文章编号:1671-4598(2020)07-0230-05 DOI:10.16526/j.cnki.11—4762/tp.2020.07.046 中图分类号:V417 文献标识码:A

空间计算机存储单元容错性研究

龚 博,郑 晨,张洪源,刘 莉,王超杰
(北京宇航系统工程研究所,北京 100076)

摘要:近年来随着商业航天的发展,空间飞行器计算机系统开始大量使用工业级甚至商业级电子元器件,这种设计带来众多优点的同时却存在一个严重问题:商用芯片无法直接适应空间环境,商用元器件存储单元在空间环境下现单粒子多位翻转 MBU(multiple bits upset)的情况越来越多,传统容错方法由于纠错能力不足无法解决此类问题,为解决上述不足文中提出了一种改进型准循环码作为容错方法来解决单粒子多位翻转问题;仿真和试验结果表明,该方法可以满足存储单元相邻位 2 位、3 位翻转的纠错需求,且编码逻辑简洁,编解码延迟较低,适于工程应用。

关键词:单粒子多位翻转;计算机存储单元;空间飞行器

Research on Fault Tolerant of Spacecraft Computer Storage Units

Gong Bo, Zhang Hongyuan, Zheng Chen, Liu Li, Wang Chaojie
(Beijing Institute of Astronautical Systems Engineering, Beijing 100076, China)

Abstract: With the development of commercial aerospace in recent years, space computer systems are increasingly using industrial—grade or even commercial—grade electronic components to complete the design, this design method brings many advantages but there is a serious problem: commercial chips cannot directly adapt to the space environment, component storage units are more and more affected by MBU (multiple bits upset), the traditional fault—tolerant methods cannot solve such problems. In order to solve the above shortcomings, a new algorithms of error correction code is proposed as a fault—tolerant method to solve the MBU problem in the space environment. The simulation and test results show that this method can meet the error correction requirements of adjacent 2 bits and 3 bits of the storage unit, the coding logic is concise and the codec time delay is low, which is suitable for engineering applications.

Keywords: MBU; spacecraft; computer storage unit

0 引言

空间中的辐射特性对电子元器件影响较大,主要因为辐射环境中高能粒子(重粒子、质子、中子、X 射线、γ 射线等)对集成电路造成的破坏,其中辐射总剂量(TID, total ionizing dose)和单粒子效应(SEE, single event effect)为主要的表现形式,近年来随着电子器件中的存储单元尺寸越来越小,TID 效应几乎可以忽略不计[1-3],但接近纳米级存储单元逻辑门和极低核心电压导致单粒子效应(SEE, single event effect)大大增强,SEE 又分为单粒子翻转(SEU, single event upset)和单粒子锁定(SEL, single event latch),本文主要是针对逻辑门翻转效应进行研究。

SEU 是指半导体逻辑器件由于受到单个高能粒子撞击而使逻辑状态翻转的情况,这种情况造成的错误不是硬伤是可恢复的,因此也被称为软错误。空间计算机系统中 SEU 发生频率最高的是面积相对较大的存储器部分(例如外部 SRAM),同时 CPU 和接口电路中存储单元也有发生的可能(例如缓存,锁存器)。

传统的容错算法和容错方案只针对单粒子单位翻转,但是随着存储单元的工艺尺寸越来越小,核心电压越来越低,单粒子多位翻转的情况越来越多。Alsat—1 小卫星的观测表明在其轨道上每天每一位存储单元发生单粒子翻转的概率约为百万分之一,其中 80% 为单粒子单位翻转,20%为单粒子两位翻转和多位翻转[4],而 UoSAT—12 卫星星载计算机 SRAM 对单子翻转的在轨实验统计得出结论在 LEO 轨道的单子多位翻转占单粒子翻转总数 5%~10%[5],尽管数据不尽相同,但我们可以得出结论:MBU 已经不容忽视。

