# Design of NAND Flash Controller Based on NB-LDPC ECC

**Article** *in* Open Access Library Journal · January 2021

DOI: 10.4236/oalib.preprints.1200293



**9 authors**, including:

Tingzhen Liu
Tencent
**20** PUBLICATIONS **10** CITATIONS

SEE PROFILE

# Design of NAND Flash Controller Based on NB-LDPC ECC

**Tingzhen Liu[1], Haozhe Yu[1], Guanhui Zhang[1], Yuge Li[1], Mingkai Zhou[1], Ziyi Zhang[2], Xiaoning Xin[1], Siyuan Liu[1], Jian Ren[1]**

[1]College of information science and engineering, Shenyang University of Technology, Shenyang, China
[2]College of electronic engineering and optoelectronic technology, Nanjing University of technology, Nanjing, China
Email: firstsg@outlook.com

## Abstract

**NAND flash, as the mainstream storage medium, has the advantages of high performance, high density, non-volatile and low power consumption. However, due to the special internal structure of NAND flash, it is necessary to design a special controller for data management, which is suitable for all kinds of storage systems. In this paper, a NAND flash controller based on NB-LDPC is completed. We will describe the construction of LDPC code, the implementation of encoding circuit, the realization of decoding circuit and the connection with interface. We run simulations and simulations through C++, Verilog, and Maltab. Part of the performance of the module is tested. The size of LDPC check matrix is 9210*1017 and the code rate is 0.89. The encoding and decoding circuits of the controller are all implemented and have ECC function.**

## Keywords

**NAND-Flash, NB-LDPC, Integrated Circuit, Communication Verification**

**Subject Areas: Communication Protocol, Communication Theory and Algorithm, Signal Processing, Electronic Engineering**



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

## 1. 引言

随着大数据时代的来临，数据存储需求量剧增。NAND Flash 作为目前主流存储介质，具有高性能、高密度、非易失性和低功耗等优点。然而，NAND Flash 特殊的内部结构，决定了需要设计专门的控制器进行数据管理，以适用于各类存储系统[1]。LDPC(低密度奇偶校验码)是一种纠错性能优越的前向纠错编码(FEC)，已经被提出应用于NAND Flash的ECC技术中。本文的工作完成了一种基于NB-LDPC的NAND flash控制器。在这篇论文中，我们将对 LDPC 码的构造、编码电路的实现、解码电路的实现和与接口连接分别进行描述。我们通过 C++、Verilog、和 Maltab 运行模拟和仿真。测试所设计的模块的部分性能。构造的 LDPC 码是 9210*1017 的矩阵，码率为 0。89。控制器的编码解码电路均被实现，具有 ECC 功能。

## 2. LDPC码的构造

### 2.1. NB-LDPC说明

根据伽罗华域运算规则，设计出非二进制低密度奇偶校验码（NB-LDPC），它与二进制 LDPC 码非常相似，只是在消息传递时需要乘或者除一个系数。比如 GF(8)域的校验矩阵，就是二进制 LDPC 校验矩阵中的 1，随机的用 GF(8)中的符号代替，可以得到表达式（0-1）

$$H_2 = \begin{bmatrix} 0 & h_{12} & h_{13} & h_{14} & 0 & 0 \\ h_{21} & 0 & 0 & h_{24} & h_{25} & 0 \\ 0 & 0 & h_{33} & 0 & h_{35} & h_{36} \\ h_{41} & h_{42} & 0 & 0 & 0 & h_{46} \end{bmatrix} \quad \backslash* \text{ MERGEFORMAT (0-1)}$$

H2 中的 $h_{ij}$ 是 GF(8)上的非 0 符号，十进制表示为 1-7。GF(8)的 Tanner 图也是与 H 矩阵的唯一对应的，但每个 V 与 C 之间的连接都是两条线，消息从 V 到 C 要乘以 $h_{ij}$，从 C 到 V 要除以 $h_{ij}$，或者说乘以 $h_{ij}^{-1}$，见图 2.1：

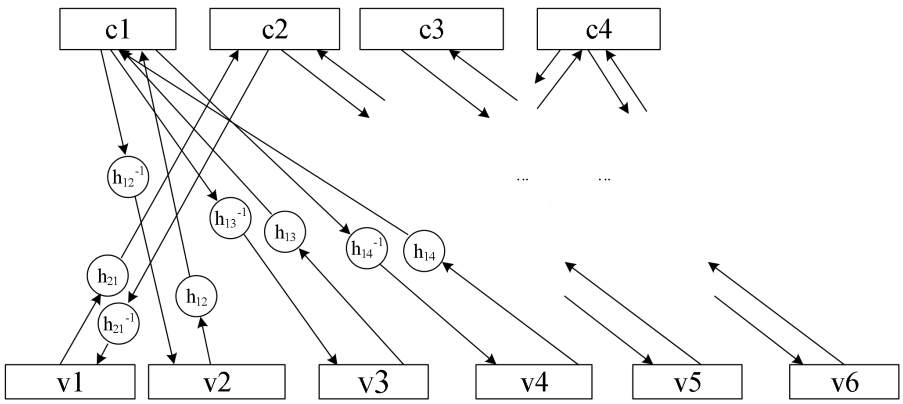



**Figure 2.1.**Tanner图.

### 2.2. 伽罗华域运算规则



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

伽罗华域运算是有限域的一种，属于抽象代数，是编码学的基础。一个有限域定义了有限个符号和加法、乘法运算规则，特征是任何两个符号做加法和乘法运算后的结果仍然是这个符号集中的符号，也就是说没有进位。

$GF(2^n)$域定义了 $2^n$ 个符号，例如 $GF(8)$，也就是 $GF(2^3)$定义了 8 个抽象符号，这 8 个符号有多种表示法。首先是二进制表示法，即用 000−111 表示。与二进制对应的是多项式表示法。在 $GF(8)$中，最左边的 1 对应 $x^2$，中间的 1 对应 $x$，最右边的 1 就对应 1，例如 101 就对应 $x^2+1$ 除上述表示法外，$GF(2^n)$域的符号还有指数表示法。$GF(2^n)$域中定义了两个特殊符号即 0 和 1，“0”称为加法恒元，即任何符号加 0 后不变，“1”称为乘法恒元，即任何符号与 1 相乘后不变，这其实与普通代数是一样的，其它符号可表示为 $a$，$a^2$，$a^3$，$a^4$，$a^5$ 和 $a^6$。$GF(2^n)$域的加法就是对应二进制表示的按位异或运算，例如 110+100=010。按这种规则，任何两个相同符号相加为 0。乘法运算需要根据多项式来运算，当结果超出定义范围时要通过本元多项式来解决，$GF(8)$域的本元多项式为

$$P(x)=x^3+x+1=0 \qquad \backslash* \text{ MERGEFORMAT (0-2)}$$

其中 0，1，$a$ 与多项式和二进制的关系属于定义，具体如表 2.1：

**Table 2.1. GF(8)符号前 3 项**

| 指数表示 | 多项式表示 | 二进制表示 |
|---|---|---|
| 0 | 0 | 000 |
| 1 | 1 | 001 |
| $a$ | $x$ | 010 |

从 $a^2$ 开始可以推导出来，既然是指数表示，就应保保持乘法关系，例如 $a^2$ 应该等于 $a \times a$，根据这种关系就可以推出后面符号对应的表示法。

$a^2 = a \times a = x \times x = x^2$，由于 $x^2$ 仍然在 $GF(8)$符号范围内，因此对应多项式就是 $x^2$，按二进制与多项式的关系，对应二进制为 110。$a^3 = a^2 \times a = x^2 \times x = x^3$，按二进制与多项式的关系，表示 $x^3$ 已经需要 4 位数，而 $GF(8)$得二进制只有 3 位，这时就要用到本元多项式了，根据公式（2-5），$x^3 = x + 1$，所以可以写出 $a^3$ 对应得多项式为 $x+1$，对应二进制为 011。于是可以得到一个完整的 $GF(8)$域符号表，见表 2.2：

**Table 2.2. GF(8)域符号表**

| 指数表示 | 多项式表示 | 二进制表示 | 十进制表示 |
|---|---|---|---|
| 0 | 0 | 000 | 0 |
| 1 | 1 | 001 | 1 |
| $a$ | $x$ | 010 | 2 |
| $a^2$ | $x^2$ | 100 | 4 |

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

| | | | |
|---|---|---|---|
| $a^3$ | $x+1$ | 011 | 3 |
| $a^4$ | $x^2+x$ | 110 | 6 |
| $a^5$ | $x^2+x+1$ | 111 | 7 |
| $a^6$ | $x^2+1$ | 101 | 5 |

