1 SerDes 接收端数据链路层设计

1.1 8B/10B 解码器设计

根据 JESD204B 协议规定，SerDes 接口为保证信内数据的直流平衡，并且为了便于时钟恢复，采用 8B/10B 码作为数据链路层的编码方式。JESD204B 中采用的 8B/10B 编解码部分主要参考 IEEE802.3 以太网协议中关于 8B/10B 的编解码部分。但是由于应用环境的不同，JESD204B 协议的 8B/10B 编解码方式同以太网相比略有不同。

1.1.1 协议分析

以太网协议提供了完整的编码表，也可以看作为解码表，这几张表格的正确性毋庸置疑，是最值得参考的资料。表中将总共 268 种的编码情况全部列出，可以作为校验编码正确性的基准值。

以太网协议建议，将一个码组的 RD[1]分为三部分，第一是上一码组计算后得出的 RD，第二是编码后 6B 部分的 RD，第三是编码后 4B 部分的 RD。而最后得到的 4B 部分的 RD 将作为这一组码的 RD 用于接下来码组的计算。RD 运算的基本结构：$last\_code\_group\_RD \rightarrow 6B\_sub-block \rightarrow 6B\_RD \rightarrow 4B\_sub-block \rightarrow 4B\_RD(new\_last\_code\_group\_RD)$。

每个 sub-block 的判断可用以下伪代码表示（6B 和 4B 略有不同）：

if 000111 or 0011 or $1s > 0s$
    $6B\_RD = +; RD\_4B = +;$
else if 111000 or 1100 or $1s < 0s$
    $6B\_RD = -; RD\_4B = -;$
else
    $6B\_RD = last\_code\_group\_RD;$
    $4B\_RD = 6B\_RD;$
endif

最后是以太网协议中关于 RD 错误的处理，在协议附录中给出了一些接收当中的 RD 错误。可以发现，RD 错误是不能精确定位的，它的检测主要是通过接收机本地的 RD 和所接收到的 RD 不符所产生的错误。但由于一系列的中性码并不会改变 RD，前一码接收产生的错误可能因为一系列的中性码而直到几个码字后才能检测到。

在 JESD204B 协议中关于 8B/10B 编解码的规定，阅读接口协议的数据链路层内容，可以发现，在编解码器之前还有一级控制，主要是用来针对数据成帧结构中的 lane、frame、multiframe 的校准、同步和错误控制，而控制的依据就是解码器中获得的控制字。

在 JESD204B 协议中只用到了 8B/10B 所用控制字的 5 个，这将简化控制字的解码复杂度，控制字分别如下：

K.29.0 即 D，表示 Multiframe 的开始。

K.28.3 即 A，表示 Lane 校准，一般在多帧最后出现。

K.28.4 即 Q，表示 Link 设置数据的开始，在他之后跟一系列设置数据，配置 Link，他也是 ILAS[2]的组成部分。

K.28.7 即 K，表示 Group 同步，可以说是链接开头最重要的部分，用来保持同步，是 CGS[3]的重要控制字。

K.28.7 即 F，表示 Frame 校准，一般表示一帧结束。

JESD204B 协议还规定了三种重要的解码错误，这些在解码器级别的错误是属于不太严重的错误，有可能经常发生，在错误并不严重的情况下并不需要进行重同步，但是需要上报给错误处理部分，供应用层决定如何处理。

错误包括以下三种：

[1]Running Disparity
[2]Initial Lane Alignment Sequence
[3]Code Group Synchronization

Not-in-table Error 这种错误意味着接收到的码字在任何 RD 情况下都不存在于码表中，就是一些非法的码字。对于这些码字，协议规定接收端需要重复之前收到的最新的没有错误的帧。

Disparity Error 这种错误就是上文提到的 RD 错误，协议规定解码器要根据收到的数据和 RD 直接解码。由于在检测到 RD 错误时，可能产生错误的不是这个码字，这样的处理方式也比较合理。

Unexpected Control Character 这种错误就是指未出现在指定位置的控制字，这一错误的具体处理需要由 lane/frame 监测部分来决定，属于接下来层级的处理，解码器并无法判断出这一错误。