目前国内外许多卫星、在轨飞行器都采用了硬件的冗余容错来实现对空间计算机外围的存储芯片的纠错和检错,实现方式有抗辐照加固、三模冗余,专用的 EDAC 芯片等,但这些方案都不适用于现代商业航天的发展模式,主要是因为抗辐射加固会大大增重,影响有效载荷重量,三模冗余无法解决多位翻转问题,而专用冗余芯片无法满足周期短成本低的要求,因此需要构建一种针对 MBU 的容错码解决方案,占用较少资源的同时提升纠错能力,从而达到计算机存储单元加固目的。

1 建立容错模型

因为空间高能粒子具有较大离散性和较低的密度,所以单粒子多位翻转除特殊情况外(例如遭遇太阳磁暴等宇宙环境,粒子密度短时间内骤增,导致高密度粒子流或多个粒子同时击中多个存储单元)一般指单个高能粒子造成

收稿日期:2020-04-20; **修回日期**:2020-05-06。

作者简介:龚 博(1984-),男,黑龙江哈尔滨人,硕士,工程师,主要从事运载火箭电气系统方向的研究。

多个存储单元翻转，这种情况的发生主要依靠两种方式，第一种为注入塌陷效应，指高能粒子以小入射角垂直于存储单元入射，产生的电荷扩散到互相相邻的位置，由于电荷的共享效应，造成邻位多位翻转，例如图 1 中的第二类和第三类。第二种为击穿效应，高能粒子以大入射角水平入射轰击并穿过多个存储单元造成多位翻转，例如图 1 中的第一类和第四类，均造成了相邻位逻辑翻转[6]；传统上认为由于能量限制第一类较多，而随着存储单元核心电压降低第二类同样值得重视。图 1 是存储单元多位翻转的矩阵示意图，水平方向为存储单元排列方向，垂直方向为存储单元照射方向。

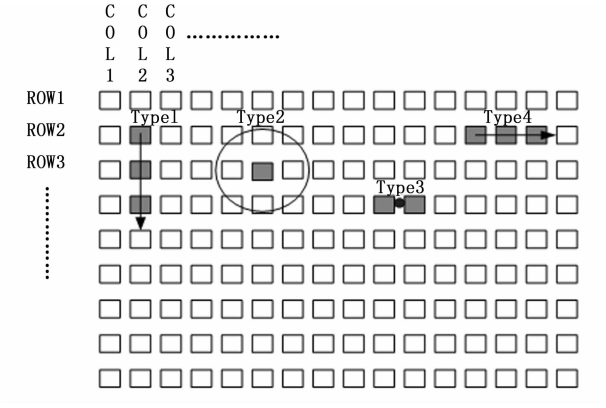

图 1　单粒子多位翻转示意图

目前市场上商用器件除部分采用物理临位和逻辑临位分开工艺的器件外，大多数存储器件都容易收到单粒子效应影响，因此需要通过试验模拟高能粒子轰击存储单元来建立容错模型，图 2 是几组对比试验[7-9]，能量单位为 MeV，取 4 个档位，并运用大量样本分析统计数据，从试验结果我们可以得出当用不同能量的质子轰击 SRAM 存储单元时可以造成大量的多位翻转，从存储矩阵的横列和纵列来看多位翻转几乎全部是相邻两位，相邻三位虽然存在但数量较少，四位及四位以上几乎可以忽略不计，由此可以得出模型中重要能力：容错能力要求在保持纠正 1 位检测两位错误的基础上能纠相邻 2bit 和 3bit 错误。

照射能量	两位翻转 三种错误样式			三位翻转 五种错误样式					四位翻转 五种错误样式				
22	773	136	80	920	34	15	0	3	726	57	66	47	38
47	681	180	117	861	62	27	13	10	621	79	103	84	42
95	653	192	132	792	79	40	26	23	482	154	109	99	41
144	686	156	133	799	62	44	23	16	455	114	136	98	49

