简介

H265内部的Class封装一般分为两种： 一种是编码类，一种是数据类。

编解码类:
TDecTop :好像是最顶层的解码器，然后内部封装了几乎下面所有的编解码类别。

TDecGop：解码GOP为单位的解码器类内部也封装了各类的解码器，包括：熵编码器，环路滤波器等等

TDecSlice 就是在TDecGOP中的成员，用于解码图像的Slice，

TDecCu： 解码CU的解码器类，内部包含存储CU块级别Res和Rec内存空间，如果要提取Pred信息，需要在这个解码器内部对Pred信息进行维护。

TComTrQuant: 量化器的类，在编码CU的过程中，交叉了其他模块的解码器，这个是关于量化的模块。

TComPrediction： 预期器，可能对于帧内预测以及帧间预测的内容，都会在这个类中进行运算。

TDecEntropy : 熵编码器，用这个类进行熵编码以及熵解码吧，理论上需要编码的信息有：预测模式，残差信息。

数据类:

TComPicYuv：文件中读取出来的YUV数据，最后的YUV数据rec的格式是这个格式，与**外部数据的吞吐**也都是从这个数据里走的。

TComPic： 这个类包含了图像的一整帧的数据，包含了由TComPicYuv复制得到的一帧的图像，以及包含了TComDataCU，这个类会存储最佳的CU数据，这个会重复存储所有的CU数据吗？具体存储是啥需要探究。这个类会为rec，org，和true_org创建内存空间，这三个所属的类别的就是：TComPicYuv()。 可以通过这个类去创建一些属于TComPicYuv的内存，通过这个Pic这个类操作PicYuv这个类。

TComYuv： 一般用于存放Rec， Res的YUV类的数据，可以通过指定大小来申请一定的内存空间。存储编码过程中的中间数据以及rec。这部分的数据一般是在编码器的类内部的，用来做临时的存储，一般情况情况下，编码阶段结束的话会被释放掉。

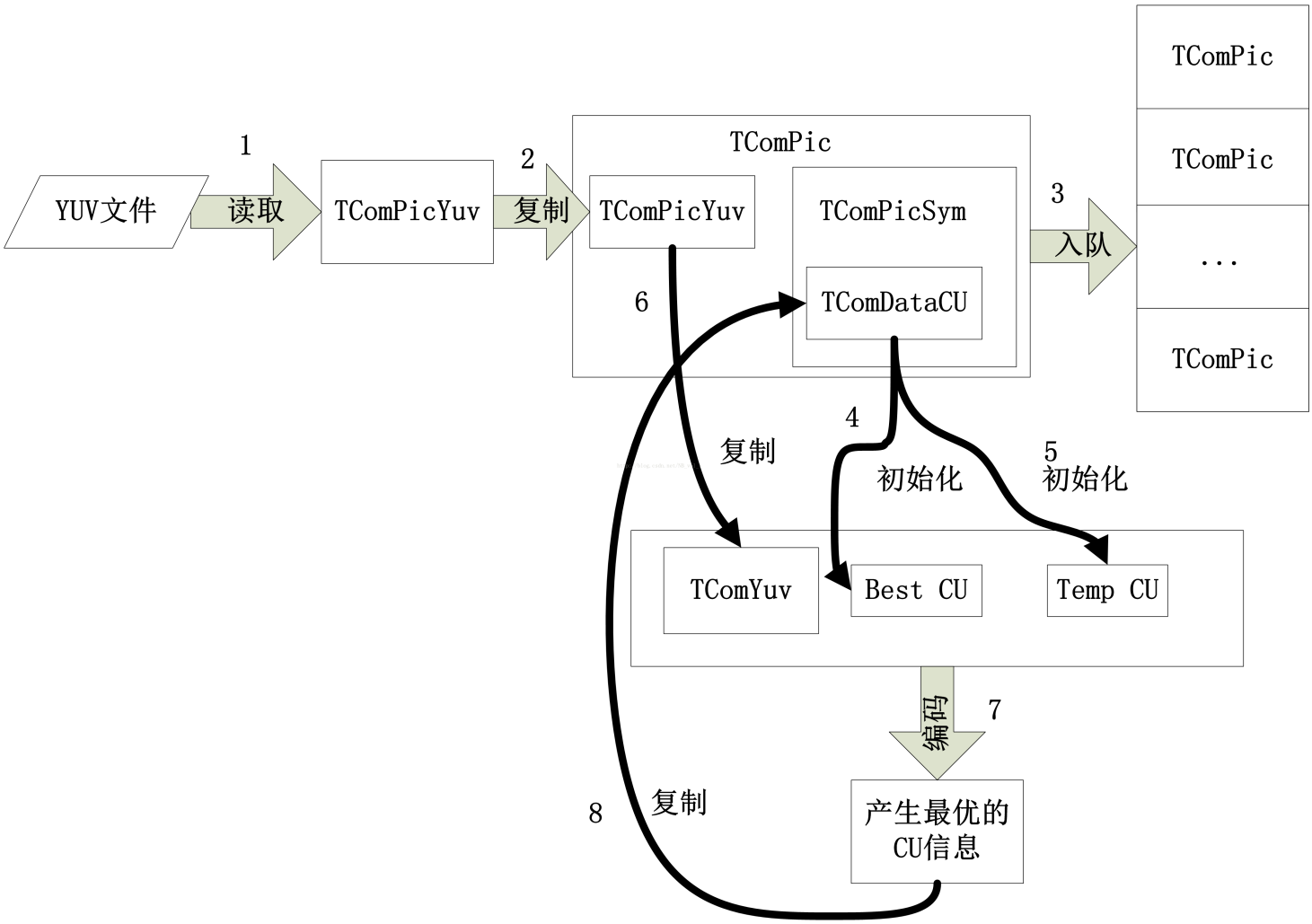
TComDataCU :可能存放CU块的数据信息，包含了CU的属性，PCM flag，Depth信息等等。

