统安全方面的研究. E-mail: 106543453@qq.com

目前纠错方法都是针对特定的故障模型而设计的，它们的纠错能力有限。而且它们有一个致命缺点，就是无法判断RAM中的错误是否超过了自身的纠错能力。当RAM的错误已经超过了纠错方法的纠错能力时，现有的纠错方法都无法判断，而是将错误的程序或数据按照纠错方法进行纠错。因为错误超过了纠错能力，所以，纠错的结果也是错误的。由于无法判断该结果是错误的，这个错误的结果往往会被系统当作是正确的指令或数据，参与执行，从而导致系统失效或被劫持。被劫持的系统可以被攻击者利用，窃取系统权限，获得敏感信息。

本文针对RAM防护方法不能判断纠错能力和故障类型的弱点，提出了一种基于二维纠错码的RAM防护技术，即二维多比特纠正报警法(two dimensional multi-bits alarm method, TDMBAM)。该方法将RAM分为程序存储区和数据存储区，并对它们进行了分区防护，并根据存储区的各自特点设计相异的纠错报警流程。在纠错过程中，TDMBAM既可以纠正任意给定宽度内的连续故障和非连续故障；又可以判断程序RAM是否产生了超过纠错码纠错能力的故障。当RAM产生纠错码不能纠正的错误时，TDMBAM产生报警信号。

1　二维多比特纠正报警法流程

在一个系统中，程序是CPU要执行的指令集合，在没有升级的情况下不变。如果程序在RAM中受到攻击，产生错误，和原始程序不一致，CPU会执行错误的指令，导致系统失效。攻击者可以趁机挟持CPU，窃取系统特权，获得敏感信息。所以，RAM中程序和原始程序保持一致对安全芯片非常重要。

在系统中，数据分为原始数据、中间数据和结果数据等。数据在程序的执行过程中会经常读出、修改或写入等。如果数据有错误，会使CPU产生错误的结果，影响控制流的执行，发生错误，被攻击者利用。所以，能够判断数据是否产生了纠错码不能纠正的错误是数据防护技术的关键。

针对程序和数据的不同特点，将程序和数据存储在不同的存储区中。TDMBAM防护方法在载入程序、数据和读出程序、数据时实现不同的功能，并对两个存储区块，采用不同的处理报警流程。TDMBAM的过程可以分为：TDMBAM载入流程、程序存储区TDMBAM读流程和数据存储区TDMBAM读流程。

1.1　TDMBAM载入流程

对于程序存储区和数据存储区，TDMBAM在写数据时先计算数据的纠错码，再将数据和纠错码存入存储区的相应区域。其写流程如图1所示。

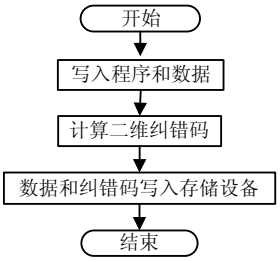

图1　TDMBAM载入流程

1.2　TDMBAM读流程

TDMBAM在读程序、数据时计算读出程序、数据的纠错码，并和载入程序、数据时存储的纠错码一起评估当前程序、数据的故障类型和纠错能力，再判断程序、数据的故障类型是否超过了纠错能力。如果没有超过，则表示故障可以被纠正，纠正程序、数据，并读出纠正后的程序、数据；否则，表示故障类型超过了纠错码的纠错能力，故障不能被纠正，发出报警信号。TDMBAM对程序存储区和数据存储区采用了不同的方法来响应和处理报警信号。