图 2　单粒子多位翻转测试结果

容错模型的容错能力提出后还需要找到合适的容错方法，抗辐照加固、专用芯片前文已经提过不适应于现在的发展状况，我们主要考虑硬件冗余和信息冗余两种方法，硬件冗余中常见为三模冗余、多模冗余、逻辑冗余等，由于前文得出需要纠 2bit、3bit 的能力，经典三模冗余无法完成，多模冗余和逻辑冗余完成需要构建大量重复逻辑资源，造成体积、重量、成本等增加同样不适合于商用航天，另一种方案是信息冗余，信息冗余是通过某种算法（既冗余位和有效位的对应关系）来对信息位进行校验和纠错，信息冗余的优点是冗余度相对较少，所占资源相对较少，适用于本文需要解决的问题，但传统的以海明码为算法的 EDAC（error detection and correction）算法同样不具有纠 2bit、3bit 的能力，不使用专用芯片的情况下需要构建一个算法来完成信息冗余方案。

2　构建信息冗余算法

信息冗余算法指在发送端按照某种特性构造增加一些冗余位，当数据有效部分和冗余部分因外界故障注入时发生错误，接收端按照同样的构造特性进行计算和比对，这种构造规则即为冗余算法，也就是常说的纠错码。由上文建模结果可知，我们需要一种抗 MBU 多位翻转的纠错算法。

因考虑工程实现的简便性，本文所讨论的编码算法均为二进制码，设输入的每组信息码的码元为 k，经过编码后的每组码的码元为 n，其中 $n>k$，通常用（n，k）来表示。要构造编码算法首先要确定编码类型，这里我们需要选择最简单的线性编码，即在有效的 8 位或 16 位数据后面通过逻辑加法增加冗余位，线性编码能最大的节省寄存器资源，否则采用复杂编码算法就需要专门开辟寄存器存储区域来专门运行和存储计算结果，不但提高了成本和资源占用且会造成计算和存储的时间延迟。

由上文纠错模型可知，传统基于海明码和扩展海明码的 EDAC 算法纠错能力不足，无法应对商用器件空间环境下的 MBU，需要我们重新构造编码算法，在线性编码的条件下完成纠错能力的提高，从工程实现和理论分析两个角度出发必须满足以下几个条件：首先，此纠错码必须为系统码，即保持有效位不变，也就是说从工程实现的角度来考虑有效的 8 位或 16 位在编码译码的过程中保持不变，减小工程实现难度；第二，从编码原理上分析，提高纠错能力必然意味着码距的增加，但通过上文建模可知，多位翻转并不是随机的，而是在存储阵列纵横的邻位翻转，这样我们只需要解决邻位翻转问题即可，码距增加的越小，编码的复杂程度越低，编码和解码所占用的资源和时间越少，越有利于工程实现；第三，此算法需要具有（n，8）的结构，因为现在工程上信息位基本均为 8 的倍数，使用（n，8）的结构可以最大程度增加冗余而又不影响工程上 8 位、16 位、32 位数据的使用，否则数据拼接这一项额外环节就会对系统造成额外负担。

现假设信息码原码为 $\boldsymbol{U}$，编码后的监督码为 $\boldsymbol{C}$，则 $\boldsymbol{C}$ 与

$\boldsymbol{U}$ 的关系可以描述为：

$$\boldsymbol{C}=\boldsymbol{U}\cdot\boldsymbol{G} \tag{1}$$

$\boldsymbol{G}$ 即为生成矩阵，解码时公式如下：

$$\boldsymbol{H}\cdot\boldsymbol{C}^{T}=0 \tag{2}$$

进行解码时 $\boldsymbol{H}$ 矩阵就叫做校验矩阵，也叫做监督矩阵，校验矩阵的各行是线性无关的，$\boldsymbol{H}$ 矩阵与生成矩阵的关系如下：

$$\boldsymbol{G}\cdot\boldsymbol{H}^{T}=0 \tag{3}$$

也就是所给定纠错码的情况下，编码和解码的线性关系就唯一确定了。由上可知 $\boldsymbol{H}$ 矩阵不仅确定了信息码和冗余码之间的关系，也决定了码的距离特性，实际上我们就是在用 $\boldsymbol{H}$ 矩阵在 2^{n} 的码字中进行筛选。

