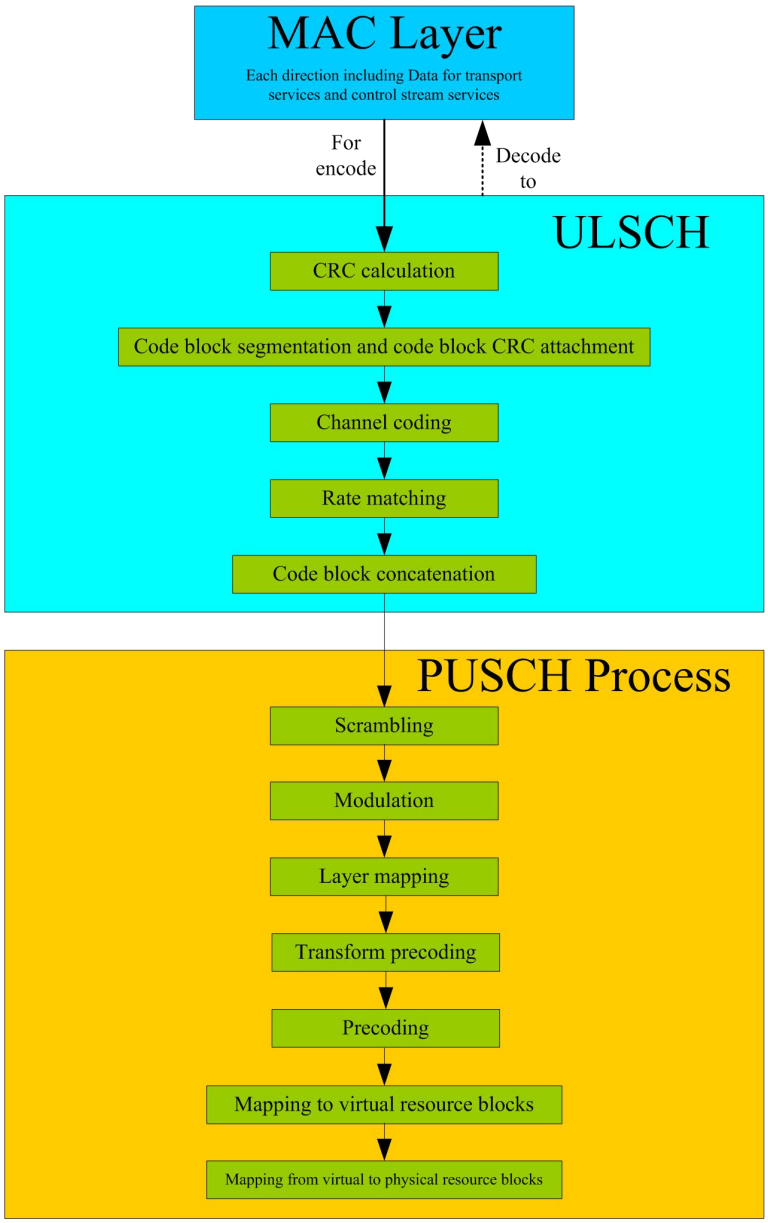
The data stream and control steam from MAC to UL-SCH , would do the following process.



[1] CRC calculation

CRC 的功能是为了实现错误检测，实质是按照循环生成多项式在原始数据的后面增加了相应的 Bits 位。循环生成多项式的类型包括：

CRC24A,CRC24B,CRC24C,CRC16,CRC11,CRC6

将原始数据的长度，从 A 拓展至 A+L，其中 L 是校验位的长度。

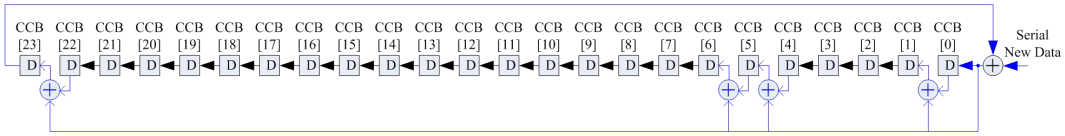
编码以系统的形式进行，意味着在最简单的有限域 GF(2)中，添加 CRC 校验位之后，最终形成的序列多项式除以相应的 CRC 生成多项式，余数为 0.