程序存储区TDMBAM读流程如图2所示。

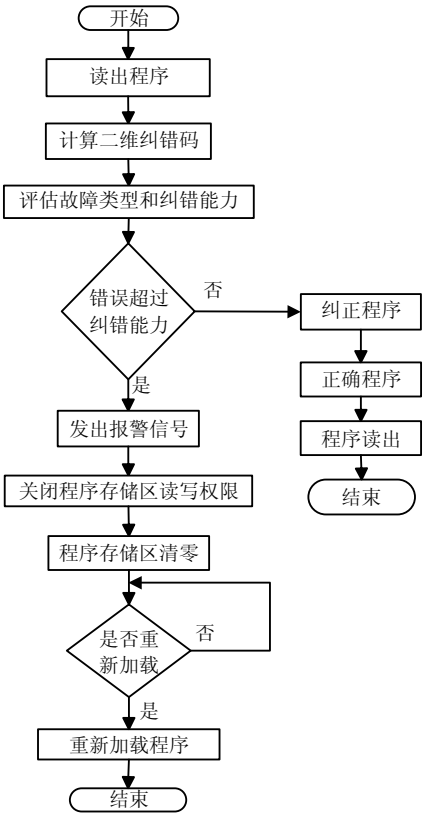

图2　程序存储区TDMBAM读流程

由于程序存储区的故障对安全芯片有严重影响，所以，TDMBAM在发出报警信号后，关闭程序存储区的读写权限，并将程序存储区清零，等待程

序重新加载。

数据存储区TDMBAM读流程如图3所示。

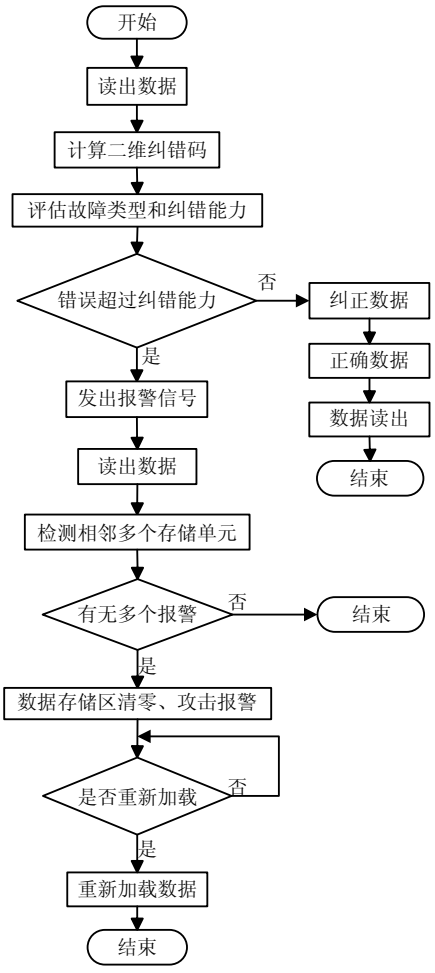

图3 数据存储区TDMBAM读流程

与程序错误会立即引起CPU执行错误不同，数据错误会在控制流上传播，导致结果错误、路径错误等错误。这些错误可以被追踪、回溯并纠正，不需要产生报警信号时立刻终止程序执行。而且，数据在程序执行过程中不停变化，如果对数据存储区采用和程序存储区一样的方法来响应报警信号，会影响系统的效率。所以，对于数据存储区，TDMBAM在发出报警信号的同时读出有错误的数据。CPU接到报警后可以继续执行，也可以跳转到前面一个能够正确执行的检查点重新执行。为了判断数据存储区的错误是否影响了多个存储单元，TDMBAM自动检测相邻存储单元中的数据是否有错误，如果其他数据没有错误，则说明错误的影响较小，系统将继续执行，如果检测到若干个存储单元的错误都不可纠正，则可以判断该错误为严重错误。TDMBAM给CPU发出报警信号，同时对RAM中的数据清零，并等待重新加载数据。

2 二维纠错码

TDMBAM选择二维纠错码作为RAM的纠错码，并以总线位宽为单位计算二维纠错码。与传统二维纠错码相比[16]，以总线位宽为单位的优点可以一次计算出二维纠错码所需要的所有校验码，而不需要为了计算列纠错码在写数据之前先读出存储单元原有的数据。TDMBAM对效率没有损害，适应高速RAM的应用。

2.1 二维纠错码的生成

二维纠错码的生成过程包括数据分组、矩阵表示、行列确定和纠错码生成4个步骤。

2.1.1 数据分组

将存储内容(程序或数据)以总线位宽为单位进行分组，将在物理上相邻的存储单元分到不同的逻辑组中。分组可以将多比特故障分散到不同组中，使系统在正常工作时每组存储内容一般只有一个比特故障，从而实现纠正多位故障。

分组的依据是RAM的故障率。设 L 是故障率，即RAM在正常工作环境下由环境产生的多比特故障的最大比特数；l 是分组数，根据 L 将存储内容分成 l 组。由于二维纠错码一次可以纠正一组中的单比特故障，所以，在保证 $L \leqslant l$ 时，可以保证二维纠错码可以纠正正常工作环境下的大多数故障，满足系统正常工作时的容错需要。由于分组后故障点分散到不同的组中，使每一组存储内容中最多有一个比特有故障。如果有一组存储内容出现了其他故障类型，则说明该故障不在正常工作时的故障模型之内，也超出了二维纠错码的纠错能力。