GF(8)域乘法运算规则就是一句话，先按多项式相乘，再用本元多项式解决溢出问题。任何数乘 1 不变，乘 0 为 0 这两条依然成立，所以从2×2开始讨论。2×2相当于 $x \times x = x^2$，而 $x^2$ 对应 100，进而 $2\times 2=4$；同理，$2\times 3 = x\times(x+1)$，对应 110=6，以此类推，可以写一个“77 乘法表”见表 2.3：

**Table 2.3. 77 乘法表**

| | | | | | |
|---|---|---|---|---|---|
| 2×2=4 | 2×3=6 | 2×4=3 | 2×5=1 | 2×6=7 | 2×7=5 |
| 3×2=6 | 3×3=5 | 3×4=7 | 3×5=4 | 3×6=1 | 3×7=2 |
| 4×2=3 | 4×3=7 | 4×4=6 | 4×5=2 | 4×6=5 | 4×7=1 |
| 5×2=1 | 5×3=4 | 5×4=2 | 5×5=7 | 5×6=3 | 5×7=6 |
| 6×2=7 | 6×3=1 | 6×4=5 | 6×5=3 | 6×6=2 | 6×7=4 |
| 7×2=5 | 7×3=2 | 7×4=1 | 7×5=6 | 7×6=4 | 7×7=3 |

## 2.3. 校验矩阵构造

以最小化四环数目为准则构造校验矩阵。首先造数个 n×n 对角阵构成的小矩阵（对角元素随机），将其以每行 r 个，每列 c 个放置，构成大矩阵 H。示意图如图 2-2 所示：

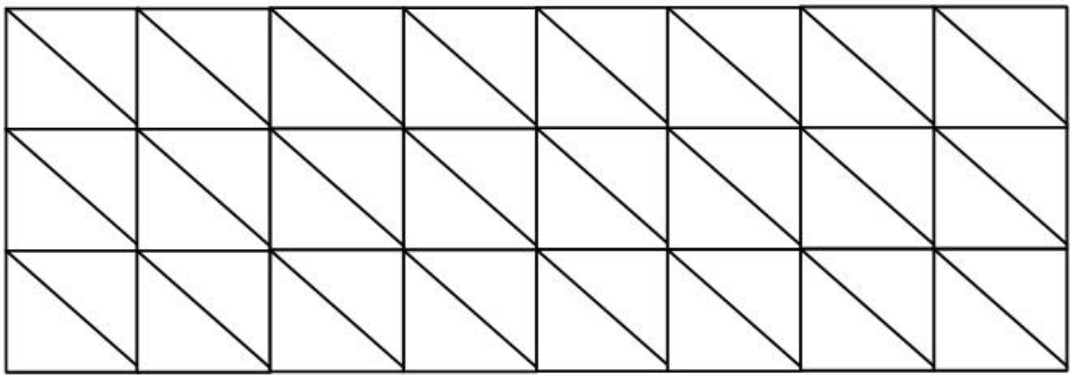


**Figure 2.2.**示意图.

本文的校验矩阵基于大矩阵的循环右移得到，即需循环进行如下步骤：

1. 随机选定 i 个小矩阵。
2. 对每个选定的小矩阵，随机循环右移 j 次（循环右移即为将矩阵中每个非零元素移动到右边相邻位置。如果该元素已经在最右，那么回到第一个位置）。

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

3. 检查 H 的四环数量，如果是目前发现最少的，暂存这个矩阵。

循环的停止条件视情况而定。在本文的实验中，H 列数为 256 时，可以很快找到四环数为 0 的校验矩阵。因此停止条件设为四环数量为 0。

具体实施中，基于本文编写的校验矩阵生成器类构造对象，就可以开始生成编码矩阵。构造后的对象中，小矩阵对角元素均为 1。这样初始化的原因是，有人认为先减少 GF(2)矩阵的四环个数，再将对角元素随机替换为实际使用的 GF(n)元素，效果会更好。这么做基于的考虑是，可能存在某些向 GF(n)的某些替换无法使得编码矩阵达到最小的四环数量，因此可以基于优化后的 GF(2)进行几种不同的随机替换分别进行优化，选择最好的。但在我们的实验中，H 列数为 256 时，（我们所尝试的）所有替换都可以很快找到四环数为 0 的编码矩阵。因此直接进行向 GF(n)的替换也是可以的，而且这样做会大大加快优化速度。

为了方便构造 GF(8)的校验矩阵，本文构造的校验矩阵信息位 8193bit，冗余位 1017 位，由于规定信息位 8192bit，冗余位 1024bit。所以信息位最后一位补 0， 冗余位减 7 位。最后设计出 339*3070 的 GF（8）校验矩阵。二进制规模即为 9210*1017。

## 3. 编码设计

从矩阵的性质我们可以知道对矩阵进行任意行置换和列置换并不会改变其中的元素值，对于 LDPC 码的校验矩阵来说它的围长也不会因此改变。为了保证校验矩阵的稀疏性，我们可以重新排列校验矩阵中的行和列，得到一个近似于下三角式的矩阵，如图 3-2 所示，整个校验矩阵 $H$ 分成 6 个分块矩阵，可以表示成如下式所示的形式：

$$H=\begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \qquad \text{* MERGEFORMAT (0-3)}$$

式（3-2）中，$A$、$B$、$C$、$D$、$E$ 依次是 $(M-g)\times(N-M)$ g、$(M-g)\times g$ 、$g\times(N-M)$、$g\times g$ 和 $g\times(M-g)$ 阶稀疏矩阵，而 $T$ 是 $(M-g)\times(M-g)$ 阶的下三角稀疏矩阵。

设码字 $C=[s,p1,p2]$，$s$ 代表信息比特序列，$p1$ 代表长度为 $g$ 的校验位，$p2$ 代表长度为($M$-$g$)的校验位。在 $HC^T=0$ 两端分别乘以 $\begin{bmatrix} 1 & 0 \\ -ET^{-1} & 1 \end{bmatrix}$，则得到

$$\begin{bmatrix} A & B & T \\ -ET^{-1}A+C & -ET^{-1}B+D & 0 \end{bmatrix}\begin{bmatrix} s \\ p1 \\ p2 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{* MERGEFORMAT (0-4)}$$

解出 $p1$，$p2$，令

$$\Phi=-ET^{-1}B+D \qquad \text{* MERGEFORMAT (0-5)}$$

显然 $|\Phi|\neq 0$，可得

$$p1^T=-\Phi(-ET^{-1}A+C)s^T \qquad \text{* MERGEFORMAT (0-6)}$$

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

$$p2^T = -T^{-1}(As^T + Bp1^T) \qquad \text{* MERGEFORMAT (0-7)}$$

实际硬件中直接保存计算 $p_1$ 和 $p_2$ 所需的中间矩阵结果，防止每次都进行重复计算。将上式展开可以得到需要保存的中间结果为 $\Phi^{-1} + ET^{-1} + C$ 、$T^{-1}A$ 和 $T^{-1}B$ 。拼接 $s$ 、$p_1$ 和 $p_2$ 和得到编码后的码字。用 C++语言得出中间结果分别用 X、L、M 矩阵，配合 Verilog 语言得出编码结果。

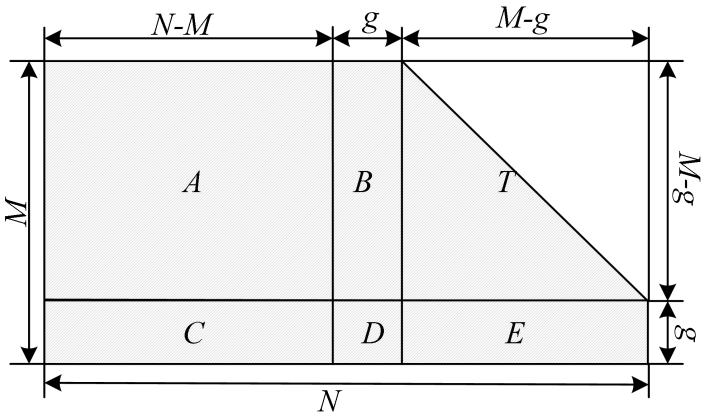



**Figure 3.2.**近似下三角矩阵的稀疏校验矩阵.

## 4. 电路设计

本文的电路设计，包括从数据输入开始，先进行编码运算，再进行解码运算最后再输出纠错后的数据。其中 LDPC 码解码器是基于硬判决实现的。通过运算简化，其实现的复杂度相比原算法大大下降。可以在此基础上更进一步优化改进，在设计 LDPC 码的 Verilog 实现时，必须合理的平衡硬件资源和解码速率，争取达到最优效果。

### 4.1. 解码算法的纠错原理

根据 Tanner 图可知，变量节点与校验节点之间的连线是由 *H* 矩阵决定的。例如，第 1 个校验节点 *C1* 与变量节点 *V1-V6* 的连接关系由 *H* 的第一行决定，只有在 *H* 矩阵中为 1 的位置所对应的变量节点与 *C1* 存在连接。Tanner 图中变量节点和校验节点的信号是双向的，从变量节点到校验节点的信号称为 *v2c* 消息，校验节点返回到变量节点的信息称为 *c2v* 消息。