FPGA 在编程设计的时候，校验位的生成需要首先赋予初始值.
协议里面似乎没有提到这个初始值，有点疑问？//需要初始化为全 0

如果 CRC 的生成多项式是 CRC24B

- $g_{\text{CRC24B}}(D)=[D^{24}+D^{23}+D^{6}+D^{5}+D+1]$ for a CRC length $L=24$;

那么逻辑设计的结构应该是：



特别注意：系统上电/软件复位/启动编码的时候，都需要设置 CRC 校验位的初始值；并且为了保证系统的可靠性（Reliability），将 CRC 校验位寄存器初始化为标准协议里面所要求的值（这里全部是'0'）,Such as

reg CCB[23:0] = InitialValueBasedOnProtocol; //For Reliability

[2] Code block segmentation and code block CRC attachment

2.1 Polar for Control channel streams

特别注意: The value of A is no larger than 1706.

2.1.1 If（ lseg != 1 ），这种情景非常简单，相当于这个步骤没有做任何处理，直接 Bypass;

2.1.2 If（ lseg == 1 ），这种情景比较复杂，在获取 a'_i 的过程中，如果原始数据的 Bits 位是偶数的话，原始数据没有做任何处理，直接 Bypass；如果原始数据的 Bits 位是奇数的话，需要在最前面插入 1 个为 0 的 Bit 位。

经过处理之后， a'_i 肯定的数据量肯定是偶数，随后将 a'_i 平均划分为两个独立的部分，分别进行 CRC 计算并将校验 Bits 位跟随在后面；生成多项式的类型包括：CRC24A,CRC24B,CRC24C,CRC16,CRC11,CRC6

最终得到两个序列块 c_{rk}（r 分别取值为 0 和 1），

for $k=0$ to $A'/C+L-1$，其中 C=2，L 是 CRC 计算校验位的长度.

如果原始数据的 Bits 位是偶数的话, A`=A,

如果原始数据的 Bits 位是奇数的话, A`=A+1.

2.2 LDPC for User data channel streams

基于第 1 步 CRC Calculation 的结果, 进行处理

$b_0, b_1, b_2, b_3, ..., b_{B-1}$, where $B = A + L$

2.2.1 如果 B 小于等于 maximum code block size,那么这种情景比较简单，相当于这个步骤没有做任何处理，直接 Bypass;

2.2.2 如果 B 大于 maximum code block size, 那么这种情景比较复杂，输入序列需要进行分段，并且分段之后，每个小段都需要增加额外的 CRC 校验比特位,长度 L=24.

注意 LDPC base graph 1, the maximum code block size is: 8448
LDPC base graph 2, the maximum code block size is:3840

分段数量 C 的计算公式是(向上取整):

$$C = \lceil B/(K_{cb} - L) \rceil .$$

其中 Kcb=maximum code block size, L=24, B 是第一步添加 CRC 校验位之后，得到的序列总长度。

经过这些处理之后，最终得到的序列总长度 B'，比第一步得到的序列总长度 B，增加了 C*L. //C=分段数量，L=24.
B'=B+C*L

那么，计算每个小段序列的长度成为重点:
Step 1: K'= B' /C
Step 2:
For LDPC base graph 1
{
Kb=22
}
For LDPC base graph 2

{

如果需要进行分段的话 Kb = 10

如果不需要进行分段的话 B>640，Kb = 10；

560<B<=640，Kb = 9；

192<B<=560，Kb = 8；

B<=192，Kb = 6；

}

然后从如下表格中，找到满足不等式 Kb*Z>=K'条件的最小 Z 值，记为 Zc

Table 5.3.2-1: Sets of LDPC lifting size Z

Set index (i_{LS})	*Set of lifting sizes (Z)*
0	{2, 4, 8, 16, 32, 64, 128, 256}
1	{3, 6, 12, 24, 48, 96, 192, 384}
2	{5, 10, 20, 40, 80, 160, 320}
3	{7, 14, 28, 56, 112, 224}
4	{9, 18, 36, 72, 144, 288}
5	{11, 22, 44, 88, 176, 352}
6	{13, 26, 52, 104, 208}
7	{15, 30, 60, 120, 240}