TComPicSym：不太清楚这个东西是干啥的。

TcomTU： 包好了TU块的数据内容。

TComTURecurse

数据类的流动图如下：



1. HM首先使用TComPicYuv保存从文件中读取出来的YUV数据,也就是TComPicYuv这里面应该包含了整个YUV文件的数据。
2. 利用TComPicYuv构造TComPic，并把YUV数据复制给他（TComPic包含了TComPicYuv成员），其中包含了数据（TComPicYuv），以及图像信息（TComPicSym，在TComPicSym中还包含了TComDataCU），实际上这里会有一个队列在，毕竟是多帧的图像，有用到信息的扔进队列里面。
3. 处理图像队列中的每一个TComPic，实际是处理TComPic中的每一个CTU/CU（存放在TComPicSym中）
4. 从TComPic中取出每一个CTU（用TComDataCU表示），调用xCompressCU进行处理
5. 根据原始的CTU初始化TEncCu中的TComDataCU（最优的和临时的），这两个在编码的时候会使用
6. 把TComPic的数据复制给TComYuv对象，表示编码过程中的数据（原始，预测、残差、重建等），也就是TComYuv对象其实表示编码过程中产生的东西。确实，残差信息之类的好像也不好存储在TComPicYuv中。需要额外申请一个辅助空间对一些信息进行保存。
7. 进行编码，tempCU用于编码过程中，bestCU用于保存最优信息
8. 产生的最优信息会被复制回TComPicSym中

输入的流情况

1. 在TDecGOP层次，会调用**TDecGop::decompressSlice**对Slice层次的信息进行解压，其中主要调用了**TDecSlice::decompressSlice**函数对程序中确定切片的内容进行解码,这个函数会如下 4.1.1 所言调用decodeCtu函数，随后调用decompressCtu函数进行解码。
2. 在decodeCtu中调用TDecCu::XdecodeCU,对CU的信息进行熵解码，获得关于预测模式之类的东西，然后经过TDecCu::xDecompressCU，完成对输入的图像的重建，但是这里的输出是没滤波之前的重建图像。
3. 在完成解码后，对图像进行滤波，执行函数TDecTop::executeLoopFilters，这个函数主要调用了TDecGop::filterPicture对图像进行滤波，然后得到滤波后的图像。