设存储内容为 $\{d_0,d_1,\cdots,d_n\}$，其中 $d_0,d_1,\cdots,d_n$ 表示组成存储内容的比特，故障率是 L。根据 $L \leqslant l$ 的条件，将存储内容分成 l 组，并将物理上相邻的比特分入不同的组。则有，第一组：$\{d_0,d_l,d_{2l},\cdots\}$；第二组：$\{d_1,d_{l+1},d_{2l+1},\cdots\}$；第三组：$\{d_2,d_{l+2},d_{2l+2},\cdots\}$ 等。

与交错(interleaving)方法不同，这种分组方法没有改变存储单元的物理位置，不会使RAM的字、列选择布线变长，不含增加面积、存取时间和功耗等。因而，该方法具有低功耗、高效率的特点。

2.1.2 矩阵表示

存储内容分组后，在逻辑上将每一组的存储内容看作是一个 (m,n) 的矩阵，其中 m 代表行数，n 代表列数。存储内容表示成矩阵形式后可以利用行号和列号对所有比特进行定位。定位是评估故障类型和纠错能力的基础。

存储内容分组后的矩阵表示如图4所示。

d_0	d_{1l}	…	$d_{(n-1)l}$
d_{nl}	$d_{(n+1)l}$	…	$d_{(2n-1)l}$
…	…	…	…
$d_{(m-1)nl}$	$d_{(m-1)(n+1)l}$	…	$d_{(m-1)(2n-1)l}$

图4　分组后存储内容的矩阵表示

2.1.3　行列确定

行 m 和列 n 的值不同会影响二维纠错码中校验位的个数，带来不同的硬件冗余代价。根据每一组存储内容的多少合理地确定矩阵的 m 和 n 的大小。m 和 n 的选择以最小化硬件代价为目标。在设计二维纠错码时可以对 m 和 n 的值进行遍历，选取总校验位最少的一组 m 和 n 的值作为矩阵的 m 和 n。

由于当系统固定之后，它的总线位宽就已经确定了，所以，对于一个系统来说 l、m 和 n 在设计电路时确定。在工作时，l、m 和 n 的值固定，不需要动态计算，不影响系统效率，适用于高速系统。

2.1.4　二维纠错码生成

对每一个矩阵的每一行和每一列都生成一个奇偶校验码。用 p_{m0}、p_{m1}、p_{m2} 等表示行校验码。它们由矩阵中一行的比特计算，其生成公式为：

$$\begin{cases} p_{m0} = d_0 \oplus d_l \oplus d_{2l} \oplus \cdots \\ p_{m1} = d_{nl} \oplus d_{(n+1)l} \oplus d_{(n+2)l} \oplus \cdots \\ p_{m2} = d_{2nl} \oplus d_{(2n+1)l} \oplus d_{(2n+2)l} \oplus \cdots \\ \vdots \end{cases} \tag{1}$$

用 p_{n0}、p_{n1}、p_{n2} 等表示列校验码。它们由矩阵中一列的比特计算，其生成公式为：

$$\begin{cases} p_{n0} = d_0 \oplus d_{nl} \oplus d_{2nl} \oplus \cdots \\ p_{n1} = d_l \oplus d_{(n+1)l} \oplus d_{(2n+1)l} \oplus \cdots \\ p_{n2} = d_{2l} \oplus d_{(n+2)l} \oplus d_{(2n+2)l} \oplus \cdots \\ \vdots \end{cases} \tag{2}$$

当矩阵内有且只有一个比特故障时，行校验码可以显示故障比特所在的行，列校验码可以显示故障比特所在的列。由行校验码和列校验码可以定位并纠正故障比特。由于分组的作用，该二维纠错码可以纠正存储内容中 l 个比特范围内的所有连续和不连续故障，从而保证RAM的正常工作。

当矩阵内有多余一个比特故障时，可以由行校验码和列校验码对故障类型进行识别，并发出警报。系统可以根据警报采用相应的措施，不会将错误地纠正存储内容当作正确的纠正存储内容执行，从而避免系统特权被攻击者窃取。

2.1.5　举例说明

以RAM的总线位宽为64位、故障率 $L=4$ 为例。TDMBAM防护技术先根据故障率 L 将存储内容分成 l 组，即 $l=4$。每组存储内容有16个比特，并将物理上相邻的比特分配到不同的组。设存储内容为：$\text{data}=\{d_0,d_1,\cdots,d_{63}\}$，其中 $d_0,d_1,\cdots,d_{63}$ 表示组成存储内容的比特。则第一组存储内容表示为：$\text{data}_1=\{d_0,d_4,\cdots,d_{60}\}$；第二组存储内容表示为：$\text{data}_2=\{d_1,d_5,\cdots,d_{61}\}$；第三组存储内容表示为：$\text{data}_3=\{d_2,d_6,\cdots,d_{62}\}$；第四组存储内容表示为：$\text{data}_4=\{d_3,d_7,\cdots,d_{63}\}$。分组可以保证RAM在正常工作时产生的多比特故障都能被纠正。