Step 3:

for LDPC base graph 1，K = 22Zc

for LDPC base graph 2，K = 10Zc

Step 4:

如果不需要进行分段的话，序列的长度也需要扩展至 K，通过在尾部增加 K-B 个填充 Bits 位（NULL，可以是'0' or '1'，都可以吗?）

如果需要进行分段的话，每个子段序列的长度都需要扩展至 K

每个子段序列包含 3 个部分：

第 1 部分，第一步添加 CRC 校验位之后，平均分段之后的数据(长度 K'- L)

第 2 部分，每一小段其后跟随的 CRC 校验位(长度 L=24，CRC 生成多项式为 $g_{\mathrm{CRC24B}}(D)$).

第 3 部分，数据填充 Bits 位(长度 K-K'，对于 NULL '0' or '1'都可以的，实际上我们只需要重点考虑地址空间的映射关系.)

[3] Channel Coding

5G NR 采用了全新的信道编码方式，即数据信道用 LDPC 编码，控制信道和广播信道用 Polar 编码。这一改进可以提高 NR 信道编码效率，适应 5G 大数据量，高可靠性和低时延的传输需求。

由于信道编码采用的是 FPGA 内部的 IP 核或 HardCopy，所以在这里没有做详细的剖析. 注意：有些控制信息可以采用 small block 超短码的信道编码

3.1 Channel coding of small block lengths

对于超短码的信道编码，输入序列是：$c_0, c_1, c_2, c_3, \ldots, c_{K-1}$，处理之后的输出序列是：$d_0, d_1, d_2, \ldots, d_{N-1}$.

根据输入序列的长度 K，可以将编码过程划分为 3 类:

3.1.1 Encoding of 1-bit information(K=1,N=Qm)

注意：由于序列长度只有 1 个 Bit 位，当调制阶数 Qm>=2 的时候才需要在序列的后面补充占位符号，[个人感悟]占位符号只是"象征性"的填充，信道编码的结果中，每个 Bit 位可以呈现为 4 种情形: [x,y,0,1].

Table 5.3.3.1-1: Encoding of 1-bit information

Q_m	Encoded bits $d_0, d_1, d_2, \ldots, d_{N-1}$
1	$[c_0]$
2	$[c_0$ y$]$
4	$[c_0$ y x x$]$
6	$[c_0$ y x x x x$]$
8	$[c_0$ y x x x x x x$]$

占位符号出现 x 和 y 的位置，Bit 位还是没有获取准确的值；需要等待在扰码阶段 scrambling stage，根据相应的公式才可以确定占位符精确的数值。具体处理过程请参见 Subclause 6.3.1.1 of [4, TS 38.211]

这样生成的调制符号（携带信息 Bits），在星座图上可以获取最大的欧式距离，从而降低解调的误码率，提升通信系统的可靠性.

3.1.2 Encoding of 2-bit information(K=2,N=3Qm)

注意：由于序列长度只有 2 个 Bit 位，当调制阶数 Qm>=4 的时候才需要在序列的后面补充占位符号，[个人感悟]占位符号只是"象征性"的填充，信道编码的结果中，每个 Bit 位可以呈现为 4 种情形: [x,y,0,1].

Table 5.3.3.2-1: Encoding of 2-bit information

Q_m	Encoded bits $d_0, d_1, d_2, \ldots, d_{N-1}$
1	$[c_0\ c_1\ c_2]$
2	$[c_0\ c_1\ c_2\ c_0\ c_1\ c_2]$
4	$[c_0\ c_1$ x x $c_2\ c_0$ x x $c_1\ c_2$ x x$]$
6	$[c_0\ c_1$ x x x x $c_2\ c_0$ x x x x $c_1\ c_2$ x x x x$]$
8	$[c_0\ c_1$ x x x x x x $c_2\ c_0$ x x x x x x $c_1\ c_2$ x x x x x x$]$

其中 c2=(c0+c1) mod 2,占位符号出现 x 和 y 的位置，Bit 位还是没有获取准确的值；需要等待在扰码阶段 scrambling stage，根据相应的公式才可以确

定占位符精确的数值。具体处理过程请参见 Subclause 6.3.1.1 of [4, TS 38.211]

这样生成的调制符号（携带信息 Bits），**在星座图上可以获取最大的欧式距离**，从而降低解调的误码率，提升通信系统的可靠性.