设 R 是接收端接收到的码字，此时 R 中已经含有错误图样，仍然用校验矩阵 $\boldsymbol{H}$ 来检查码字是否满足校验方程，设编码码字 $\boldsymbol{C}=(c_6,c_5,c_4,c_3,c_2,c_1,c_0)$，错误码字为 $\boldsymbol{E}=(e_6,e_5,e_4,e_3,e_2,e_1,e_0)$，则 $\boldsymbol{R}=\boldsymbol{C}+\boldsymbol{E}$。收到 $\boldsymbol{R}$ 后用 $\boldsymbol{H}$ 矩阵进行校验，如下所示：

$$\boldsymbol{H}\cdot\boldsymbol{R}^{T}=\boldsymbol{H}\cdot(\boldsymbol{C}+\boldsymbol{E})^{T}=\boldsymbol{H}\cdot\boldsymbol{C}^{T}+\boldsymbol{H}\cdot\boldsymbol{E}^{T} \tag{4}$$

由方程（2）可以知道 $\boldsymbol{H}\cdot\boldsymbol{C}^{T}=0$，带入方程（4）得 $\boldsymbol{H}\cdot\boldsymbol{R}^{T}=\boldsymbol{H}\cdot\boldsymbol{E}^{T}$，设 $\boldsymbol{S}$ 矩阵为：

$$\boldsymbol{S}^{T}=\boldsymbol{H}\cdot\boldsymbol{R}^{T}=\boldsymbol{H}\cdot\boldsymbol{E}^{T} \tag{5}$$

则 $\boldsymbol{S}$ 就被称为伴随式或校验子，根据校正子和错误图样对应关系表查找对应错误。

本文在一种纠正相邻错误（15，9）的循环码的基础上进行改进[10-12]，这种码是系统码和线性码，可以纠正相邻 2 位和 3 位错误，这种（15，9）的循环码的生成多项式为 $g(x)=x^{6}+x^{5}+x^{4}+x^{3}+1$，它的校验矩阵 $\boldsymbol{H}$ 如矩阵（6）所示。

$$\boldsymbol{H}=\begin{bmatrix}1&0&0&1&1&1&0&0&1&1&0&0&0&0&0\\1&1&0&1&0&0&1&0&1&0&1&0&0&0&0\\1&1&1&1&0&1&0&1&1&0&0&1&0&0&0\\1&1&1&0&0&1&1&0&0&0&0&0&1&0&0\\0&1&1&1&0&0&1&1&0&0&0&0&0&1&0\\0&0&1&1&1&0&0&1&1&0&0&0&0&0&1\end{bmatrix} \tag{6}$$

对比上文几个必要条件我们得出：此码为 2 进制线性码，具备工程实现方便的特点，此码码距为 3，虽然能纠正邻位错位，却不具备检测任意 2 位错位的能力，需要重新构造，增加其码距，提高其容错能力，同时，此码不满足（n，8）结构，我们也需要对其重新构造，使其符合我们计算机常用存储单元数据位要求，这里以最基本的 8 位为例，其他 8 的倍数依此类推。

我们首先对其进行结构上的改进，删除第九列，码字变为（14，8）结构，$\boldsymbol{H}$ 矩阵变为矩阵（7），使其满足我们结构上（n，8）的要求，但是其码距仍然为 3，通过计算要检查任意 2 位错位同时保持纠正邻位错误码距至少为 4，考虑到工程实现上的复杂程度，考虑提高码距为 4。

$$\boldsymbol{H}=\begin{bmatrix}1&0&0&1&1&1&0&0&1&0&0&0&0&0\\1&1&0&1&0&0&1&0&0&1&0&0&0&0\\1&1&1&1&0&1&0&1&0&0&1&0&0&0\\1&1&1&0&0&1&1&0&0&0&0&1&0&0\\0&1&1&1&0&0&1&1&0&0&0&0&1&0\\0&0&1&1&1&0&0&1&0&0&0&0&0&1\end{bmatrix} \tag{7}$$