变量节点负责接收数据和修改数据并发出 *v2c* 消息，节点负责汇总分发 *c2v* 消息。在简单的硬判决解码算法中，一个变量节点发送的 *v2c* 消息的含义是“我觉得这个节点的数据是 0（或 1）”，*c2v* 节点向某个变量节点发送的消息的含义是“别人认为你的数据应该是 1（或 0）”。如果，每个变量节点在接收到 *c2v* 消息后都发现“我觉得”与“别人觉得”是一致的，解码运算就结束了，当前保存在变量节点中的数据就是正确



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

数据。如果有某个或某几个变量节点接收到的 *c2v* 消息与自己原来“觉得”的不一致，那么，“反对票”最多的那个变量节点就要修改自己的数据，直到大家意见统一或达到最大迭代次数为止。

校验节点发给某个变量节点的 c2v 消息，是剔除了来自该节点本身的数据后的所有其它变量节点的数据之和。变量节点把自己当前推荐给校验节点的数据与 c2v 的数据相加（加表示异或）。如果为“0”，就表明与这个校验节点相关的其他变量节点同意自己的观点。这样每个变量节点与两个检验节点相连，会收到两条 c2v 消息，于是会有 3 种可能，即全票通过，1:1（一票赞成一票反对）和两票反对。然后一般让反对票最多的那个节点修改数据，其他节点暂时不改。修改后的数据再传给校验节点，校验节点重新汇总并分发消息，如此迭代，直到意见统一或达到最大迭代次数为止。

1） 校验

图 4-1 中的 $x_1$ 那一行是没有错误的数据，用 $x_{n:l}$ 表示第 $n$ 个接收数据的 l 位，则验证结果为：

$$c_1 = x_{1:2} \oplus x_{1:3} \oplus x_{1:4} = 1 \oplus 0 \oplus 1 = 0 \quad \text{* MERGEFORMAT (4-1)}$$

$$c_2 = x_{2:1} \oplus x_{2:4} \oplus x_{2:5} = 1 \oplus 1 \oplus 0 = 0 \quad \text{* MERGEFORMAT (4-2)}$$

$$c_3 = x_{3:3} \oplus x_{3:5} \oplus x_{3:6} = 0 \oplus 0 \oplus 0 = 0 \quad \text{* MERGEFORMAT (4-3)}$$

$$c_4 = x_{4:1} \oplus x_{4:2} \oplus x_{4:6} = 1 \oplus 1 \oplus 0 = 0 \quad \text{* MERGEFORMAT (4-4)}$$

解码器发现 $c$ 为零向量就停止了，仅实现了校验功能，纠错功能没有启动。

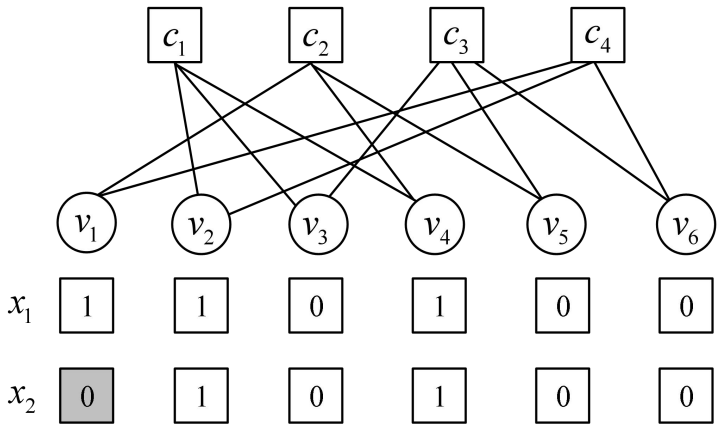



**Figure 4.1.H1的Tanner图.**

当接收到错误数据 $x_2$ 时，上面公式的 $c_2$ 和 $c_4$ 就不可能为 0 了，所以就启动纠错功能。定义变量节点 $j$ 发给校验节点 $i$ 的消息为 $v2c_{j:i}$，校验节点 $i$ 发给变量节点 $j$ 的消息用 $c2v_{i:j}$ 表示。在第一次迭代时，$v2c$ 消息就是接收到的数据。

2）纠错

校验节点 $c_1$ 返回的消息为：

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

$$c2v_{1:2}=v2c_{3:1}\oplus v2c_{4:1}=0\oplus 1=1$$
$$c2v_{1:3}=v2c_{2:1}\oplus v2c_{4:1}=1\oplus 1=0 \quad \text{* MERGEFORMAT (4-5)}$$
$$c2v_{1:4}=v2c_{2:1}\oplus v2c_{3:1}=1\oplus 0=1$$

校验节点 $c_2$ 返回的消息为：

$$c2v_{2:1}=v2c_{4:2}\oplus v2c_{5:2}=1\oplus 0=1$$
$$c2v_{2:4}=v2c_{1:2}\oplus v2c_{5:2}=0\oplus 0=0 \quad \text{* MERGEFORMAT (4-6)}$$
$$c2v_{2:5}=v2c_{1:2}\oplus v2c_{4:2}=0\oplus 1=1$$

校验节点 $c_3$ 返回的消息为：

$$c2v_{3:3}=v2c_{5:3}\oplus v2c_{6:3}=0\oplus 0=0$$
$$c2v_{3:5}=v2c_{3:3}\oplus v2c_{6:3}=0\oplus 0=0 \quad \text{* MERGEFORMAT (4-7)}$$
$$c2v_{3:6}=v2c_{3:3}\oplus v2c_{5:3}=0\oplus 0=0$$

校验节点 $c_4$ 返回的消息为：

$$c2v_{4:1}=v2c_{2:4}\oplus v2c_{6:4}=1\oplus 0=1$$
$$c2v_{4:2}=v2c_{1:4}\oplus v2c_{6:4}=0\oplus 0=0 \quad \text{* MERGEFORMAT (4-8)}$$
$$c2v_{4:6}=v2c_{1:4}\oplus v2c_{2:4}=0\oplus 1=1$$

将变量节点本身的数据与收到的信息对比，相同的为“赞同”票，不一样的为“反对”票。为了观察方便，详细信息见表 5.1。

**Table 4.1. 接收数据为 x2 时的 c2v 消息表**

| 变量节点 | 当前数据 | *c2v* 消息 1 | *c2v* 消息 2 | 含义 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 两票反对 |
| 2 | 1 | 1 | 0 | 1:1 |
| 3 | 0 | 0 | 0 | 两票赞同 |
| 4 | 1 | 1 | 0 | 1:1 |
| 5 | 0 | 1 | 0 | 1:1 |
| 6 | 0 | 1 | 0 | 1:1 |

成功的指出第一个变量节点是错误的，让 $v_1$ 把数据改为 1 即可。

### 4.2. LDPC模块

对数据输入到纠错到输出整个过程都通过 LDPC 模块实现的，它将编码解码模块连接在一起。外部端口如图 4-2 所示，端口信号的定义如表 4.2 所示。

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

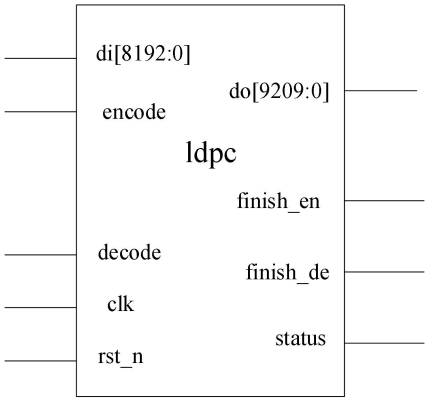



**Figure 4.2.LDPC模块.**

**Table 4.2. LDPC 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *di* | 信息位输入 | 8193 | 输入 |
| *encode* | 编码使能 | 1 | 输入 |
| *decode* | 解码使能 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *do* | 结果输出 | 9210 | 输出 |
| *finish_en* | 编码结束信号 | 1 | 输出 |
| *finish_de* | 解码结束信号 | 1 | 输出 |
| *status* | 解码成功信号 | 1 | 输出 |

模块 LDPC 的功能是，完成编码解码功能。准备好信息位后，LDPC 模块会接收到 *encode* 信号，然后开始编码，成功后 *finish_en* 信号为 1。LDPC 模块反馈后，会收到 *decode* 信号，然后开始解码运算，如果在最大迭代次数内成功纠错，则 *finish_de* 和 *status* 都为 1，如果没能改错，迭代次数满了后只会将 *finish_de* 置 1。输出的数据就是正确的结果。

## 4.3. 编码模块

编码模块的任务就是将输入的信息位，进行编码，得到校验位，并且合并全部输出。外部端口如图 4-3 所示，端口信号的定义如表 4.3 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