1.1.2 解码思路

通过对现有论文编解码方式的理解分析，可以发现编码的方式比较多样，但解码仍主要停留在通过逻辑的方式。在 [6]、[5] 和 [3] 中采用的是纯逻辑解码方式；在 [4] 和 [2] 这两篇文章中采用的是多路选择器的方式。

在现有文章的解码中，RD 值仅作为差错检测，这是一种对信号资源的浪费，如果通过 RD 来进行解码，可以更好的利用现有的码表。对于已知码组的 RD 信息，可以得出下一组码可能的编码情况，这时就可以通过取反操作来压缩解码表的大小，更快的处理解码操作。例如，已知前一组码字的 RD 信息为 $RD+$，则可以推测出接下来的 $6B$ 数据的 RD 信息只有可能为 $RD-$ 或者均衡两种可能。因此只需要处理一种极性的码字就可以完成对整个编码的解码，复杂度变为原来的一半。

对于数据字而言，在获知当前 RD 状态的前提下，解码就分为两种情况。一种是相反的极性，还有一种是均衡的极性。对于相反的极性而言，不需要对另一种极性解码，整张解码用表就可以缩小一半，一方面节约了芯片面积额，一方面提高了解码效率。对于均衡极性而言，由 [3] 中编码原理分析可知，对于均衡码的解码其实非常简便，只需要输出其低 5 或 3 位。因此，只需要设计一个均衡码判断电路，就可以快速选择是否通过解码逻辑极性解码。

对于控制字而言，由 [2] 中提到的控制信息检测可以发现，通过 RD 和固定位置的比特就可以区分该码字是否为控制字，并且确定是哪一类控制字 ($K.28.x$ 还是 $K.23.7$、$K.27.7$、$K.29.7$、$K.30.7$)。在分析编码可知，控制字的 $3B$ 或 $5B$ 部分的编码规则同数据字是相同的。那么就可以“借用”数据字的解码部分来对控制字部分解码，准确输出控制码字。其中包括了均衡和非均衡的情况，处理逻辑同数据字，唯一不同的就是控制字状态标记是否拉高。

最后对于解码模块的考虑，一般情况下都是采用的是 verilog 语言的 case 逻辑，但是对于高效的电路来说要尽量避免对语言原生性能的依赖。通过逻辑化简的方法对更小的码表进行化简，这样的得到表达式速度更快，并且面积较小。