原码字数据 *D7D6D5D4D3D2D1D0* 共计 8 位，根据校验矩阵（7）得出增加冗余码后码字的长度变为 14 位，设为 *D7D6D5D4D3D2D1D0C5C4C3C2C1C0*，其中 *C5C4C3C2C1C0* 为冗余码，通过公式 $\boldsymbol{C}\cdot\boldsymbol{H}^{T}=0$ 来约束，同时设 $\boldsymbol{S}$ 为伴随式，共计六位，为 S5S4S3S2S1S0，我们通过公式 $\boldsymbol{S}=\boldsymbol{E}\cdot\boldsymbol{H}^{T}$ 得出伴随式 $\boldsymbol{S}$，计算结果可知伴随式不是一一对应，重叠情况如表 1 所示。

表 1 改进校验矩阵对应的伴随式重叠表

伴随式值	伴随式重叠	重叠个数及位数
000100	C2 D50	一位/任意两位
000111	C210 D74	连续三位/任意两位
001000	C3 D61	一位/任意两位
010000	C4 D72	一位/任意两位
010001	C40 D65	任意两位/相邻两位
010101	D543 D60	相邻三位/任意两位
011101	D73 D10	任意两位/相邻两位
100001	C50 D3	一位/任意两位
100010	C51 D76	任意两位/相邻两位
101010	D654 D71 D30	相邻三位/任意两位
101101	D765 D41	相邻三位/任意两位
110000	C54 D40	任意两位/相邻两位
110111	D31 D70	任意两位/任意两位

伴随式共 6 位，含义如下：C2 表示 C2 位单个错误，D50 表示 D5 和 D0 位都出现错误，D210 表示 D2、D1、D0 三位邻位错误，而 C2 和 D50 对应同一个伴随式值，表示伴随式重叠，即纠错码无法分辨这两种错误，同理 D654、D71 和 D30 表示三种错误对应同一个伴随式无法完成分辨和纠错。

通过上表我们可以看到我们仍然面临以下几个问题：1）部分错误使用同一伴随式，这样就会使我们在工程计算上分辨不出原码或冗余码的故障来源于哪种错误，从而无法就行纠错；2）工程上实现的意义仅在于纠正或检验出原码的正确与否，而冗余码出错也可以忽略，但此伴随式冗余码却占用者一定资源。

为解决以上两个问题，我们进一步分析可以发现 D31D70 这样的错误虽然是两位错误重叠，但我们仅仅要求检测两位，纠正相邻两位和三位，而不要求纠正任意 2 位，所以 D31D70 这样的错误我们工程上不关心可以忽略不计，同时我们发现所有重叠的错误中由于矩阵本身的性质导致均为奇偶重叠，即一位错误与两位错误伴随式相同，两位

错误和三位错误伴随式相同，并没有一位错误与三位错误伴随式相同的情况，因此我们很自然想到在重叠的基础上用奇偶来区分，增加一列奇偶区分列，可以使码距由 3 变为 4，满足检测任意两位的同时也保持了纠正任意一位同时纠正相邻两位、三位错误的能力，因此我们调整后矩阵 $\boldsymbol{H}$ 如矩阵（8）所示。

$$\boldsymbol{H}=\begin{bmatrix}1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\1&1&0&1&1&1&0&1&1&0&0&0&0&0&0\\1&0&0&1&0&0&1&0&0&1&0&0&0&0&0\\1&1&1&1&0&1&0&1&0&0&1&0&0&0&0\\1&0&1&0&0&1&1&0&0&0&0&1&0&0&0\\0&1&1&1&0&0&1&1&0&0&0&0&1&0&0\\0&0&1&1&1&0&0&1&0&0&0&0&0&1&0\end{bmatrix} \tag{8}$$

第二次改进矩阵求出的伴随式重叠统计如表 2 所示。从表中分析我们得出任意一位、相邻两位和相邻三位都不重叠，且任意两位可以检测，因此我们得出结论：改进后的循环码保证纠正 1 位检测 2 位错误的基础上针对 MBU 的错误模式满足了纠正相邻 2 位和 3 位的能力的要求。