TDecGop

TDecGop::decompressSlice

TDecSlice

包含了TDecCu编码器用于后续的CTU级别的解码，TDecEntropy熵解码器，TDecSbac类（这个是啥？）

TDecSlice::decompressSlice

用于完成一帧图像的解码。其中涉及了大量熵编码的内容.输入是：

1. TComInputBitstream的bit流
2. TComPic pcPic
3. TDecConformanceCheck

输入是pcPic，根据slice的cur位置，定位当前正在解码的slice是属于哪一块的。

然后循环方位slice内的CTU数据，创建TComDataCU的对象pCtu，然后对这个CTU进行decode和decompress。

主要流程如下：

1. 初始化：CTU、熵解码器等。
2. 如果是dependent slice，利用之前的slice优化上下文模型。

3. 遍历所有CTU进行解码：
   3.1 初始化CTU。
   3.2 设置熵解码器，加载上下文模型。
   3.3 SAO。
   3.4 调用**decodeCtu**（详见HEVC代码学习39：decodeCtu和xDecodeCU函数）和**decompressCtu**解码CTU。
   3.5 存储第二个CTU的上下文模型。
   3.6 如果是最后一个CTU，不分析剩余比特；否则分析剩余比特。
4. 如果是dependent slice，加载上一个slice结束时的上下文模型。

TDecCu

列出几个需要关注的核心函数以及其作用。图像先经过XdecodeCU

TDecCu::XdecodeCU

该函数通过迭代完成一个CTU的所有CU的解码工作，主要是各种预测信息的解码，应该是预测模式，是xEncodeCU（详见
HEVC代码学习35：xEncodeCU函数）的逆处理。有4个输入参数：（当前CU，分块索引，深度和标识是否是slice最后一个CTU的flag），

TDecCu::xDecompressCU

xDecodeCU只是用来解码各种flag和系数，没有进行预测重构。在完成相关信息的解码之后，会在decompressCtu和xDecompressCU中完成的预测重构。该函数中会递归进行CU的预测重构。在之前的decodeCtu中将预测重构所需的划分flag、预测模式索引、系数等信息解码，这里只需要根据这些信息进行对应的划分、预测重构即可。

输入是：

1. CTU块（但是由于后续会进行递归调用，即也是CU块的数据）
2. CU块的初始地址位置
3. 当前CU块的深度信息值。

因为递归的问题，所以比较难理解，假设以最小的8x8的CU块为例子，会把8X8块的数据从Pic中复制到TDecCu中存储CU块的内存上。然后将这部分数据输入到。

然后根据预测模式选择帧内预测还是帧间预测。会在进入预测之前，把TDecCu类内的Resi的数据给清空了。

1. 帧间预测：
    xReconInter(m_ppcCU[uiDepth], uiDepth);
2. 帧内预测：
    xReconIntraQT(m_ppcCU[uiDepth], uiDepth);)

在得到每个CTU的预测结果之后，填充PCM的buffer，然后，通过TDecCu::xCopyPic 讲TDecCu内的Rec数据复制到piPic中，完成了整个解码闭环。

TDecCu::xReconPCM

获得来自PCM模式传输下的数据情况，PCM模式好像是直接传输一个flag标志位置，表示对于预测块和当前块的差距特别小，不需要额外的码流表示。

TDecCu::xReconInter

帧间的内容暂时先放一放！！！

TDecCu::xReconIntraQT

帧内编码开始，输入是

1. 当前CU的数据块,可以从最小的CU块理解。
2. 当前CU的深度值。

函数进去先对CU进去先对这个CU进行判决，判断是不是PCM模式,如果是，则直接调用xReconPCM函数进行重构，如果不是，则下一步.

然后进入一个 tuRecurseCU(pcCU,0),
TComTURecurse(tuRecurseCU,false,(uiInitTrDepth==0)?TComTU::DONT_SPLIT : TComTU::QUAD_SPLIT)

这里好将CU划分为对应的PU，然后通过一个循环，不断执行xIntraRecQT()得到重构的结果，我猜测，上面的结果可能得到了关于CU块划分的列表? 类似于这种的东西的吧。

随后执行函数：xIntraRecQT进行重构，

TDecCu::xIntraRecQT

这个类用于拿到TU之后，获得TU的预测值，然后对图像进行重构。
输入是：

1. TDecCu::m_ppcYuvRec

2. TDecCu::m_ppcYuvRec
3. TDecCu::m_ppcYuvResi
4. chanType --分量类型
5. PU 据partSize 将CU划分为对应的PU

这里输入一致是m_ppcYuvRec前者是在函数中体现为Rec，而后者体现为Pred信息，因为会先得到Pred信息，但是这部分信息可以在拿到Resi信息后转化为Rec，所以不需要维护Pred信息，避免内存的浪费。