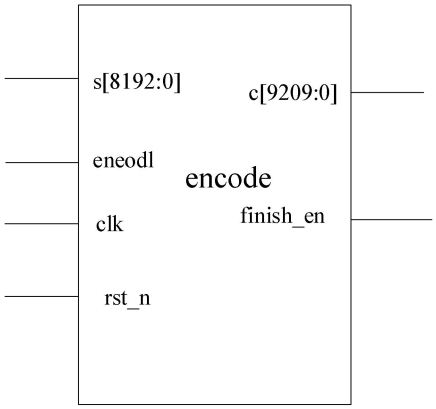



**Figure 4.3.encode模块.**

**Table 4.3. encode 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *s* | 信息位输入 | 8193 | 输入 |
| *encode* | 编码使能 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *c* | 结果输出 | 9210 | 输出 |
| *finish_en* | 编码结束信号 | 1 | 输出 |

模块 encode 的功能是，根据信息比特序列，得到信息校验位。输入 8193 位信息比特序列后，得到 *encode* 信号，就先算出 *p1*，然后算出 *p2*，最后整合输出数据，并输出 *finish_en* 表示解码成功。

### 4.4. 解码模块

解码模块任务是判断数据是否正确并改正，包含了 v2c2v 模块、控制模块和 3070 个 vn 模块。外部端口如图 4-4 所示，端口信号的定义如表 4.4 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

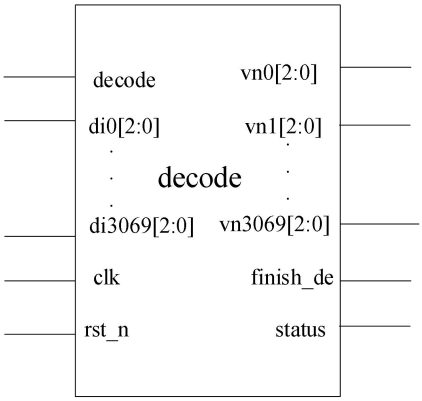



**Figure 4.4.decode模块.**

**Table 4.4. decode 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *di0* | 数据输入 | 3 | 输入 |
| … | … | … | … |
| *di3069* | 数据输入 | 3 | 输入 |
| *decode* | 解码使能 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *vn0* | 结果输出 | 3 | 输出 |
| … | … | … | … |
| *vn3069* | 结果输出 | 3 | 输出 |
| *finish_de* | 解码结束信号 | 1 | 输出 |
| *status* | 解码成功信号 | 1 | 输出 |

模块 decode 的功能是纠错。输入 3070 个数据后，decode 模块会收到 decode 信号，通过模块内部运算，如果最后输出的数据正确，则会将 finish_de 和 status 信号置 1，如果输出的数据错误，只会将 finish_de 置 1。

### 4.5. vn模块

vn 模块任务是将一个数据的纠错过程。外部端口如图 4-5 所示，端口信号的定义如表 4.5 所示。

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

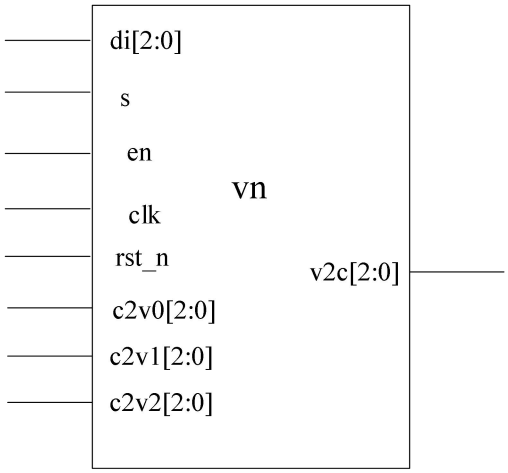



**Figure 4.5. vn模块.**

**Table 4.5. vn 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *di* | 数据输入 | 3 | 输入 |
| *s* | 状态信号 | 1 | 输入 |
| *en* | 状态信号 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *c2v0* | 结果输出 | 3 | 输入 |
| *c2v1* | 编码结束信号 | 3 | 输入 |
| *c2v2* | 解码结束信号 | 3 | 输入 |
| *v2c* | 解码成功信号 | 3 | 输出 |

模块 vn 的功能是，对数据进行纠错。输入数据前，先将 *s* 和 *en* 置 0，然后再输入数据，通过 v2c2v 模块进行判断，如果正确则直接输出；如果错误，就先将 *en* 和 *s* 置 1，然后将 v2c2v 模块产生的三个 *c2v* 信息，与输入的数据进行一系列运算，再将数据输出。

### 4.5.1. vn模块内部图

vn 模块内部结构图如图 4-6 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

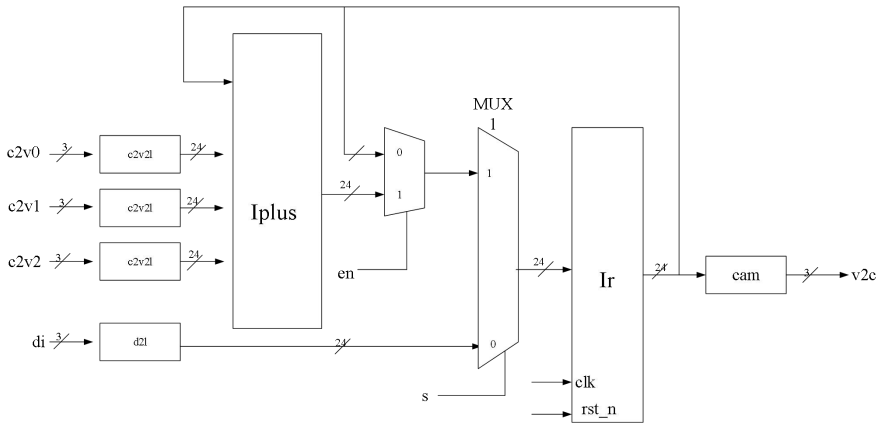



**Figure 4.6. vn模块内部结构图.**

图 4-6 中，*lr* 模块是一个 24 位的寄存器，为将整个模块描述为 RTL 级，应将其余模块理解为函数。硬判决算法中，变量节点也要维护一个“不置信度”表，称为 LLR 表。这个 LLR 表就是图 4-6 中的 *lr* 寄存器。LLR 表的初始值是由输入数据决定的，输入 *GF*(8)数据与 LLR 表的关系如图 4-7 所示。LLR 表类型的信号可以定义为 reg[0:23]，但里边的数据应理解为{*l*0[2:0]，*l*1[2:0]，*l*2[2:0]，*l*3[2:0]，*l*4[2:0]，*l*5[2:0]，*l*6[2:0]，*l*7[2:0]}。

| 输入 | [0:2] L0 | [3:5] L1 | [6:8] L2 | [9:11] L3 | [12:14] L4 | [15:17] L5 | [18:20] L6 | [21:23] L7 |
|---|---|---|---|---|---|---|---|---|
| 000 → | 000 | 001 | 001 | 010 | 001 | 010 | 010 | 011 |
| 001 → | 001 | 000 | 010 | 001 | 010 | 001 | 011 | 010 |
| 010 → | 001 | 010 | 000 | 001 | 010 | 011 | 001 | 010 |
| 011 → | 010 | 001 | 001 | 000 | 011 | 010 | 010 | 001 |
| 100 → | 001 | 010 | 010 | 011 | 000 | 001 | 001 | 010 |
| 101 → | 010 | 001 | 011 | 010 | 001 | 000 | 010 | 001 |
| 110 → | 010 | 011 | 001 | 010 | 001 | 010 | 000 | 001 |
| 111 → | 011 | 010 | 010 | 001 | 010 | 001 | 001 | 000 |

**Figure 4.7. 输入GF(8)数据到LLR的转换.**

图 4-6 中的 *d2l* 可以理解一个函数，作用是实现图 4-7 所示的 *GF*(8)符号到 LLR 表的转换。如果将 *l*0—*l*7 都定义为 3 位向量，按 Verilog 语法，有 *lr*={*l*0,*l*1,*l*2,*l*3,*l*4,*l*5,*l*6,*l*7}；但 *l*0—*l*7 都是“压缩数据”，含义是 2 的幂，运算时，要将 0 理解为 $2^0$=1，1 理解为 $2^1$=2，2 理解为 $2^2$=4，3 理解为 $2^3$=8。