表 2　改进校验矩阵对应的伴随式重叠表

伴随式值	伴随式重叠	重叠个数及位数
0000100	D60 C20	任意两位/任意两位
0001010	C31 D62	任意两位/任意两位
0001110	D74 D20	任意两位/任意两位
0010000	D63 C40	任意两位/任意两位
0010100	C42 D30	任意两位/任意两位
0100000	D72 D40 C50	任意两位/任意两位
0100100	C52 D64	任意两位/任意两位
0101000	C53 D50	任意两位/任意两位
0101110	D42 D70	任意两位/任意两位

3　工程设计与实现

工程上我们可以根据改进后的 $\boldsymbol{H}$ 矩阵求出编码方程和解码的伴随式方程。电路实现时所需要的逻辑运算都是三种基本运加逻辑异或运算，这种逻辑电路通过可编程逻辑器件来实现，且可以根据不同的算法做出相应修改，具有很好的灵活性。以 8bit 为例得出编码计算公式和伴随式计算公式分别如公式（9）和公式（10）所示。

$$\begin{aligned}
C6&=D7\oplus D6\oplus D4\oplus D3\oplus D2\oplus D0\\
C5&=D7\oplus D4\oplus D1\\
C4&=D7\oplus D6\oplus D5\oplus D4\oplus D2\oplus D0\\
C3&=D7\oplus D5\oplus D2\oplus D1\\
C2&=D5\oplus D4\oplus D1\oplus D0\\
C1&=D6\oplus D5\oplus D4\oplus D3\oplus D0\\
C0&=C6\oplus C5\oplus C4\oplus C3\oplus C2\oplus C1\oplus\\
&\quad D7\oplus D6\oplus D5\oplus D4\oplus D3\oplus D2\oplus D1\oplus D0
\end{aligned} \tag{9}$$

$$\begin{aligned}
S6&=C6\oplus D7\oplus D6\oplus D4\oplus D3\oplus D2\oplus D0\\
S5&=C5\oplus D7\oplus D4\oplus D1\\
S4&=C4\oplus D7\oplus D6\oplus D5\oplus D4\oplus D2\oplus D0\\
S3&=C3\oplus D7\oplus D5\oplus D2\oplus D1\\
S2&=C2\oplus D5\oplus D4\oplus D1\oplus D0\\
S1&=C1\oplus D6\oplus D5\oplus D4\oplus D3\oplus D0\\
S0&=C0\oplus C6\oplus C5\oplus C4\oplus C3\oplus C2\oplus C1\oplus\\
&\quad D7\oplus D6\oplus D5\oplus D4\oplus D3\oplus D2\oplus D1\oplus D0
\end{aligned} \tag{10}$$

EDAC 容错原理图如图 3 所示，在 CPU 和存储器之间增加 EDAC 模块，由可编程逻辑器件 FPGA 实现，处理器和存储器之间除了地址总线相连外其他控制信号和数据信号均通过 EDAC 模块处理后再发给存储器，即对存储器的读写均由 FPGA 完成。