TDecCu::xIntraRecBLK

上面两层的函数包括：TDecCu::xIntraRecQT 和 TDecCu::xIntraRecQT 两个函数好像并没有真正地进行实质地解码工作，实际上还是有这个函数进行解码的。

这个函数的输入和 TDecCu::xIntraRecQT 是一样的，

1. TDecCu::m_ppcYuvRec
2. TDecCu::m_ppcYuvRec
3. TDecCu::m_ppcYuvResi
4. chanType --分量类型
5. PU 关于Pu的信息的，TCom PU类型的，应该就是PU的数据

根节点都是CU，编码过程中是CU->PU,CU->TU，在帧内模式的情况下如果PU是N*N,那么CU->TU是一定会劈分一次

解码过程，先判断出帧内情况，所以根据PU先判断CU是否劈分了一次，之后再根据深度信息得到TU

经过一个 TComTURecurse() 又把PU来自于（tuRecurseCUWithPU）分解成subTURecurse，这里是一个subTU的集合包好像是TU，然后重复执行xIntraRecBLK函数，也就是自己。

调用m_pcPrediction (TDecCu内部的预测器类)的predIntraAng函数拿到预测的信号.

随后进行逆变换，通过到调用m_pcTrQuant->invTransformNXN() 调用了量化器的invTransformNXN() 函数有点奇怪，还是说，逆变换功能被继承到了量化器的类里面，这里是可以得到Resi的值的。

得到上述两个步骤拿到预测值和残差值之后，就开始了重构工作，

对应位置相加，在TDecCu中写入的rec的结果，相加的结果会经过ClipDB进行过滤，得到rec的输出。这部分数据依旧记录在TDecCu内部申请的Rec的内存中。

TDecCu::xCopyPic

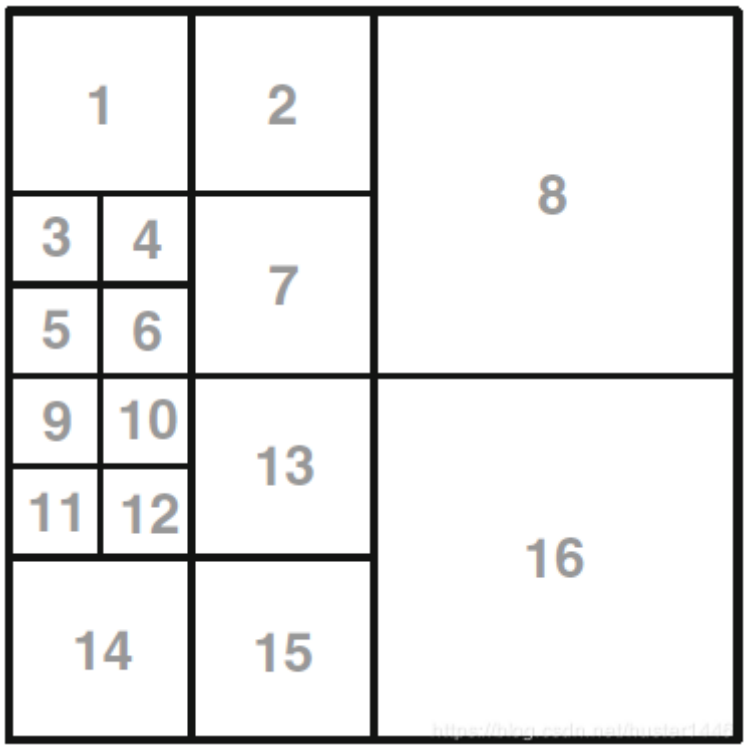
这个函数比较重要，应该是把编码器里面存储的Rec，Res 给弄到最终的图像的数据类上，可以搞懂这个函数，这样就能知道里面的处理单元的问题了。

代码中变量记录

uiAbsPartIdx

ui是unsinged int 这个大家应该都知道，那Part是什么意思呢？其实是每个4×4单元被称为一个Part。那Abs是表示CU在CTU中的绝对下标，同样也有Rel表示相对的下标，如一个CU有可能会被划分成不同深度的TU，TU就可以通过GetTransformDepthRel()函数来获得TU相对CU的深度。