图 4-6 中的的 *c2v2l* 模块也要理解为一种函数，作用是将 *c2v* 消息转换为 LLR 表格式，转换方法如图 4-8 所示。

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

| c2v | L0 | L1 | L2 | L3 | L4 | L5 | L6 | L7 |
|---|---|---|---|---|---|---|---|---|
| 000 | 000 | 001 | 001 | 001 | 001 | 001 | 001 | 001 |
| 001 | 001 | 000 | 001 | 001 | 001 | 001 | 001 | 001 |
| 010 | 001 | 001 | 000 | 001 | 001 | 001 | 001 | 001 |
| 011 | 001 | 001 | 001 | 000 | 001 | 001 | 001 | 001 |
| 100 | 001 | 001 | 001 | 001 | 000 | 001 | 001 | 001 |
| 101 | 001 | 001 | 001 | 001 | 001 | 000 | 001 | 001 |
| 110 | 001 | 001 | 001 | 001 | 001 | 001 | 000 | 001 |
| 111 | 001 | 001 | 001 | 001 | 001 | 001 | 001 | 000 |

**Figure 4.8. c2v消息到LLR表格式的转换.**

图 4-6 中 *lplus* 模块是比较复杂的函数，本身包含若干个底层函数，内部运算关系见图 4-9。

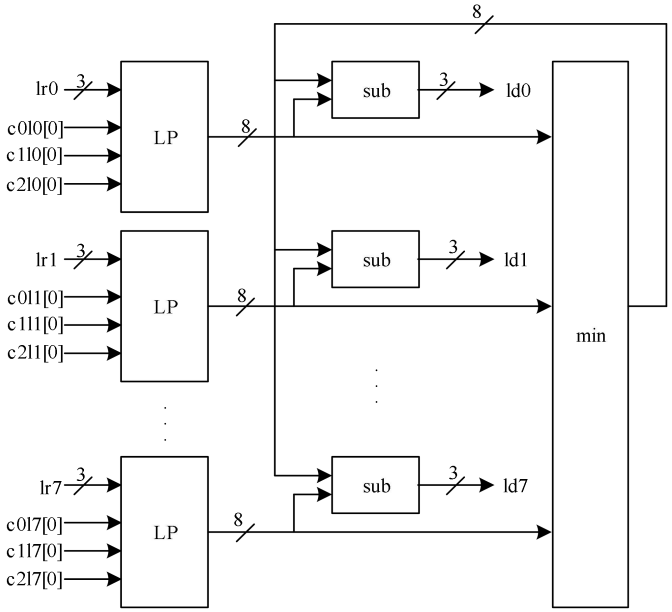



**Figure 4.9. clplus模块内部运算关系.**

图 4-9 中的 *lp* 模块的作用比较复杂，它实现的功能先解压 *lr* 寄存器中的“压缩数据”*lx*，这里 *x*=0,1,…7。将数据恢复为 $2^n$ 的形式，再与 *c2v* 消息得到的 LLR 格式数据消息相加。例如 *lr* 寄存器中原来的数据是从输入数据 0 转换而来的则

$$lr = \{0,1,1,2,1,2,2,3\} \qquad \text{* MERGEFORMAT (4-9)}$$

如果来自校验节点的 *c2v* 消息分别为 *c2v*0=1，*c2v*1=1，*c2v*2=2。则转换后的 LLR 数据分别为：

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

$$c2v0l=\{1,0,1,1,1,1,1,1\} \quad \text{* MERGEFORMAT (4-10)}$$

$$c2v1l=\{1,0,1,1,1,1,1,1\} \quad \text{* MERGEFORMAT (4-11)}$$

$$c2v2l=\{1,1,0,1,1,1,1,1\} \quad \text{* MERGEFORMAT (4-12)}$$

按定义，*lr* 应先做指数展开，用 *lrp* 表示，有

$$lrp=\{1,2,2,4,2,4,4,8\} \quad \text{* MERGEFORMAT (4-13)}$$

然后直接将（4-2）（4-3）（4-4）和（4-15）的对应位相加，得

$$lrpc=\{4,3,4,7,5,7,7,11\} \quad \text{* MERGEFORMAT (4-14)}$$

再然后通过 *min* 模块处理，找到 8 个数中得最小值“3”，然后 *lp* 模块获得减去得最小值，即

$$dp=\{1,0,1,4,2,4,4,8\} \quad \text{* MERGEFORMAT (4-15)}$$

将（4-7）中得数据“近似为 $2^n$，然后保留 *n*”。在做“压缩”处理时（4-7）中的“1”只能近似 $2^1$，仍记为 1，因为不能有两个或两个以上 0，“2”也记为“1”，“4”记为 2，“8”记为 3。即最后的结果为 *dl*，

$$dl=\{1,0,1,2,1,2,2,3\} \quad \text{* MERGEFORMAT (4-16)}$$

这样，每个元素仍可用 3 位二进制数表示。最后图 4-9 中的 *ld*0—*ld*7 将被合并为一个 LLR 表格式的 24 位数据，经图 5-10 中的 *cam* 函数运算后得到 *v2c* 消息，这个 *v2c* 消息就是该变量节点最可能的 *GF*(8)数据。

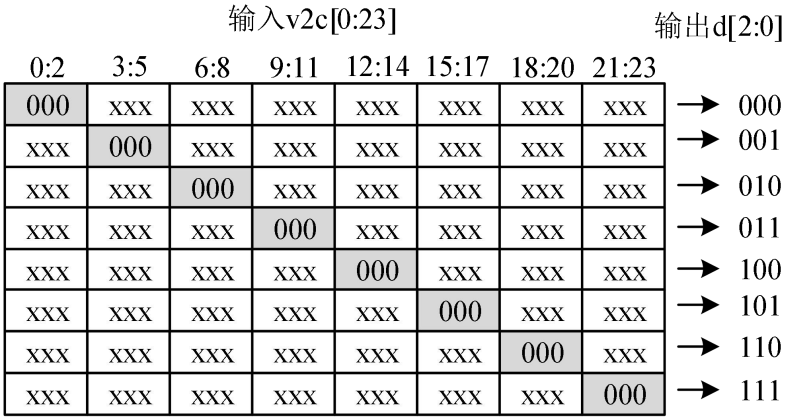



**Figure 4.10. cam功能.**

### 4.6. v2c2v模块

v2c2v 模块的任务有两个，一个是判断来自变量节点的数据是否正确，二是将数据发给变量节点（剔除了来自该节点本身的数据后的所有其它变量节点的数据之异或）。外部端口如图 4-11 所示，端口信号的定义如表 4.6 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

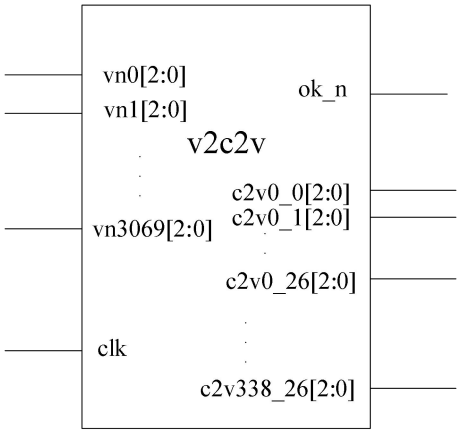



**Figure 4.11. v2c2v模块.**

**Table 4.6. v2c2v 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *vn0* | 数据输入 | 3 | 输入 |
| … | … | … | … |
| *vn3069* | 数据输入 | 3 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *ok_n* | 判断信号 | 1 | 输入 |
| *c2v0_0* | 信息输出 | 3 | 输出 |
| *c2v0_1* | 信息输出 | 3 | 输出 |
| … | … | … | … |
| *c2v0_26* | 信息输出 | 3 | 输出 |
| *c2v1_0* | 信息输出 | 3 | 输出 |
| … | … | … | … |
| … | … | … | … |
| *c2v338_26* | 信息输出 | 1 | 输出 |

模块 v2c2v 的功能是，先将输入的信息进行判断，如果 *ok_n* 为 0，表示数据没有错，如果 *ok_n* 为 1，则说明数据有错，然后将校验节点的信息处理一下子再发给变量节点。这个处理是剔除了来自该节点本身的数据后的所有其它变量节点的数据之和（异或），因为校验矩阵的列重为 3，所以对于输入的每一个信息，都会有 3 个对应的数据输出。

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

### 4.6.1. 校验节点与变量节点的连接关系

对于采用 *GF*(*N*)的 LDPC 来说，*v*2*c* 消息在传到校验节点时需要乘以 *H* 矩阵中相应的元素，这种乘法要符合 *GF*(*N*)运算规则。校验节点向变量节点发回的 *c*2*v* 消息需要除以 *H* 矩阵中的元素，或者说乘以该元素的倒数，运算规则也是 *GF*(*N*)规则。