具体逻辑框图如图 1所示。

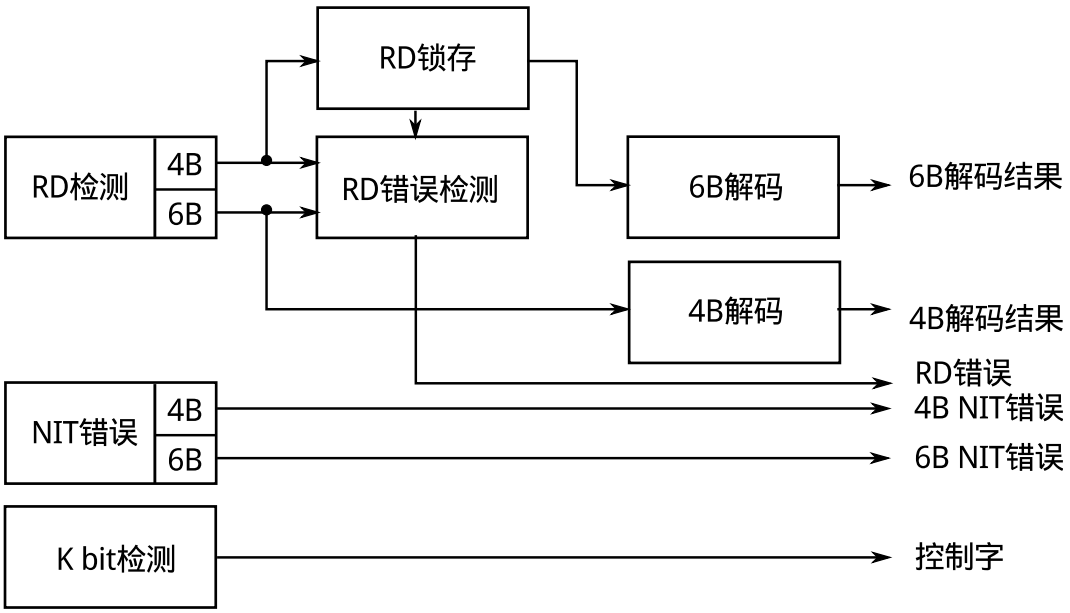

图 1: *SerDes* 接收端数据流传输结构框图

1.1.3 第一级，预处理级

K Bit 检测 根据输入完整的 $10B$ 数据判断是否为控制信号。有以下伪代码：

if $iedc == 1111|0000$
　　$K.28.x = 1; K.x.7 = 0;$
else if $jhgfie == 010111|101000$
　　$K.28.x = 0; K.x.7 = 1;$
else
　　$K.28.x = 0; K.x.7 = 0;$
endif

本设计中的 8B/10B 编码，只采用了 6B 部分为 K.28 的控制字。由完整检测逻辑可以得出，只需要检测 6B 部分的 cdei 位是否为 0000 或者 1111，就能判断出结果。所以判断控制字本设计可以采用最简单的逻辑表达式：

$$K\_bit = (c\&d\&e\&i) \quad | \quad (c|d|e|i)$$

$6B$ 和 $4B$ 平衡检测 就是对给出的并行数据，分别输出 $6B$ 和 $4B$ 是否平衡，平衡即指 0 和 1 的数量是否相同。有以下伪代码[4]：

if $RL(6B) == 3$
　　$Balance\_6B = 1;$
else
　　$Balance\_6B = 0;$
endif

if $RL(4B) == 2$
　　$Balance\_4B = 1;$
else
　　$Balance\_4B = 0;$
endif

RD 信息检测 RD 信息检测主要指分别计算 6B 和 4B 部分的 RD，每一部分又分别有两个输出，$RD-$ 和 $RD+$。因为每一组码字的 RD 一共存在三种可能，即正、负和平衡，不属于负和正的码字，既为平衡，平衡的判断也可由上文中的伪代码确定。这些重要的信息主要用于极性错误检测和解码。通过对合法码字的真值表进行化简，可以得到快速 RD 极性检测模块，准确输出正确码字 RD 的负和正信号。

最终模块输出 4 个信号，即 RD_6B_pos、RD_6B_neg、RD_4B_pos、RD_4B_neg。

NIT 错误检测 Not-In-Table Error 的检测也分为 4B 和 6B 两部分。

6B 部分的错误码字一共为 14 种，如表 1，参考 [3] 一文的方法可以分为两类，既考虑 abcd 全为 0 或全为 1 的情况和 abcd 有且仅有 1 个 1 或仅有 1 个 0 的情况。前者直接可以判断该码字为错误码字，后者再观察 ef 是否全为 0 或全为 1，也可判断码字是否错误。

4B 部分的错误码字相对较为复杂，考虑到 K.28 中只有 5 个控制字是合法的，所以要对余下的码字报错。4B 部分报错情况如表 2所示。

[4]其中 RL 表示游程长度计算，即 1 的个数。下同。

表 1: *6B* 码字错误情况

abcdei	abcdei
000000	111111
000001	111110
000010	111101
000100	111011
001000	110111
010000	101111
100000	011111

表 2: *4B* 码字错误情况

abcdei	hgfj
xxxxxx	0000
	1111
001111	0101
	1001
	0110
110000	1010
	1001
	0110

1.1.4 RD 错误检测

RD 错误检测部分主要负责检测关于 RD 的错误，主要就是指不能出现连续相同的 RD 变化。这就与寄存器中上一串码字的 RD 有关，所以 RD 错误检测还负责解码器 RD 的刷新。检测可以通过 RD 信息计算模块提供的信号进行判断，并得到新的 RD 存入寄存器，如表 3所示。

表 3: *RD* 错误检测及新 *RD* 生成表

last	6B+	6B-	4B+	4B-	err	new
0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	0	1	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	1
1	0	0	0	0	0	1
1	0	0	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	0	0	0
1	0	1	0	1	1	0
1	0	1	1	0	0	1
1	1	0	0	0	1	1
1	1	0	0	1	1	0
1	1	0	1	0	1	1