3.1.3 Encoding of other small block lengths(K belong[3,11],N=32)

输入序列是：$c_0, c_1, c_2, c_3, \ldots, c_{K-1}$，处理之后的输出序列是：$d_0, d_1, d_2, \ldots, d_{N-1}$.

信道编码的结果 di，统计第 i 行里面，左侧 K 个数据中满足如下逻辑条件((Mi,k==1'b1)&&(ck==1'b1))的次数；

如果是偶数的话，那么 di=0；如果是奇数的话，那么 di=1.

Table 5.3.3-1: Basis sequences for (32, K) code

i	$M_{i,0}$	$M_{i,1}$	$M_{i,2}$	$M_{i,3}$	$M_{i,4}$	$M_{i,5}$	$M_{i,6}$	$M_{i,7}$	$M_{i,8}$	$M_{i,9}$	$M_{i,10}$
0	1	1	0	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	0	0	1	1
2	1	0	0	1	0	0	1	0	1	1	1
3	1	0	1	1	0	0	0	0	1	0	1
4	1	1	1	1	0	0	0	1	0	0	1
5	1	1	0	0	1	0	1	1	1	0	1
6	1	0	1	0	1	0	1	0	1	1	1
7	1	0	0	1	1	0	0	1	1	0	1
8	1	1	0	1	1	0	0	1	0	1	1
9	1	0	1	1	1	0	1	0	0	1	1
10	1	0	1	0	0	1	1	1	0	1	1
11	1	1	1	0	0	1	1	0	1	0	1
12	1	0	0	1	0	1	0	1	1	1	1
13	1	1	0	1	0	1	0	1	0	1	1
14	1	0	0	0	1	1	0	1	0	0	1
15	1	1	0	0	1	1	1	1	0	1	1
16	1	1	1	0	1	1	1	0	0	1	0
17	1	0	0	1	1	1	0	0	1	0	0
18	1	1	0	1	1	1	1	1	0	0	0
19	1	0	0	0	0	1	1	0	0	0	0
20	1	0	1	0	0	0	1	0	0	0	1
21	1	1	0	1	0	0	0	0	0	1	1
22	1	0	0	0	1	0	0	1	1	0	1
23	1	1	1	0	1	0	0	0	1	1	1
24	1	1	1	1	1	0	1	1	1	1	0
25	1	1	0	0	0	1	1	1	0	0	1
26	1	0	1	1	0	1	0	0	1	1	0
27	1	1	1	1	0	1	0	1	1	1	0
28	1	0	1	0	1	1	1	0	1	0	0
29	1	0	1	1	1	1	1	1	1	0	0
30	1	1	1	1	1	1	1	1	1	1	1
31	1	0	0	0	0	0	0	0	0	0	0

信道编码，在这个步骤中可以精确获取每个 Bit 的数值，无需等待后续的处理步骤，比如扰码 Scrambling stage.

[4] Rate matching

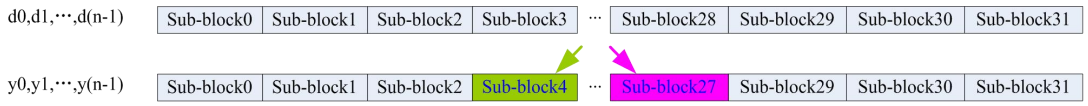
4.1 Rate matching for Polar code

极化码的速率匹配是按编码块定义的，包含子块交织、比特选择和比特交织。

4.1.1 sub-block interleaving

有待处理的输入序列 $d_0,d_1,d_2,\ldots,d_{N-1}$, Bits 位=N，先从简单的过程，进行推理:

(1) Polar 码，信道编码输出的序列是 $d_0,d_1,d_2,\ldots,d_{N-1}$，其中序列的长度 N=2^n，同时 n 的最小值是 5，因此 N 能够被 32 整除（请参见 TS 38.211 Page 12，5.3.1 相关条款），新的序列 $y_0,y_1,y_2,\ldots,y_{N-1}$。相当于将信道编码输出的序列 $d_0,d_1,d_2,\ldots,d_{N-1}$ 分为 32 个子块，随后**将子块作为整体，交换在原序列中的位置（有必要的时候），如下图所示：**