分组后将每组存储内容表示成矩阵形式。并用遍历的方式确定行数 m 和列数 n，目标是校验位最少。如 $m=2$，$n=8$ 时，行列校验码共10位；$m=4$，$n=4$ 时，行列校验码共8位。所以选择 $m=4$，$n=4$。存储内容分组的矩阵表示如图5所示。

d_0	d_4	d_8	d_{12}
d_{16}	d_{20}	d_{24}	d_{28}
d_{32}	d_{36}	d_{40}	d_{44}
d_{48}	d_{52}	d_{56}	d_{60}

图5　第一组存储内容的矩阵表示

对每一个矩阵的每一行和每一列都生成一个奇偶校验码。以第一组存储内容为例，校验码如图6所示：

d_0	d_4	d_8	d_{12}	p_{m0}
d_{16}	d_{20}	d_{24}	d_{28}	p_{m1}
d_{32}	d_{36}	d_{40}	d_{44}	p_{m2}
d_{48}	d_{52}	d_{56}	d_{60}	p_{m3}
p_{n0}	p_{n1}	p_{n2}	p_{n3}	

图6　第一组行列校验位生成图

图6中，p_{m0}、p_{m1}、p_{m2}、p_{m3} 是行校验码，由矩阵中一行的比特计算。它们的生成公式为：

$$\begin{cases} p_{m0} = d_0 \oplus d_4 \oplus d_8 \oplus d_{12} \\ p_{m1} = d_{16} \oplus d_{20} \oplus d_{24} \oplus d_{28} \\ p_{m2} = d_{32} \oplus d_{36} \oplus d_{40} \oplus d_{44} \\ p_{m3} = d_{48} \oplus d_{52} \oplus d_{56} \oplus d_{60} \end{cases} \tag{3}$$

p_{n0}、p_{n1}、p_{n2}、p_{n3} 是列校验码，由矩阵中一列的比特计算。它们的生成公式为：

$$\begin{cases} p_{n0} = d_0 \oplus d_{16} \oplus d_{32} \oplus d_{48} \\ p_{n1} = d_4 \oplus d_{20} \oplus d_{36} \oplus d_{52} \\ p_{n2} = d_8 \oplus d_{24} \oplus d_{40} \oplus d_{56} \\ p_{n3} = d_{12} \oplus d_{28} \oplus d_{44} \oplus d_{60} \end{cases} \tag{4}$$

其他组的校验码生成和第一组相同。对于一个64比特的存储内容，有32个比特的校验码来保护。

2.2 二维纠错码的纠错过程

对于矩阵来说，矩阵中的单比特故障，可以利用行、列校验码定位，如图7所示。

d_0	d_4	d_8	d_{12}	p_{m0}
d_{16}	d_{20}	d_{24}	d_{28}	p_{m1}
d_{32}	d_{36}	d_{40}	d_{44}	p_{m2}
d_{48}	d_{52}	d_{56}	d_{60}	p_{m3}
p_{n0}	p_{n1}	p_{n2}	p_{n3}	

图7 矩阵内1位错误图

图7中，设 d_{20} 比特有故障，则导致行校验码 p_{m1} 和列校验码 p_{n1} 都有效。可以看出，p_{m1} 和 p_{n1} 所代表的行和列相交于一点，则这点就是有故障的比特。

当矩阵内有且只有1位故障时，在矩阵内最多有一个行校验位和一个列校验位显示存储内容有故障，且它们所代表的直线相交于一点。这个交叉点就是有故障的比特。通过对该比特取反可以得到正确的存储内容值。

理论上，能够导致行校验码 p_{m1} 和列校验码 p_{n1} 都显示有错误的情况不只图7所表示的单点故障，还有其他故障类型，如图8所示的故障类型也表现为行校验码 p_{m1} 和列校验码 p_{n1} 都显示存储内容有错误。