根据此表化简逻辑表达式即可得到准确的 RD 错误和本码字的 RD 情况。为了保证时序的正确和避免未知情况的发生，参考了 [1] 中的有限状态机方法来对 RD 进行判断、存储。

1.1.5 第二级，解码级

解码表输入级 根据 *RD* 情况，对输入码字进行反转。伪代码如下：

if $RD == RD+$
$6B = 6B$;

else
$6B$ =!$6B$;
endif

$RD\_6 = RD|Balance\_6B$
if $RD\_6 == RD+$
$4B = 4B$;
else
$4B$ =!$4B$;
endif

$6B/5B$ **和** $4B/3B$ **解码级** 根据解码表对输入码字解码。解码表中存的查找项均为 RD 为正时的解码，根据输入输出解码结果。

直接输出或解码输出选择级 根据码字平衡与否选择是直接输出低五位还是输出查表解码后的结果。

控制字输出或数据字输出选择级 根据 $K.x.7$ 和 $K.28.x$ 两位，选择输出控制字还是数据字。伪代码如下：

if $K.x.7 == 1$
$8B = 5B + 111; K = 1;$
else if $K.28.x == 1$
$8B = 00111 + 3B; K = 1;$
else
$8B = 5B + 3B; K = 0;$
endif

不同于 [2] 文中所描述的 CASE 方法，也不同于 [3] 中的纯逻辑方法。本设计引入了各个模块的 $RD+$ 信息，在读入数据之前先根据 RD_pos 信息对码字进行反转。这样，在解码时只要考虑 RD 为 $RD-$ 和平衡的情况。对于 6B 解码表由原来的 48 种情况减少为 34 种（如表 4所示）；对于 4B 解码表，由于其解码需要考虑 K Bit 的情况，由原来的 28 种情况减少为 16 种（如表 5所示）。并且由于快速的 RD 信息计算，使得解码模块能够更快的进行解码，缩短了解码所需时间。最后通过逻辑化简的方法对更小的码表进行化简，这样的得到表达式速度更快，并且面积较小。

表 4: *6B* 解码表

abcdei	EDCBA	abcdei	EDCBA
000011	11100	011001	11001
000101	01111	011010	11010
000110	00000	011100	11100
000111	00111	100001	11110
001001	10000	100010	11101
001010	11111	100011	00011
001011	01011	100100	11011
001100	11000	100101	00101
001101	01101	100110	00110
001110	01110	101000	10111
010001	00001	101001	01001
010010	00010	101010	01010
010011	10011	101100	01100
010100	00100	110001	10001
010101	10101	110010	10010
010110	10110	110100	10100
011000	01000	111000	00111

表 5: *4B* 解码表

K	hgfj	HGF
0	0001	111
0	0010	000
0	0011	011
0	0100	100
0	0101	101
0	0110	110
0	1000	111
0	1001	001
0	1010	010
0	1100	011
1	0001	111
1	0010	000
1	0011	011
1	0100	100
1	1010	101
1	1100	011

1.2 解扰器设计

1.3 Frame/Lane 对齐字符检测设计

参考文献

[1] Abdullah-Al-Kafi. Development of fsm based running disparity controlled 8b-10b encoder-decoder. *HCTL Open Int. J. of Technology Innovations and Research*, 2, 2013.

[2] Actel. Implementing an 8b/10b encoder/decoder for gigabit ethernet in the actel sx fpga family. Technical report, Actel Corporation, October 1998.

[3] A. X. Widmer. A dc-balanced, partitioned-block, 8b/10b transimission code. *IBM Journal of research and development*, 27‘(5):440–451, September 1983.

[4] 温龙. 8b/10b 解码器设计. 科学技术与工程, 18(7), 2007.

[5] 贺传峰. 一种新的 8b/10b 编解码硬件设计方法. 高技术通讯, 15(3), 2005.

[6] 赵王虎. 基于逻辑设计的光纤通信 8b/10b 编解码方法研究. 电路与系统学报, 8(2), 2003.