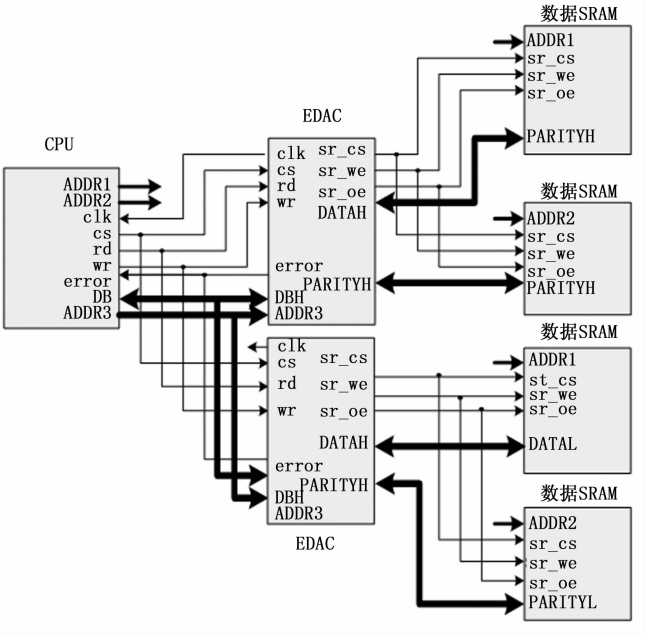

图 3　EDAC 模块设计原理图

两片存储器芯片分为数据 RAM 和校验 RAM，分别用来存放数据码和编码模块生成的冗余码。其中 addr1、addr2 分别连接数据和校验 SRAM，addr3 连接 FPGA 实现 EDAC 模块，数据 16 位分为高 8 位和低 8 位，DATAH 为高 8 位数据原码，PARITYH 为高 8 位数据校验码，低 8 位同理。EDAC 模块被分为平行的两部分，一部分对高 8 位数据进行校验，一部分对低 8 位数据进行校验，两部分数据位和校验位存储到四片 SRAM 中，4 片 SRAM 的地址都由 CPU 统一控制，FPGA 只对存储器进行数据的读写。

EDAC 实现模块内部结构图如图 4 所示，其中 EDAC 模块负责连接 CPU 和 SRAM 之间的数据流，对于 CPU 来说 SRAM 是透明的，当片选信号有效时，写信号过程如下：wr/rd 读写控制信号置 1，三态门 A 打开，CPU 发出数据经过三态门进入编码器，生成纠错码，此时多路选择器打开，数据码和纠错码通过三态门 B 写入存储器。当片选信号有效时，读信号过程如下，wr/rd 读写控制信号置 0，三

态门 B 打开，需要校验的数据由存储器进入锁存器，由 FPGA 内部的时序控制器发出锁存信号 sr _ en 将数据锁存，纠错模块将锁存的数据就行校验，并在读过程结束时由时序控制器发出是否有错的信号 error，如果没有错误将数据通过三态门 A 送到 CPU，如果有错将多路选择器打，将纠正后的数据通过三态门 B 回写入存储器。

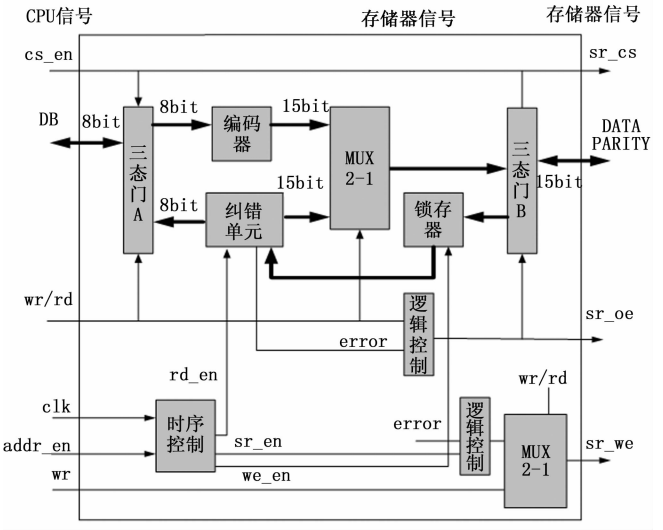

图 4 EDAC 内部结构图

时序控制器产生纠错结束标志信号 cr _ over，锁存标志信号 sr _ en，写控制信号 we，具体操作如下，首先是锁存信号 sr _ en 的产生，根据 CPU 数据手册，CPU 地址总线使能最大延迟为 20 ns，片选使能延迟也约为 20 ns，而本系统中片选信号通过 FPGA 还要有约为 5 ns 的延迟，而从地址有效到数据有效的时间约为 15 ns，从外部通过三态门 B 到锁存器的延迟约为 5 ns，一共延迟约为 20＋5＋15＋5＝45 ns，加上一些余量锁存时间确定为 60 ns 可以确保采到有效数据，纠错模块的逻辑电路延迟约为 12 ns，取 80 ns 可以确定纠错完成时间，cr _ over 信号置 1，如果发生错误数据还要回写至存储器，存储器最小写入时间约为 18 ns，加上通过三态门的约为 5 ns 的延迟，约为 80＋18＋5＝103 ns，取 120 ns 为最后结束时间，所以 we 信号从 80 ns 至 120 置 1，其他时间置 0。