图8中所示错误表示RAM故障最少在21个比特的范围内引起5个以上比特的错误，且故障点的位置固定。这种故障产生的概率很小，所以不在考虑的范围之内。

可以看出，二维纠错码可以纠正的单比特错误在校验码上的表现形式只有一种，即当行、列校验码都有且只有一位显示存储内容有故障时，故障可以被定位和纠正。

d_0	d_4	d_8	d_{12}	p_{m0}
d_{16}	d_{20}	d_{24}	d_{28}	p_{m1}
d_{32}	d_{36}	d_{40}	d_{44}	p_{m2}
d_{48}	d_{52}	d_{56}	d_{60}	p_{m3}
p_{n0}	p_{n1}	p_{n2}	p_{n3}	

图8 与图7校验结果相同的故障图

在存储内容写入RAM时计算 p_{mj} 和 p_{nj}，并保存在内存中，其中 $j = \{0,1,2,3\}$。在存储内容读出RAM时重新计算二维纠错码，记为 p'_{mj} 和 p'_{nj}。比较 p_{mj}、p_{nj} 和 p'_{mj}、p'_{nj}，得到检验因子 C_{mj} 和 C_{nj}：

$$\begin{cases} C_{mj} = p_{mj} \oplus p'_{mj} \\ C_{nj} = p_{nj} \oplus p'_{nj} \end{cases} \tag{5}$$

式中，$j = \{0,1,2,3\}$。

任意检验因子 C_{mj} 和 C_{nj} 有效都表示小组内存在故障。

用位置因子 L_i 表示故障存储内容的位置，其中 $i = \{0,1,\cdots,9\}$。位置因子可以用检验因子计算：

$$\begin{cases} L_0 = C_{m0}\overline{C_{m1}C_{m2}C_{m3}} \\ \vdots \\ L_9 = \overline{C_{n0}C_{n1}C_{n2}C_{n3}} \end{cases} \tag{6}$$

当 $i = \{0,1,2,3\}$ 时，L_i 表示行校验码有且只有一位显示存储内容有故障。当 $i = \{4,5,6,7\}$ 时，L_i 表示列校验码有且只有一位显示存储内容有故障。当 $i = 8$ 时，L_i 表示行校验码没有显示存储内容有故障。当 $i = 9$ 时，L_i 表示列校验码没有显示存储内容有故障。

用纠正因子 S_k 表示有故障存储内容的位置，计算如下：

$$\begin{cases} S_0 = L_0 L_4 \\ S_1 = L_0 L_5 \\ \vdots \\ S_{60} = L_3 L_7 \end{cases} \tag{7}$$

根据纠正因子纠正错误比特：

$$d_{k\_\text{correct}} = d_k \oplus S_k \quad k = \{0,1,\cdots,63\} \tag{8}$$

此例中的二维纠错码可以纠正4比特范围内的任意比特的连续和不连续的错误。

2.3 二维纠错码的报警过程

二维纠错码不仅可以纠正一个矩阵内的一位故障，即物理上相邻的 l 比特范围内的任意连续和非连

续故障。也可以识别任何超过这个范围的故障类型，并给出报警信号。

可以被纠正的故障是行、列校验码都有且只有一位显示有故障。除了这种故障模型外的其他形式都表示该矩阵内存在纠错码纠正不了的故障。如只有一位或多位行校验码表示存储内容有故障且没有列校验码表示存储内容有故障；只有一位或多位列校验码表示存储内容有故障且没有行校验码表示存储内容有故障；有多于一位的行校验码和列校验码表示存储内容有故障等。这些情况表示该矩阵内存在不止一个比特故障，则可以判断该故障类型超过了TDMBAM的二维纠错码的纠错能力。