交换子块的原则，是基于如下表格进行：

Table 5.4.1.1-1: Sub-block interleaver pattern $P(i)$

i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$	i	$P(i)$
0	0	4	3	8	8	12	10	16	12	20	14	24	24	28	27
1	1	5	5	9	16	13	18	17	20	21	22	25	25	29	29
2	2	6	6	10	9	14	11	18	13	22	15	26	26	30	30
3	4	7	7	11	17	15	19	19	21	23	23	27	28	31	31

注意：新的序列 $y_0,y_1,y_2,\ldots,y_{N-1}$ 与信道编码输出的序列 $d_0,d_1,d_2,\ldots,d_{N-1}$，长度保持一致，都是 N.

4.1.2 Bit selection

这个过程，输出序列的长度是 E，根据 K/E/N 之间的关系，可以列出如下分支：

(1) 如果 E=N,那么这种情景非常简单，相当于这个步骤没有做任何处理，

直接 Bypass;

(2) 如果 E>N,这种情景,相当于将 $y_0, y_1, y_2, \ldots, y_{N-1}$ 序列存储在一个 RAM 空间: DataYForBitSelection, From Address 0 to Address N-1

//(N 能够被 32 整除)

此时，构建一个地址寄存器 ReadAddress[Width-1:0]，其中 Width =(log2E)向上取整，令 ReadAddress 从 0 自增至 E-1，并且将 ReadAddress[(log2N)-1:0]作为 RAM 空间 DataYForBitSelection 的地址，将其循环读出赋值给序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ 即可

(3) 如果 E<N,并且 K/E>7/16,那么只需要取 $y_0, y_1, y_2, \ldots, y_{N-1}$ 序列前面 E 个数据赋值给序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ 即可

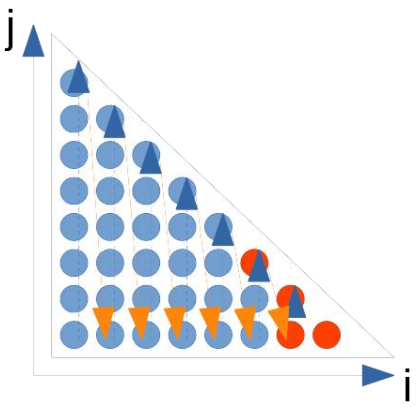
(4) 如果 E<N,并且 K/E<=7/16,那么只需要取 $y_0, y_1, y_2, \ldots, y_{N-1}$ 序列后面 E 个数据赋值给序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ 即可

4.1.3 Interleaving of coded bits

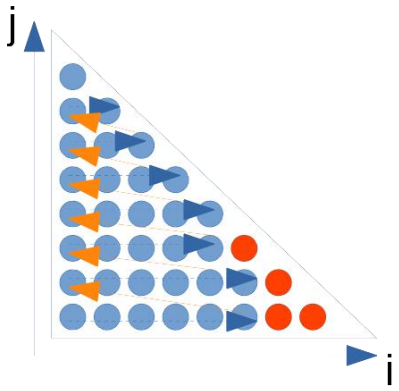
特别注意: The value of E is no larger than 8192.

[1] If (IBIL != 1)，这种情景非常简单，相当于这个步骤没有做任何处理，直接 Bypass，得到序列 $f_0, f_1, f_2, \ldots, f_{E-1}$;

[2] If (IBIL == 1)，这种情景比较复杂，首先找到满足公式 $T(T+1)/2 \geq E$ 的最小值 T. (例如 T=8,E=32)



随后将 $e_0, e_1, e_2, \ldots, e_{E-1}$ 序列，按照从下到上，从左到右的顺序，纵向填充写入至三角形里面,其中红色标记的位置是 NULL.



随后再按照从左到右，从下到上的顺序，将非 NULL 的数据从三角形里面依次读出，写入序列 $f_0, f_1, f_2, \ldots, f_{E-1}$，其中红色标记的位置是 NULL.