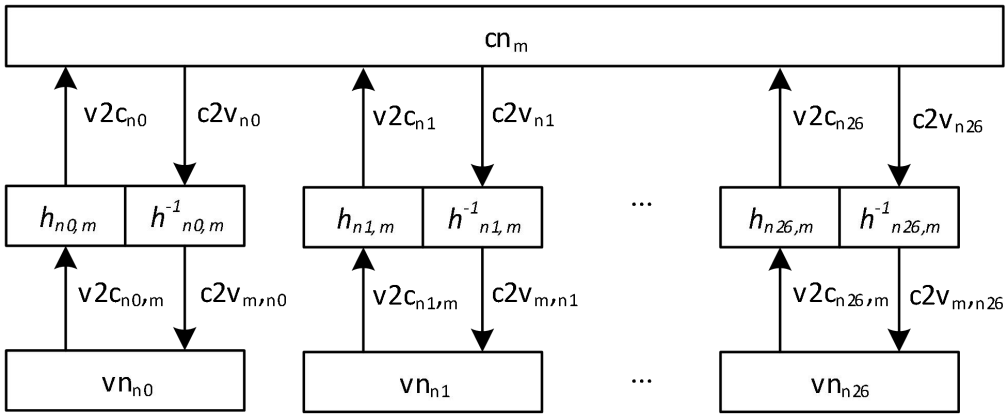
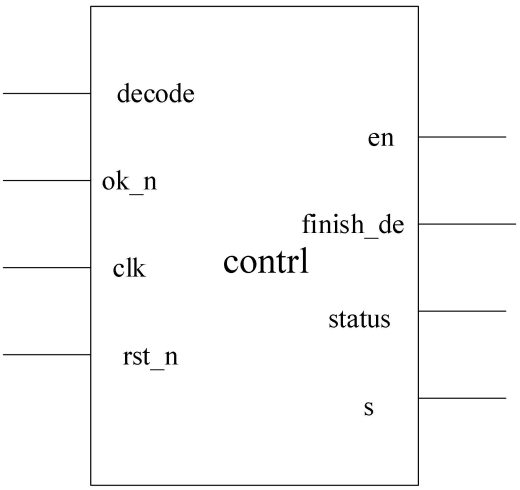



**Figure 4.12. 校验节点与变量节点的连接关系.**

图 4-12 是一个校验节点 $cn_m$ 与若干变量节点的连接关系示意图，变量节点的个数是由“行重”决定的，图 4-12 表示的行重为 27，本文的行重大部分为 27，极少数为 28。其中的乘法其实是一个变量与常数相乘，对 *GF*(8)的问题就是 7 种，可以理解为 7 个表或函数，可以命名为 *mh*1，…，*mh*7 也要写各一个函数，虽然数不变，但以后可能需要加驱动。

### 4.7. contrl模块

contrl 模块任务是将解码的几个模块进行整合统一，便于运算。外部端口如图 4-13 所示，端口信号的定义如表 4.7 所示。




OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

Figure 4.13. contrl模块.

Table 4.7. contrl 模块端口信号定义

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *ok_n* | 信号输入 | 1 | 输入 |
| *decode* | 解码使能 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *en* | 信号输出 | 1 | 输出 |
| *finish_de* | 解码结束信号 | 1 | 输出 |
| *s* | 信号输出 | 1 | 输出 |
| *status* | 解码成功信号 | 1 | 输出 |

解码器控制器内部有计算器和状态机。它只有两种状态，即工作状态和等待状态。在接收到 *decode* 命令后进入“工作状态”，进入工作状态后，使 *en* 信号为 1，允许 *lr* 寄存器更新（*en* 优先级低于 *s*），解码器开始迭代。在工作状态下发现 *ok_n* 信号为 0 即将 *finish* 置 1，*status* 置 1，以后转到“等待状态”，如果内部计数器达到最大值，将 *finish* 置 1，*status* 置 0，以后转到“等待状态”。

## 4.8. LDPC_joggle模块

LDPC_joggle 模块任务是将 Flash 输入的数据存起来然后读出数据进行编码解码，后将生成的数据一点一点的输出给 Flash。包含了 LDPC 模块和 RAM 模块外部端口如图 4-14 所示，端口信号的定义如表 4.8 所示。

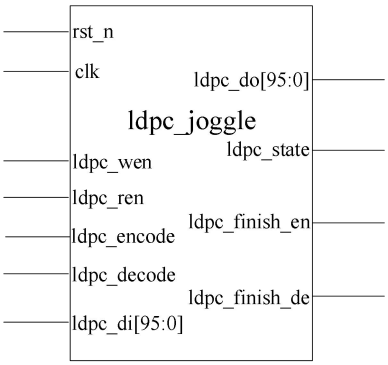



Figure 4.14. LDPC_joggle模块.

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

Table 4.8. LDPC 模块端口信号定义

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *LDPC_di* | 数据输入 | 96 | 输入 |
| *LDPC_encode* | 编码使能 | 1 | 输入 |
| *LDPC_decode* | 解码使能 | 1 | 输入 |
| *clk* | 系统时钟 | 1 | 输入 |
| *rst_n* | 复位信号 | 1 | 输入 |
| *LDPC_wen* | 数据写使能 | 1 | 输入 |
| *LDPC_ren* | 数据读使能 | 1 | 输入 |
| *LDPC_do* | 结果输出 | 96 | 输出 |
| *LDPC_finish_en* | 编码结束信号 | 1 | 输出 |
| *LDPC_finish_de* | 解码结束信号 | 1 | 输出 |
| *LDPC_state* | 解码成功信号 | 1 | 输出 |

模块 LDPC_joggle 的功能是将 Flash 和 LDPC 连接起来。先将 *LDPC_encode* 信号置 1 ，表示是编码时间，此时，*LDPC_wen* 会发不断地产生脉冲，一次脉冲表示数据输入 96 位，并将数据写入 in_RAM 模块中。因为输入地数据为 8193 位，所以要产生 86 次脉冲。待数据输入完整后，*LDPC_ren* 会置 1，编码的数据输入会先读出，然后进行编码，编码完成 *LDPC_finish_en* 会为 1，表示编码结束，这时 *LDPC_encode* 会回到零；待 Flash 处理好后，会有 *LDPC_decode* 置 1，进入解码状态，解码完成会有 *LDPC_finish_de* 信号为 1，然后会有 *LDPC_wen* 信号，让生成的数据存入 out_RAM 模块中，再给 *LDPC_ren* 脉冲，将数据输出到 Flash 中。编码数据读出的时候需要把最后一组数据去掉 63 位，解码数据写入时需要添加 6 位 0。

## 4.9. RAM模块

RAM 模块任务是将数据存入或者输出。外部端口如图 4-15 所示，端口信号的定义如表 4.9 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

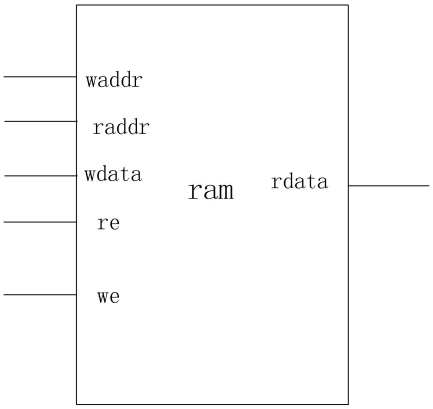



**Figure 4.15. RAM模块.**

**Table 4.9. RAM 模块端口信号定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *waddr* | 写地址 | 自定义 | 输入 |
| *raddr* | 读地址 | 自定义 | 输入 |
| *wdata* | 写数据 | 自定义 | 输入 |
| *rdata* | 读数据 | 自定义 | 输出 |
| *we* | 写使能 | 1 | 输入 |
| *re* | 读使能 | 1 | 输入 |

模块 RAM 的功能是，对数据存入取出。写数据时，先将输入的数据传给 wdata，输入的地址赋给 waddr，然后将 we 置 1 就写入了；读数据时，先将读出的地址赋给 raddr，然后将 re 置 1，后将 rdata 数据传出即可。

## 4.10. Flash内部

**Table 4.9. Flash 内部寄存器定义**

| 信号名称 | 功能定义 | 位宽 | 方向 |
|---|---|---|---|
| *reg0* | 寄存器 | 32 | 只写 |
| *reg1* | 寄存器 | 32 | 只写 |
| *reg2* | 寄存器 | 32 | 只写 |
| *reg3* | 寄存器 | 32 | 只写 |



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

| | | | |
|---|---|---|---|
| *reg4* | 寄存器 | 32 | 只读 |
| *reg5* | 寄存器 | 32 | 只读 |

297

298 寄存器的功能：

299 reg0[0] 模块使能

300 reg0[2 : 1] 功能设置

301 000 复位值

302 001 读内存页

303 010 写内存页

304 011 块擦除

305 100 读取 FLASH 的 ID

306 101 获取 FLASH 设置

307 110 设置 FLASH

308 111 复位模块

309 reg1[23 : 0] 块地址（当读写设置时为设置地址)