除了可以被纠正的故障和没有故障的情况，如果产生了其他形式的故障，则产生报警信号。报警信号 A 为：

$$A=\overline{(L_0+L_1+L_2+L_3)(L_4+L_5+L_6+L_7)+L_8L_9} \quad (9)$$

二维纠错码发出报警信号，指示当前错误类型超过了二维纠错码的纠错能力。

3　RAM多位错误纠正和判断的电路设计

RAM多位错误纠正和判断的电路设计如图9所示。图中RAM由存储内容行、译码器、列I/O、计算纠错码模块、计算检验、纠正因子模块和纠正内容模块组成。

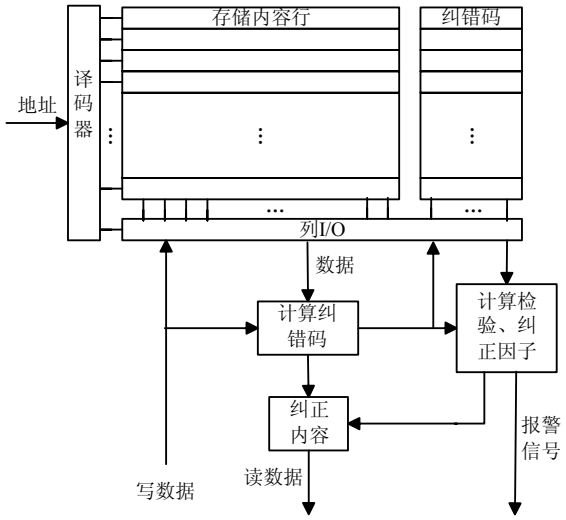

图9　用TDMBAM检测和纠正错误

载入存储内容时，译码器和列I/O根据地址信号选定存储内容单元。存储内容写入选定的存储内容单元。同时存储内容经过计算纠错码模块，计算行、列校验码 p_{mj} 、 p_{nj} 。并根据地址信号将校验码存入RAM的相应区域。

读出存储内容时，存储内容通过列I/O进入计算纠错码模块，计算读出存储内容的行、列校验码 p'_{mj} 和 p'_{nj} 。它们和RAM中的校验码 p_{mj} 和 p_{nj} 一起输入给计算检验、纠正因子模块，得到检验因子 C_{mj} 和 C_{nj} 。由检验因子计算位置因子和报警信号。在报警信号无效时，由位置因子计算纠正因子 S_k 。纠正因子和存储内容一起输入给纠正存储内容模块，得到正确的输出存储内容。

4　实验与分析

4.1　实验环境

对密码芯片防御方法的测试可以在芯片生产后或在设计阶段进行。在生产后进行测试存在安全性测试周期长以及防御方法的性能无法在设计前期得到检验等问题[17]。所以，另一种广泛应用的方法是用算法仿真的方法在设计阶段进行防御方法测试[18-19]。

本文采用在设计阶段对TDMBAM方法进行算法仿真测试。在算法测试、理论验证阶段，算法仿真测试的好处是快速、高效、经济，而且保证测试结果的可靠性。

TDMBAM的执行和报警过程由软件进行仿真，并监控系统在故障程序、数据有错误时系统的状态，如图10所示。

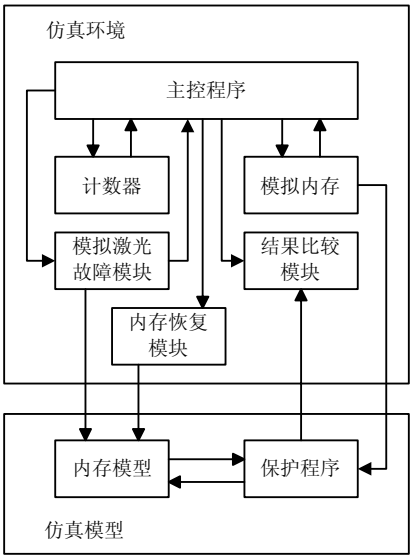

图10　TDMBAM的仿真环境和仿真模型

软件仿真包括仿真环境和仿真模型两部分。仿真环境是软件仿真的平台，包括主控程序、计数器、模拟激光故障模块、内存恢复模块、模拟内存和结果比较模块。主控程序主要控制仿真的流程和生成标准的仿真结果。计数器记录仿真的次数，在本次

仿真实验中，计数器设置为100万次。模拟激光故障模块产生随机数在一定范围内产生单比特或多比特故障。内存恢复模块在每次仿真结束后将内存模型恢复成原来是状态。模拟内存中存储着原始的程序和数据。结果比较模块对标准的仿真结果和保护程序得到的仿真结果进行比较。仿真模型是对TDMBAM方法的软件仿真模型，包括内存模型和保护程序。内存模型是对硬件存储器的模拟。保护程序实现TAMBAM方法。

模拟激光故障模块模拟的激光注入的设备是Riscure公司的高精度激光错误注入平台，该平台有808 nm光枪和1 064nm两种光枪。808 nm光枪强度最大14 w，光斑最小 $6\times1.4\ \mu m^2$，最大 $60\times14\ \mu m^2$；1 064 nm光枪强度最大20 w，光斑最小 $6\times1.4\ \mu m^2$，最大 $60\times14\ \mu m^2$。当对存储器进行激光注入时，可以在存储器的特定区域产生故障，故障的比特数大致固定。模拟激光故障模块用随机数在程序和数据的特定位置中产生单比特和多比特故障。