4.2 Rate matching for LDPC code

LDPC 码的速率匹配是按编码块定义的，包含子比特选择和比特交织。

4.2.2 Bit selection

[1]每个编码块序列，需要写入长度为 Ncb 的循环 Buffer

如果 ILBRM == 0 ,这种情况比较简单，Ncb 直接等于每个编码块序列的长度 N.

如果 ILBRM != 0 ,这种情况比较复杂，$N_{cb} = \min(N, N_{ref})$，

$N_{ref} = \left\lfloor \frac{TBS_{\text{LBRM}}}{C \cdot R_{\text{LBRM}}} \right\rfloor$

公式中的 C=分段子块的数量，RLBRM=2/3.

TBS_{LBRM} is determined according to Subclause 6.1.4.2 in [6, TS 38.214] for UL-SCH and Subclause 5.1.3.2 in [6, TS 38.214] for DL-SCH/PCH.

[2]随后确定每个编码子块，速率匹配输出序列的长度成为关键

如果**编码子块没有分配传输调度**，速率匹配输出序列的长度 Er=0;

//as indicated by CBGTI according to Subclause 5.1.7.2 for DL-SCH and 6.1.5.2 for UL-SCH in [6, TS 38.214]

如果**编码子块分配传输调度**，那么按照如下公式计算速率匹配输出序列的长度 Er.

if $j \le C' - \mathrm{mod}(G/(N_L \cdot Q_m), C') - 1$

$$E_r = N_L \cdot Q_m \cdot \left\lfloor \frac{G}{N_L \cdot Q_m \cdot C'} \right\rfloor;$$

else

$$E_r = N_L \cdot Q_m \cdot \left\lceil \frac{G}{N_L \cdot Q_m \cdot C'} \right\rceil;$$

end if

其中，**j 是分配了传输调度的编码子块的新序号**;

Qm 是 OFDM 调制阶数;

NL 是传输块 Transport Block 层映射的数量;

C'=C：if CBGTI is NOT present in the DCI scheduling the transport block

C'= the number of scheduled code blocks of the transport block, if CBGTI is present in the DCI scheduling the transport block

G: 时域和频域上分配的资源（Resource Block）里面，remove the resource elements used for DMRS and other reference signals from PDSCH/PUSCH,剩下的可用于传输用户数据 User Data 的编码 Bits 数量

随后，我们就可以得到速率匹配输出序列的长度：

如果**编码子块没有分配传输调度，那么 Er=0，**

分配了传输调度的编码子块，如果时频资源可用于传输用户数据 User Data 的 Bits 数量 G 恰好可以被 **NL*Qm 所整除，**那么对于每个编码子块来说，速率匹配输出序列的**长度相同**.

如果 G 不可以被 **NL*Qm 所整除，**那么对于编码子块来说，速率匹配输出序列的**长度，前面几个编码子块要比后面几个编码子块短(NL*Qm)个 Bits.**

[3]关于每个编码子块，根据 rv_{id} range from(0,1,2,3)传输冗余的版本号和 LDPC base graph，可以得到 k_0，请参见如下表格 Table 5.4.2.1-2:

Table 5.4.2.1-2: Starting position of different redundancy versions, k_0

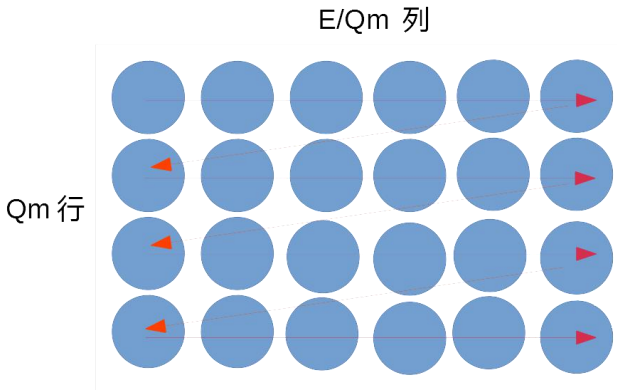
rv_{id}	k_0	
	LDPC base graph 1	LDPC base graph 2
0	0	0
1	$\left\lfloor \frac{17N_{cb}}{66Z_c} \right\rfloor Z_c$	$\left\lfloor \frac{13N_{cb}}{50Z_c} \right\rfloor Z_c$
2	$\left\lfloor \frac{33N_{cb}}{66Z_c} \right\rfloor Z_c$	$\left\lfloor \frac{25N_{cb}}{50Z_c} \right\rfloor Z_c$
3	$\left\lfloor \frac{56N_{cb}}{66Z_c} \right\rfloor Z_c$	$\left\lfloor \frac{43N_{cb}}{50Z_c} \right\rfloor Z_c$

