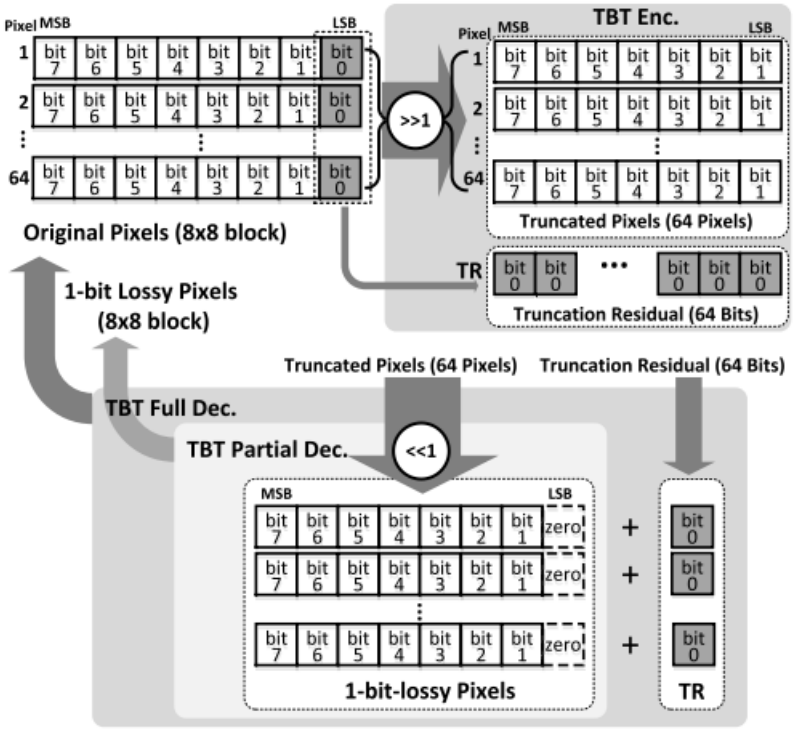
帧压缩算法

一、算法流程

1. TBT(Tail Bit Truncation)

TBT 通过对输入的图片每个像素单元进行右移操作，从而移除图像的最低位部分，具体如图一所示。以 1bit 移位为例，当输入图片的为 8*8 的区块大小时，移位得到的尾数据为 64bit，剩下的图像像素单元数值位于(0,127)的区间内，总体而言，经过平移后的数据分为两类：分割后的数据(truncated pixels)和分割后的残差(TR)。

TBT 的图像处理过程是无损的，不会对于图像效果产生任何影响，而 TBT 的主要功能是为了降低后续的残差，经过移位操作，在后续无论是求解 BP 残差，还是 NP 部分与预测值的残差，都会降低为原来的 1/2，这样在进一步的残差编码中，就有更大的空间降低编码长度，提升压缩比例，从而在不影响图像质量的情况下，更好的进行帧压缩。同样，通过 TBT 操作，可以防止 IBP 部分在计算预测值的时候出现溢出，而若按照输入图像 8bit 每像素点直接计算预测值，需要先分别移位再相加，这样会造成预测效果的降低，而即便是绝对值为 1 的偏差，也可能会带来整个区块编码长度的大幅上升。



图一 TBT 的具体操作过程

2. IBP(In Block Prediction)

将 TBT 经过移位后的数据传入 IBP 模块，在 IBP 模块会将输入的 8*8 图像块分为 IP、BP 和 NP 三个部分，用 X 来表示输入的 8*8 图像，那么 IP 对应着 X(0,0)，BP 对应

着最一行和第一列剩下的 14 个元素，即 X[1:8,0]和 X[0,1:8]，NP 则为剩下的 7*7 部分元素 X[1:8,1:8]。IP 会按照原值进行存储，而 BP 则为了降低存储空间，会先沿着行和列求出相应的差值 BP_res，对应的计算公式如下：

$$\mathrm{BP\_res}[\mathrm{i},0] = X[i,0] - X[i-1,0](0 < \mathrm{i} < 8)$$

$$\mathrm{BP\_res}[0,\mathrm{i}] = X[0,i] - X[0,i-1](0 < i < 8)$$

对于 NP 求残差则相对复杂，通过求出像素实际值得大小与预测值得差，得到相应的残差，相应的计算公式如下：

$$\mathrm{NP\_res}[\mathrm{i},\mathrm{j}] = X[i,j] - Prediction[i,j](0 < \mathrm{i} < 8, 0 < \mathrm{j} < 8)$$