具体的仿真流程如图11所示。

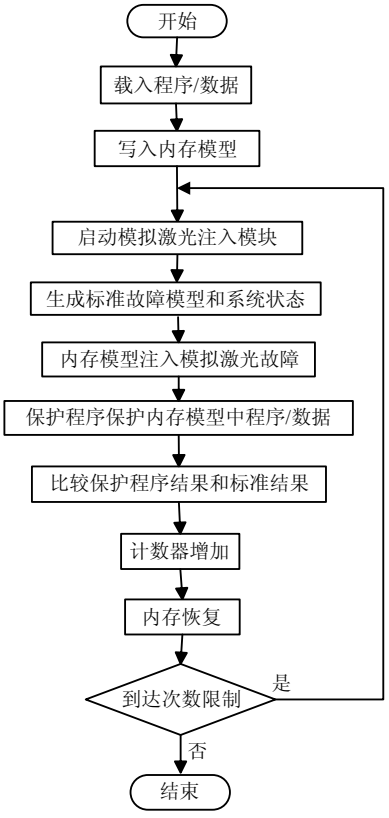

图11 TDMBAM的仿真流程

主控程序先向模拟内存中载入程序和数据。之后模拟内存通过保护程序将程序和数据写入内存模型，并同时计算二维纠错码写入内存模型中。主控程序接到模拟内存的完成信号后启动TDMBAM仿真模型。首先，主控模块启动模拟激光故障模块，随机的生成单比特或多比特故障。其次，主控程序根据模拟内存的内容和模拟激光故障的结果生成标准的故障模型和系统状态结果，这个结果作为仿真过程中的标准结果。再次，模拟激光故障模块将故障写入内存模型中。之后，保护程序读出内存模型中的程序和数据，并根据读出的程序和数据生成真实的故障模型和系统状态结果。这个结果在结果比较模块和标准的结果进行比较得出仿真的结果。最后，计数器加1，主控程序启动内存恢复模块，恢复内存模型中的程序和数据。再重复TDMBAM的仿真过程，直到达到仿真次数。

为了使仿真更具普遍性，可以用多个程序和数据进行仿真测试。

4.2 实验结果

对算法程序进行了100万次故障仿真。产生故障如表1所示。

表1 模拟激光注入产生的故障类型

故障范围比特数	发生次数
0	1 882
1	109 384
2	273 552
3	399 637
4	126 791
>4	88 754

TDMBAM防护方法在测试过程中没有漏报和误报，即TDMBAM纠正了所有故障范围在4比特及4比特以下的故障，并对所有故障范围超过4比特的故障做出报警。所以，TDMBAM的故障纠正率是100%，故障报警率是100%。

对于超过纠错能力的故障类型，TDMBAM防护方法按照防护流程，对报警信号进行响应。实验表明，响应后系统进入了不同的状态，包括纠正故障状态、程序存储区故障重新加载状态、数据存储区故障重新加载状态、数据存储区故障被控制电路包容状态和系统故障状态等。其中纠正故障状态、程序存储区故障重新加载状态、数据存储区故障重新加载状态、数据存储区故障被控制电路包容状态都是TDMBAM防护方法预先确定的系统正常运行的状态；系统故障状态表示系统失效，可能被攻击者利用。

表2显示了输入故障程序、数据后，系统进入不同状态的次数和比率。

表2　模拟激光注入产生故障的系统状态

系统状态	发生次数	所占比例/%
纠正故障	928 184	92.818 4
程序存储区故障重新加载	38 572	3.857 2
数据存储区故障重新加载	14 128	1.412 8
数据存储区故障包容	10 593	1.059 3
系统故障	8 523	0.852 3

对于程序存储区，只要检测出不能纠正的故障就需要重新加载重构信息，共重新加载38 572次，占故障次数的3.857 2%。

对于数据存储区，检测出不能纠正的故障后产生报警信号，并对相邻3个数据单元进行检测。如果相邻3个数据单元有一个存储单元有故障报警，则重新加载数据并对数据存储区清零。TDMBAM检测出了所有超过纠错能力的33 244个故障。并对其中14 128个故障进行了重新加载，占故障次数的1.412 8%。在没有重新加载的19 116次注入中有10 593次注入产生的故障被控制电路包容，占故障次数的1.059 3%，有8 523次故障导致了系统失效，占故障次数的0.852 3%。

总的来说，故障注入后，系统继续按预定流程继续运行的次数占总次数的99.147 7%，系统失效的次数为0.852 3%。

5　结 束 语

本文提出一种针对RAM的二维多比特纠正报警(TDMBAM)防护方法。该方法不仅可以纠正任意给定宽度内的连续故障和不连续故障，而且可以在故障类型超过纠错能力时采用容错、重新加载数据等方式防止系统失效，从而保证安全芯片运行的可靠性和安全性。故障报警使系统可以识别异常故障，并对异常故障做出相应的反应。在安全芯片的攻防过程中增加防御方法的主动性和多样性，使安全芯片更加安全、可靠。防护方法对报警信号的响应会对系统可靠性和系统效率产生影响，是下一步研究的重点。

参 考 文 献

[1] 王红胜, 宋凯, 张阳, 等. 针对高级加密标准算法的光故障注入攻击[J]. 计算机工程, 2011, 37(21): 97-99.
WANG Hong-sheng, SONG Kai, ZHANG Yang, et al. Optical fault injection attack on advanced encryption standard algorithm[J]. Comprter Engineering, 2011, 37(21): 97-99.