随后将起始地址赋值为 k_0，将循环 Buffer 里面非 NULL 的数值依次顺序读出，注意当循环 Buffer 的地址自增至 Ncb-1 的时候，需要跳转至 0.

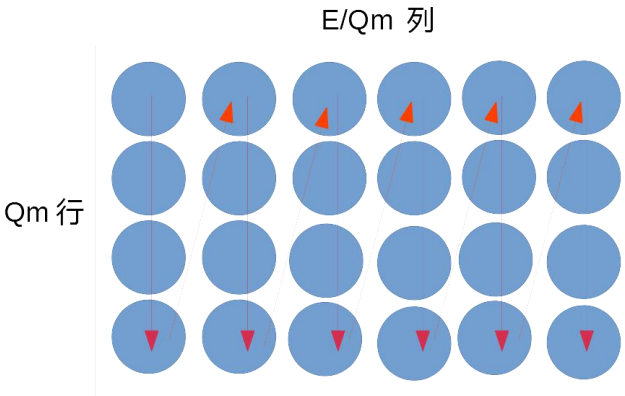
写入序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ **，数据长度为 E.**

[4]这里需要明确:每个编码子块所生成的序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ **,数据长度都是(NL*Qm)的整数倍.**

随后进行的操作是将序列 $e_0, e_1, e_2, \ldots, e_{E-1}$ 放入一个 **Qm 行，E/Qm 列的矩阵空间，进行摆放，如图所示：**



随后再从 Qm 行，E/Qm 列的矩阵空间，将数据读出并写入新的序列 $f_0, f_1, f_2, \ldots, f_{E-1}$，如图所示：



完成 LDPC 码，速率匹配的整个处理过程.

4.3 Rate matching for channel coding of small block lengths

The input bit sequence to rate matching is $d_0, d_1, d_2, \ldots, d_{N-1}$

The output bit sequence after rate matching is denoted as $f_0, f_1, f_2, \ldots, f_{E-1}$

处理过程：相当于将 $d_0, d_1, d_2, \ldots, d_{N-1}$ 序列存储在一个 RAM 空间：DataDForRateMatching, From Address 0 to Address N-1

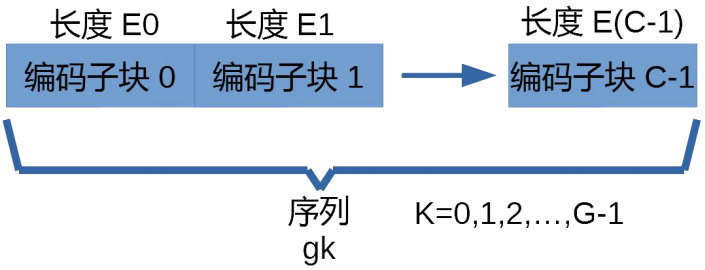
此时，构建一个地址寄存器 ReadAddress，令 ReadAddress 从 0 自增至 E-1，将(ReadAddress MOD N)作为 RAM 空间

DataDForRateMatching 的读地址，将其循环读出赋值给序列 $f_0, f_1, f_2, \ldots, f_{E-1}$ 即可.

[5] Code block concatenation

Rate Matching 模块输出的结果是 C 个编码子块，其中每个编码子块的序列长度分别是 Er.

Code Block concatenation 所实现的功能就是将这些编码子块连接起来而已,得到新的序列 g_k for $k = 0, \ldots, G-1$，如图所示：



其中 G=E0+E1+…+E(C−1).