Z字形扫描顺序在编码中经常用到，如下图是CTU划分成CU后的Z字形编码顺序，CU按此顺序逐个进行编码

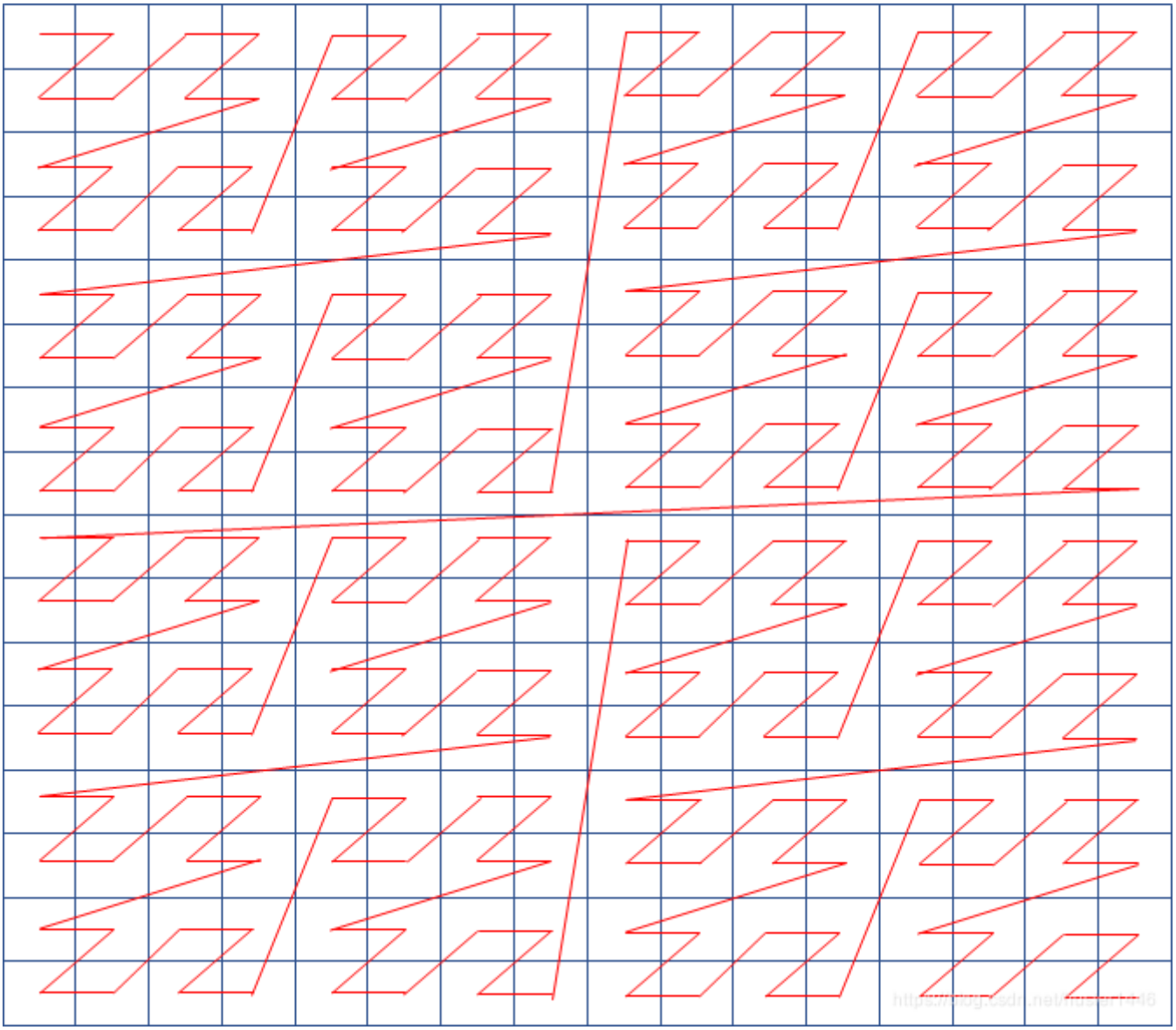


但是CU在CTU中的index却不是简单等于这个编码顺序，因为不确定一个CTU会划分成多少个CU，只是知道编码顺序并不能确定CU的位置。

因为编码过程中最小的单元是4×4大小，所以可以将CTU划分成若干个4×4大小的块，进而确定CU的位置，这也是uiAbsPartIdx变量的作用。

CU的Z-index

默认CTU大小是64×64，那么就可以划分成256个4×4大小的块。将这256个块按Z字形顺序编号，得到其各自的Z-Order。如下图中每个小方块大小为4×4，红线代表Z字形扫描的顺序。