而预测值得求解方式，根据传入到 IBP 模块的 intra_mode 以及相应的三种 IBP mode，来计算出三种预测值，最终会得到三个 7*7 的预测图，通过 mode decision 算法来决定最优预测值。对应的 intra mode 和 IBP mode 表如下：

TABLE II
MODE MAPPING FOR HEVC INTRA MODE AND IBP MODE

Intra Mode	IBP Mode	Intra Mode	IBP Mode
Planner	4,5,7	11-18	0,2,6
DC	4,5,7	19-26	1,5,6
2-10	0,4,5	27-34	1,3,4

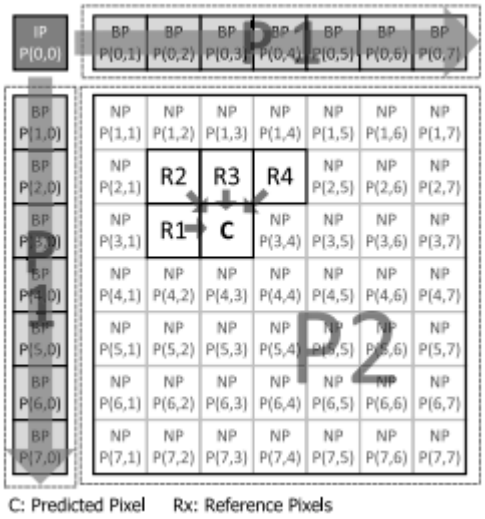
图二 intra mode 与 IBP mode 映射表

TABLE I
IBP MODE

Mode	Prediction (C=)
0	R1
1	R3
2	(R1+R2)/2
3	(R3+R4)/2
4	(R1+R4)/2
5	(R1+R3)/2
6	((R1+R2)/2+R3)/2
7	((R1+R2)/2+(R3+R4)/2)/2

图三 IBP mode 的计算公式

与 IBP mode 计算公式表相对应的空间映射图，对应如下：



图四 IBP mode 空间映射表

3. VLC(Variable Length Coding)

将 IBP 模块的 BP 残差和 NP 残差拼成一个 8*8 的残差区块之后，需要进行编码，以降低存储空间，在《In-Block Prediction-Based Mixed Lossy and Lossless Reference Frame Recompression for Next-Generation Video Encoding》里，单独设计了一个编码方式，对于 8*8 的残差块进行分割，分成 4*4 的四个残差块，然后通过比较每个 4*4 区块内的残差绝对值，求出最大值，就可以在编码表中找到相应的编码方式，具体如图五所示。

TABLE III
PROPOSED VLC TABLE

	01	**10**	**00**	**110**
Max. value	*0*	*1*	*2*	*3~4*
0	-	1	01	001
±1		0S	1S	01S
±2			00S	10S
±3				11S
±4				000S
	1110	**11110**	**111110**	**111111**
Max. value	*5~8*	*9~16*	*17~32*	*>32*
0	0001	00001	00001	
±1	001S	0001S	0001S	
±2	010S	0010S	0010S	
…	…	…	…	
±7	111S	0111S	0111S	
±8	0000S	1000S	1000S	
…		…	…	
±11		1011S	1011S	xxx…xxS
±12		1100S	11000S	
…		…	…	
±15		1111S	11011S	
±16		00000S	1110000S	
…			…	
±31			1111111S	
			0000000S	

图五 VLC 编码表

这个编码表根据对应残差区块最大绝对值 p，将 p 大的编码长度变长，将 p 小的区块，每个元素编码长度变短，这样可以在尽量小的存储预算的情况下，完成数据压缩，而这样的编码分布方式和实际残差分布也十分相近，0 和±1 出现的频率达到 60%~80%，0 和±1 的编码方式对于压缩率的影响程度远大于其他值，因此越小的残差，对应的编码长度尽量的越小，而大值虽然编码长度长，但是出现的频率低，因此可以不用太过在意。

二、算法改进

1. 对于 r4 的算法改进

在图四中关于 IBP mode 的计算方式，当需要预测的像素点位于行末时，r4 的位置溢出，这样每次计算预测值都要先判断所在元素的位置，若对应位置为行末，那么令 r4=0，对应的 python 代码如下：