[6] Scrambling

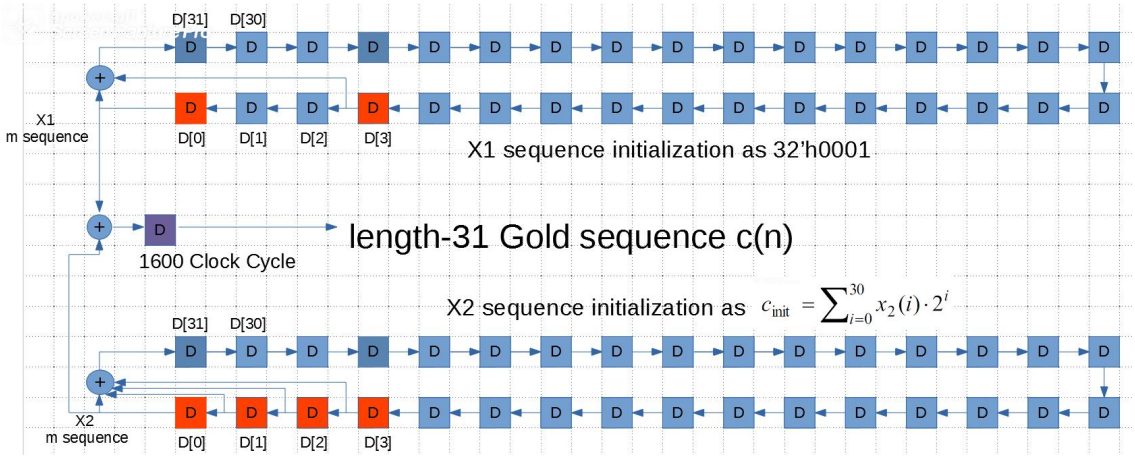
扰码过程是基于码字 Codeword 进行的，物理信道上所传输的码字 q 可以记为：$b^{(q)}(0), \ldots, b^{(q)}(M_{\text{bit}}^{(q)} - 1)$，其中 $M_{\text{bit}}^{(q)}$ 表示此码字的数据 Bits 长度，扰码过程输出的序列为：$\tilde{b}^{(q)}(0), \ldots, \tilde{b}^{(q)}(M_{\text{bit}}^{(q)} - 1)$.

特别注意：如果在信道编码阶段涉及 Channel coding of small block lengths 的话，那么需要根据占位符 UCI placeholder bits 来对扰码的输出进行控制.

如果占位符是 x 的话，那么扰码 Bit 的结果是 1，如果占位符是 y 的话，那么扰码 Bit 的结果是前面 1 个 Bit 的重复.

如果在信道编码阶段没有涉及 Channel coding of small block lengths，直接将码字序列和伪随机序列 $c^{(q)}(i)$ 异或即可.

伪随机序列 $c^{(q)}(i)$,由长度为 31 的 Golden 序列定义，此序列是由两个小 m 序列通过异或运算生成的，逻辑设计的结构，如图所示：



其中 m 序列 x2,需要初始化为 $c_{\text{init}} = n_{\text{RNTI}} \cdot 2^{15} + n_{\text{ID}}$.

如果来自上层的参数 dataScramblingIdentityPUSCH 已经得到配置的话，那么 $n_{\text{ID}} \in \{0,1,\ldots,1023\}$ 等于参数 dataScramblingIdentityPUSCH, RNTI 等于 C-RNTI, MCS-C-RNTI or CS-RNTI,在公共搜索空间，传输没有计划使用 DCI 格式 0_0,否则 $n_{\text{ID}} = N_{\text{ID}}^{\text{cell}}$.

n_{RNTI} 对应于与 PUSCH 传输相关联的 RNTI(无线网络临时标识符),请参见 clause 6.1 of [6, TS 38.214].

[7] Modulation

调制实质，根据 modulation schemes 确定调制阶数 Qm,然后将 Qm 个相邻的 Bits 位捆绑在一起，映射为 1 个复数调制符号.

Table 6.3.1.2-1: Supported modulation schemes.

Transform precoding disabled		Transform precoding enabled	
Modulation scheme	Modulation order Q_{m}	Modulation scheme	Modulation order Q_{m}
		π/2-BPSK	1
QPSK	2	QPSK	2
16QAM	4	16QAM	4
64QAM	6	64QAM	6
256QAM	8	256QAM	8

显然，复数调制符号的数目 = 码字所包含的 Bits 数目/调制阶数 Qm.