对应的每个4×4单元的Z-index为：

0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85
2	3	6	7	18	19	22	23	66	67	70	71	82	83	86	87
8	9	12	13	24	25	28	29	72	73	76	77	88	89	92	93
10	11	14	15	26	27	30	31	74	75	78	79	90	91	94	95
32	33	36	37	48	49	52	53	96	97	100	101	112	113	116	117
34	35	38	39	50	51	54	55	98	99	102	103	114	115	118	119
40	41	44	45	56	57	60	61	104	105	108	109	120	121	124	125
42	43	46	47	58	59	62	63	106	107	110	111	122	123	126	127
128	129	132	133	144	145	148	149	192	193	196	197	208	209	212	213
130	131	134	135	146	147	150	151	194	195	198	199	210	211	214	215
136	137	140	141	152	153	156	157	200	201	204	205	216	217	220	221
138	139	142	143	154	155	158	159	202	203	206	207	218	219	222	223
160	161	164	165	176	177	180	181	224	225	228	229	240	241	244	245
162	163	166	167	178	179	182	183	226	227	230	231	242	243	246	247
168	169	172	173	184	185	188	189	232	233	236	237	248	249	252	253
170	171	174	175	186	187	190	191	234	235	238	239	250	251	254	255

而一个CU的Z-order则是以当前CU左上方的4×4单元的Z-index表示，也就是我们经常看到的变量uiAbsPartIdx存储的值。

接口：

```
UInt uiAbsPartIdx =pcCU->getZorderIdxInCtu()
```

并且对应的Z-scan 扫描和光栅扫描表格：

```
Zscan_idx=g_rasterToZscan[raster_idx]
Raster_idx=g_zscanToRaster[zscan_idx]
```

RasterToZscan

0	1	4	5	16	17	20	21	64	65	68	69	80	81	84	85
2	3	6	7	18	19	22	23	66	67	70	71	82	83	86	87
8	9	12	13	24	25	28	29	72	73	76	77	88	89	92	93
10	11	14	15	26	27	30	31	74	75	78	79	90	91	94	95
32	33	36	37	48	49	52	53	96	97	100	101	112	113	116	117
34	35	38	39	50	51	54	55	98	99	102	103	114	115	118	119
40	41	44	45	56	57	60	61	104	105	108	109	120	121	124	125
42	43	46	47	58	59	62	63	106	107	110	111	122	123	126	127
128	129	132	133	144	145	148	149	192	193	196	197	208	209	212	213
130	131	134	135	146	147	150	151	194	195	198	199	210	211	214	215
136	137	140	141	152	153	156	157	200	201	204	205	216	217	220	221
138	139	142	143	154	155	158	159	202	203	206	207	218	219	222	223
160	161	164	165	176	177	180	181	224	225	228	229	240	241	244	245
162	163	166	167	178	179	182	183	226	227	230	231	242	243	246	247
168	169	172	173	184	185	188	189	232	233	236	237	248	249	252	253
170	171	174	175	186	187	190	191	234	235	238	239	250	251	254	255

ZscanToRaster

0	1	16	17	2	3	18	19	32	33	48	49	34	35	50	51
4	5	20	21	6	7	22	23	36	37	52	53	38	39	54	55
64	65	80	81	66	67	82	83	96	97	112	113	98	99	114	115
68	69	84	85	70	71	86	87	100	101	116	117	102	103	118	119
8 9	9 9	24	25	10	11	26	27	40	41	56	57	42	43	58	59
12	13	28	29	14	15	30	31	44	45	60	61	46	47	62	63
72	73	88	89	74	75	90	91	104	105	120	121	106	107	122	123
76	77	92	93	78	79	94	95	108	109	124	125	110	111	126	127
128	129	144	145	130	131	146	147	160	161	176	177	162	163	178	179
132	133	148	149	134	135	150	151	164	165	180	181	166	167	182	183
192	193	208	209	194	195	210	211	224	225	240	241	226	227	242	243
196	197	212	213	198	199	214	215	228	229	244	245	230	231	246	247
136	137	152	153	138	139	154	155	168	169	184	185	170	171	186	187
140	141	156	157	142	143	158	159	172	173	188	189	174	175	190	191
200	201	216	217	202	203	218	219	232	233	248	249	234	235	250	251
204	205	220	221	206	207	222	223	236	237	252	253	238	239	254	255