310 reg2[15 : 0] 页地址（当写设置时为设置内容）

311 reg3[31 : 0] 目标 RAM 地址

312 reg4[0] 当前功能状态 0：正在进行 1：完成

313 reg4[1] 读取内存页的解码状态 0：失败 1：成功

314 reg5[31:0] 当读设置时为对应设置地址的内容，当读 ID 时为读取到的 ID

315 ## 5. 仿真与结论

316 ### 5.1. 无接口仿真

317 通过 modelsim 仿真，可以看到对输入的 8193 位信息位数据通过编码运算后，显示 p1 与 p2 的结

318 果，而且 *finish_en* 信号为 1，如图 5-1 所示。当编码完成后，会显示“decode is successful!”并将正确

319 的结果输出，也可以看到 *finish_de* 和 *status* 信号都为 1，如图 5-1 和 5-2 所示。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

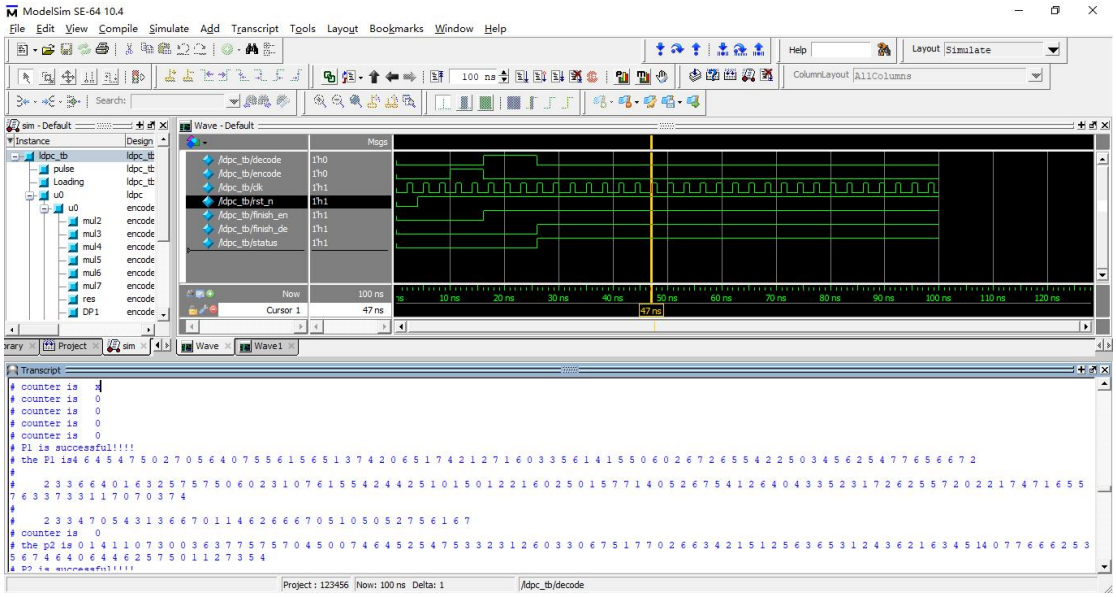



Figure 5.1.

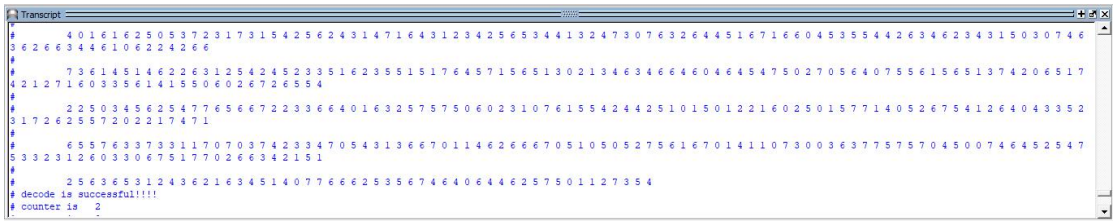



Figure 5.2.

## 5.2. 有接口仿真

对含有接口的仿真，从图 5-3 和图 5-4 可以看出，数据写入，并进行编码还是没有问题的，可以显示出“Encode is successful”，*LDPC_finish_en* 也显示为 1：