[2] 温圣军, 张鲁国. 安全芯片错误诱导攻击防护方案[J]. 计算机工程, 2009, 35(17): 135-137.
WEN Sheng-jun, ZHANG Lu-guo. Protection scheme of error inject attack for secure chip[J]. Comprter Engineering, 2009, 35(17): 135-137.

[3] WOUDENBERG J G J V, WITTEMAN M F, MENARINI F. Practical optical fault injection on secure micro-controllers[C]//2013 Workshop on Fault Diagnosis and Tolerance in Cryptography. Tokyo, Japan: [s.n.], 2011: 91-99.

[4] MUIR J A. Seifert's RSA fault attack: Simplified analysis and generalizations[C]//8th International Conference, Icics 2006. Raleigh: [s.n.], 2006: 420-434.

[5] RAJSUMAN R. Design and test of large embedded memories: an overview[J]. IEEE Design and Test of Computers, 2001, 18(3): 16-27.

[6] PRAVEEN K S, JEREMY R, SRINIVAS K. Selective triple modular redundancy (STMR) based single-event upset(SEU) tolerant synthesis for FPGAs[J]. IEEE Transactions on Nuclear Science, 2004, 51(5): 2957-2969.

[7] PRAAT B, CAFFREY M, GRAHAM P, et al. Improving FPGA design robustness with partial TMR[C]//The 44th Reliability Physics Symposium. San Jose, California, USA: [s.n.], 2006: 26-30.

[8] MUKHERJEE S S, EMER J, FOSSUM T, et al. Cache scrubbing in microprocessors: Myth or necessity?[C]//The 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04). Papeete, Tahiti, French Polynesia: [s.n.], 2004: 3-5.

[9] AMD. BIOS and Kernel developer's guide for AMD athlon 64 and AMD opteron processors[EB/OL]. [2017-08-17]. http://go.microsoft.com/fwlink/?LinkId=28165.

[10] AWASTHI M, SHEVGOOR M, SUDAN K, et al. Efficient scrub mechanisms for errorprone emerging memories [C]//The 18th International Symp High Performance Computr Archit. New Orleans, Louisiana, USA: [s.n.], 2012: 25-29.

[11] DUBROVA E. Fault-tolerant design[M]. New York: Springer, 2013.

[12] ANDO H. High-performance energy-efficient micro-processor design[M]. Netherlands: Springer, 2006.

[13] SHAFIK R A, MATHEW J, PRADHAN D K. Energy efficient fault-tolerant systems[M]. New York: Springer, 2014.

[14] CARDARILLI G C, LEANDRI A, MARINUCCI P, et al. Design of a faulttolerant solid state mass memory[J]. IEEE Transactions on Reliability, 2003, 52(4): 476-491.

[15] EBRAHIMI M, RAO P M B, SEYYEDI R, et al. Low-cost multiple bit upset correction in SRAM-based FPGA configuration frames[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2016, 24(3): 932-943.

[16] KIM J, HARDAVELLAS N, MAI K, et al. Multi-bit error tolerant caches using two-dimensional error coding[C]// The 40th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, UK: [s.n.], 2007: 13-17.

[17] TIRI K, HWANG D. A side-channel leakage free coprocessor IC in 0.18 μm CMOS for embedded AES-based cryptographic and biometric processing[C]// ACM/IEEE Design Automation Conference(DAC). [S.l.]: IEEE, 2005: 222-227.

[18] 周永彬, 徐秋亮. 中国密码学发展报告2008[M]. 北京: 电子工业出版社, 2009.
ZHOU Yong-bin, XU Qiu-liang. The development report of China cryptology 2008[M]. Beijing: Publishing House of Electronics Industry, 2009.

[19] YAO J, ZHANG T. Hardware/software co-design to secure cryptochip from side channel analysis at design time[C]//Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT). [S.l.]: IEEE, 2010, 6: 88-91.

编　辑　税　红