4 测试与仿真结果

本节将改进型循环码、扩展海明码以及准循环码进行仿真、测试和结果比较，比较他们的纠错能力和消耗的资源。

扩展的海明码、准循环码、改进的循环码功能仿真图如图 5、图 6、图 7 所示。clk 是系统时钟，clkinsert 是采样时钟，是验证环节错误注入需要的，cstate 和 nstate 为两段式状态机的两个状态，din 为输入数据，paritycode 为编码数据，parityerror 是软件注入的错误，s 代表生成的伴随式，dataout 为输出，e 为最后检查出的错误，warnning 为不能解决错误向 CPU 发出的中断请求信号。

三种算法的纠错能力和资源消耗的总结如表 3 所示，

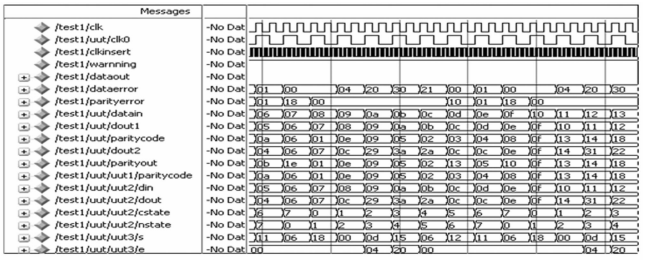

图 5 扩展的海明码编码解码仿真图

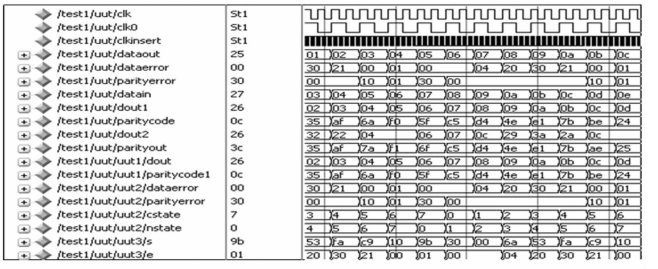

图 6 准循环码编码解码仿真图

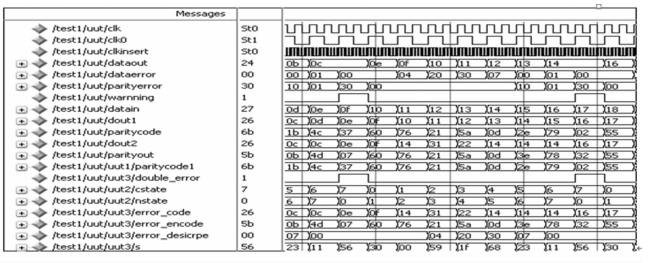

图 7 改进型循环码编码解码仿真图

其中实现资源中的 S 指 Xilinx FPGA 基本单元 Slice，L 指 4 输入查找表 LUT，此表以 8bit 数据为例，所以存储资源 1byte 用来存放数据码，1byte 用来存放冗余码，三种算法均为 2byte，而时间延迟为布局布线加上时序约束后仿真软件的报告中给出的可能的最长的时间延迟路径。通过表中比较我们可以看到三种算法可靠度都非常高也都比较接近，而我们是通过 FPGA 并行实现的编码，所以码率在工程实

表 3 三种算法纠错能力与资源消耗比较比较

	纠错能力	可靠度	码率	实现资源		存储资源	时间延迟	
				编码	解码		编码	解码
扩展海明码	纠一检二	0.9995	0.62	4S/7L	26S/50L	2byte	5.531ns	9.079ns
准循环码	任意两位	0.9999	0.5	4S/8L	74S/129L	2byte	5.531ns	11.854ns
改进循环码	纠一检二邻位纠错	0.99995	0.53	6S/11L	56S/104L	2byte	4.283ns	16.109ns

(下转第 264 页)