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

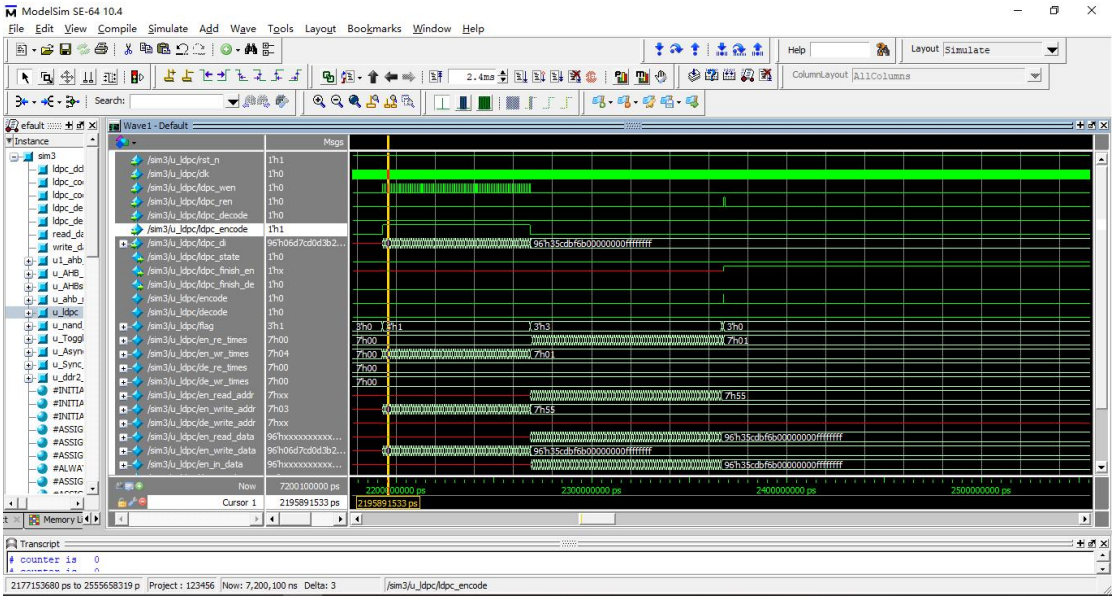



**Figure 5.3.**

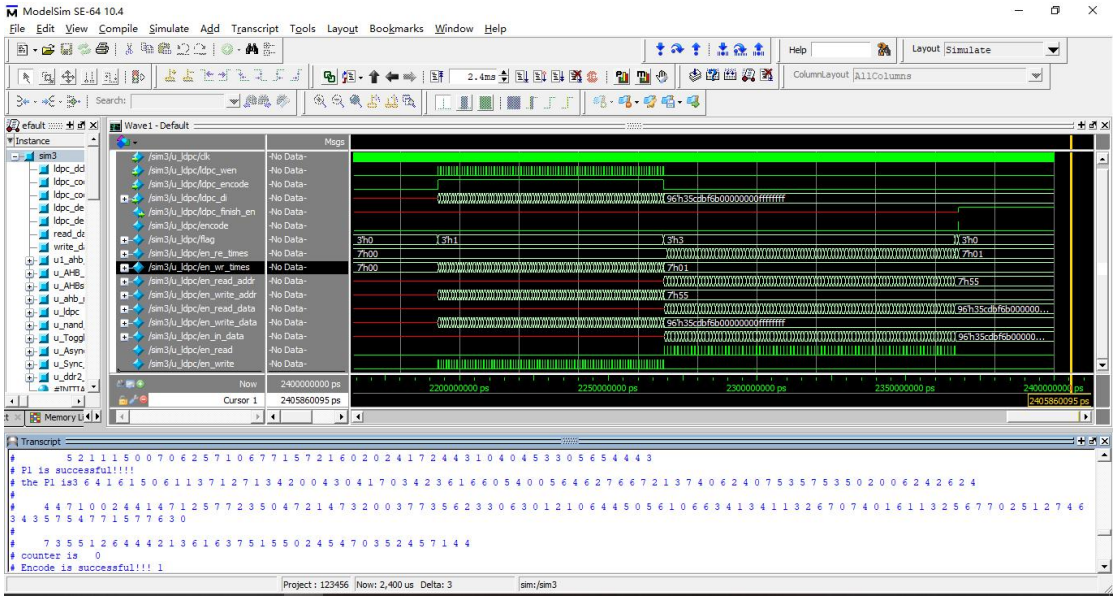



**Figure 5.4.**



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

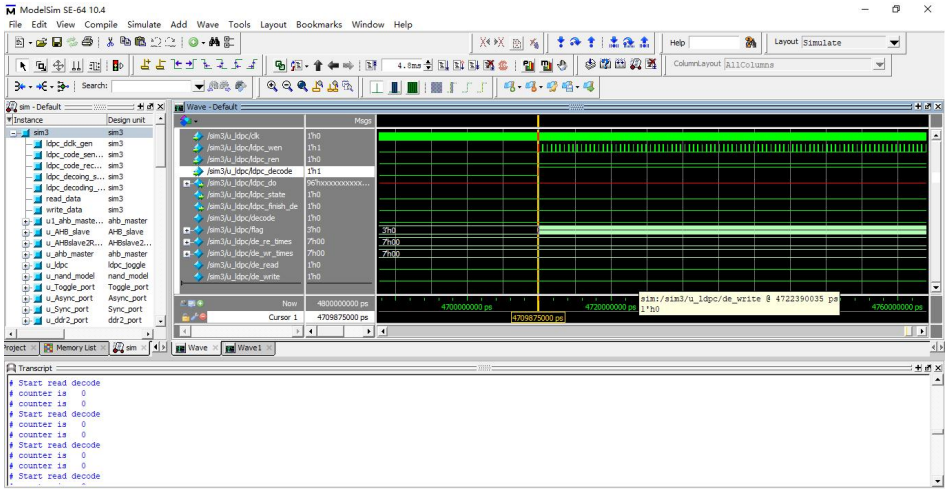
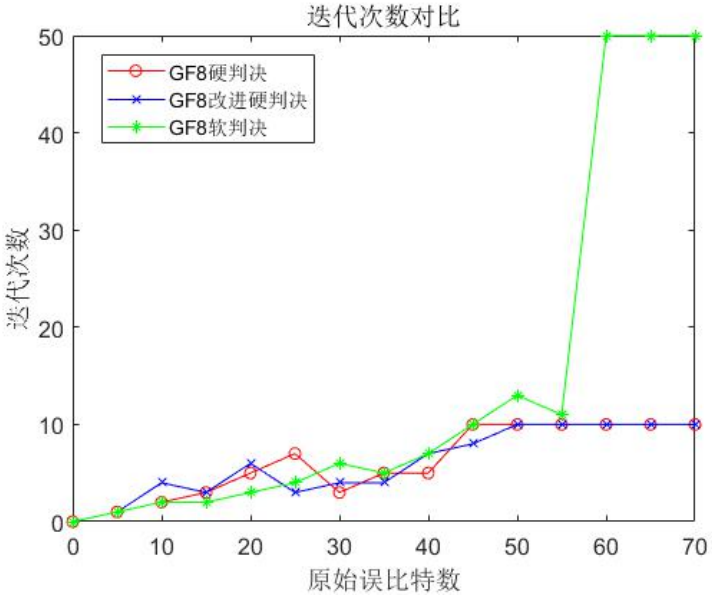



**Figure 5.5.**

## 5.3. 性能测试

通过 Matlab 模拟仿真硬判决、改进型硬盘决和软判决三种解码电路。在仿真中，本文构造了码长为 9210，码率为 0.89 的 LDPC 码，并通过程序随机对码字产生 0~70 个原始误比特数，分别用三种解码电路对其进行纠错，比较其在同一原始误比特率下完全纠错所需的平均迭代次数以及最大纠错能力。

图 5-6 反映出三种解码电路在纠错时的收敛速度，对应包含 0~70 个以内的原始误比特数的码字，分别用三种解码电路对其进行纠错，比较三者在同一原始误码率下完全纠错所需的迭代次数。可以看出，改进型硬判决的平均最大迭代次数小于硬判决而大于软判决，即改进型硬判决纠错速度快于硬判决，慢于软判决。






OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

**Figure 5.6. 错误率信息精度与迭代效率.**

图 5-7 比较了三种解码算法最大所能纠错的原始误比特数。其中硬判决为 40 位，改进型硬判决为 45 位，软判决为 55 位，三种解码算法在各自最大纠错能力内都能将误比特数降为 0。

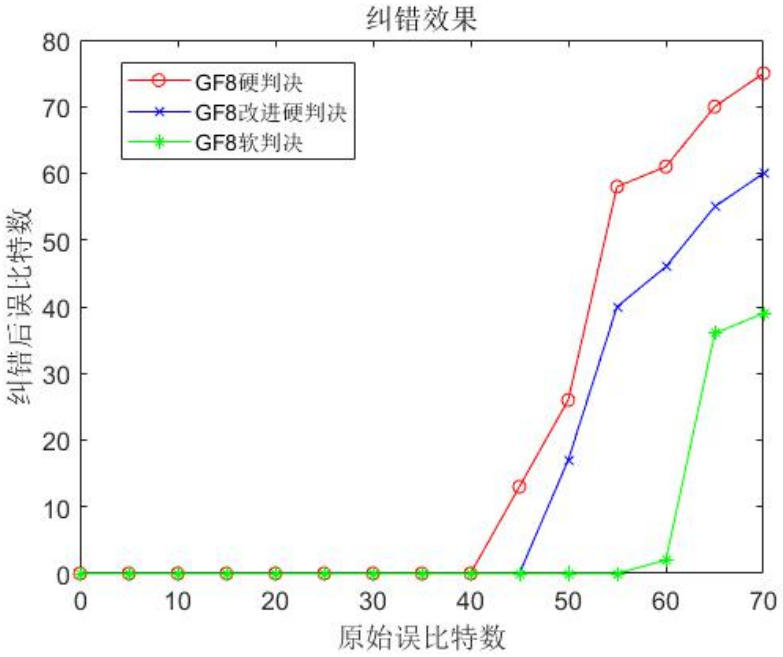



**Figure 5.7. 最大迭代次数内的纠错比特数.**

为了直观评价本文硬判决和改进型硬判决的纠错性能，用 Matlab 对其进行了计算机编程仿真。对编码后的码字进行 BPSK 调制，并通过高斯加性白噪声信道，设置最大迭代次数为 10 次，在两种解码电路下进行解码和纠错，并将结果与非编码系统进行比较。



OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15

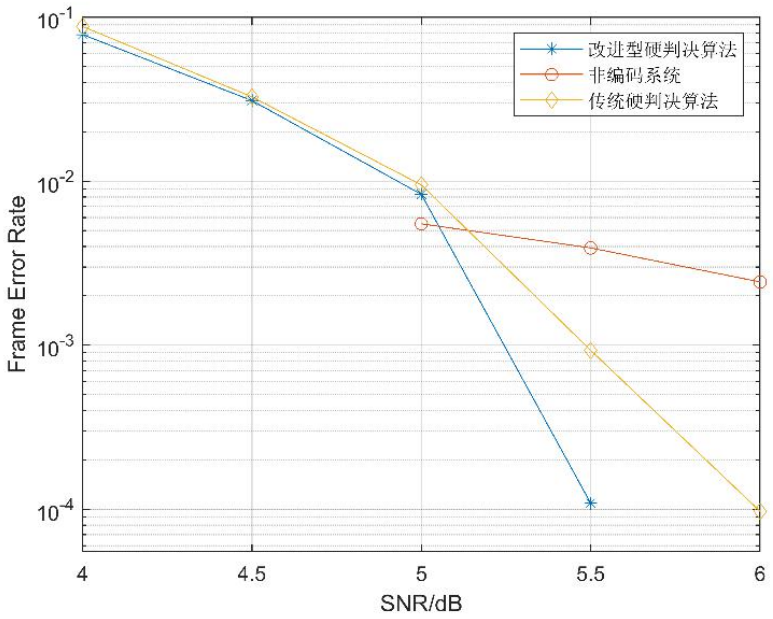



**Figure 5.8. SNR-FER图.**

图 5-8 反映了在不同信噪比的高斯噪声下信号通过两种译码算法进行译码纠错后的误码率，其中横坐标 SNR 定义为信号与噪声之比，单位为 dB，SNR 越大，噪声对信号的干扰越小，其导致原始码字的错误率越小，纵坐标 FER(FRAMe Error Rate)定义为译码纠错结束后，错误的符号数与该码字总符号数之比。

从图中看出：

传统硬判决算法在 SNR 大于 6、改进型硬判决算法在 SNR 大于 5.5，都能使 FER 降低到 10-4，且改进型硬判决在 SNR 大于 6 时，FER 降为 0，两种算法都具有明显的纠错效果。

在误码率为 10-4 时，改进型硬判决算法性能明显优于传统硬判决算法，其编码增益较后者大 0.5dB。

相较于非编码系统，本作品中的两种译码算法在 SNR 大于 5.5 时有明显的纠错效果和较大的编码增益。

## References

[1] 刘卫. NAND Flash 控制器的设计与验证[D]. 国防科学技术大学.

[2] 彭立, 朱光喜. 基于校验和的 LDPC 码硬判决解码算法的研究[J]. 移动通信, 2004, 028(0z1):12-15.

[3] 彭立, 朱光喜. 基于可靠性的 LDPC 码软判决解码算法的研究[J]. 信息技术, 2004, 000(006):48-50.

OALib PrePrints | http://dx.doi.org/10.4236/oalib.preprints.1200293 | CC-BY 4.0 Open Access
Received: 2020/12/19, Published: 2021/1/15