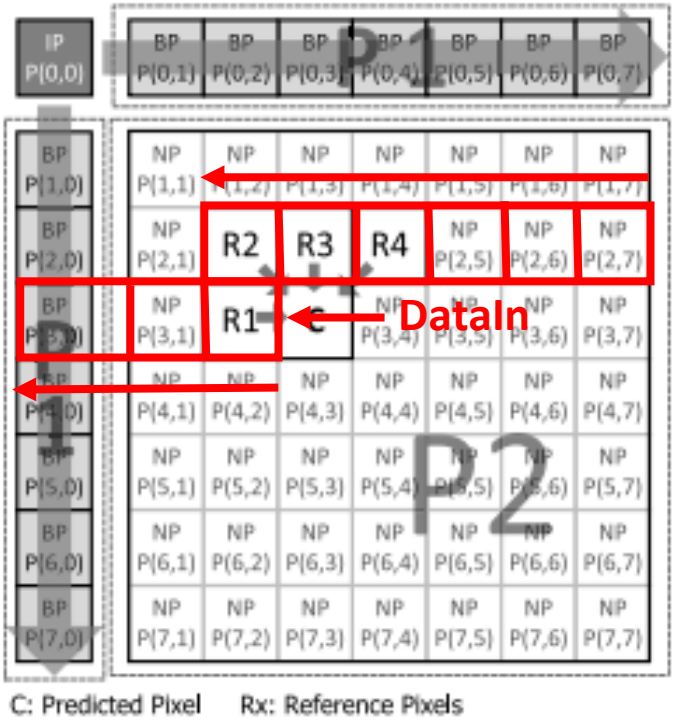
```
# NP residual
modelist = domain[intra_mode]
predicts = np.zeros([len(modelist),n-1,m-1])
BP_mode = None
best_predict = None
p = 0.0
for k in range(len(modelist)):
    for i in range(1,n):
        for j in range(1,m):
            r1 = X[i,j-1]
            r2 = X[i-1,j-1]
            r3 = X[i-1,j]
            if j+1>=8:
                r4 = 0
            else:
                r4 = X[i-1,j+1]
            predicts[k,i-1,j-1]=ibp_mode[modelist[k]](r1,r2,r3,r4)
    if printall:
        print k
        print predicts[k]
    current = PSNR(X[1:,1:],predicts[k],l)
    if current > p:
        p = current
        BP_mode = modelist[k]
        best_predict = k
```

图六 对于预测过程进行仿真的预测代码

但是这样的 r4 计算办法，带来的问题是除了要在硬件设计上多一个行末判断，还会造成 IBP mode 7 的量纲变为原本的 3/4，本身就存在量纲问题的还有 IBP mode 6，对

于 8*8 的图像本身的像素点变化就很小，经过 TBT 处理的像素点尺度变化会更小，1/4 的量纲变化，会直接带来预测结果的不准确，造成相应的残差更大，编码长度也会更长。

因此，为了很好的适应硬件设计，在对于行末元素进行预测的时候，把 r4 的位置对应第二行行首元素，这样通过一个 9 个寄存器构成的移位寄存器链就可以完成 r1、r2、r3 和 r4 的记录工作，相应的示意图如下：



图七 移位寄存器链

这样 r4 不会再因为位于行末，而造成量纲的问题。对应的 python 代码位于图八。

2. 对于 mode decision 的算法改进

找到 mode decision 的具体方案，因此初始时采用了计算 PSNR 的方式进行算法拟合，具体对应的公式如下：

$$\mathrm{PNSR} = 10 * \log_{10} \frac{(2^n - 1)^2}{MSE}$$

$$\mathrm{MSE} = \frac{1}{49} \sum_{i=1}^{8} \sum_{j=1}^{8} (X[i,j] - Prediction[i,j])^2$$

这样通过计算预测图和原图的相似程度，从而判读预测效果的好坏，PSNR 指数越大，预测效果越好。PSNR 对应的 python 代码见图九，为了防止 MSE 为零，因此在后面加上了 1e-8 进行修正。

```
# NP
old_X = X[:,:]
X = X.flatten()
modelist = domain[intra_mode]
predicts = zeros([len(modelist),n-1,m-1])
BP_mode = None
best_predict = None
p = 0.0
for k in range(len(modelist)):
    for i in range(1,n):
        for j in range(1,m):
            idx = i*m+j
            r1 = X[idx-1]
            r2 = X[idx-1-m]
            r3 = X[idx-m]
            r4 = X[idx-m+1]
            predicts[k,i-1,j-1] = ibp_mode[modelist[k]](r1,r2,r3,r4)
    if printall:
        print k
        print predicts[k]

    current = PSNR(old_X[1:,1:], predicts[k], 1)
    if current > p:
        p = current
        best_predict = k
        BP_mode = modelist[k]
```

图八 对于移位寄存器链仿真的 python 代码

```
def PSNR(X,Y,l):
    MAX = 2**l-1
    delta = X-Y
    MSE = np.sum(delta * delta)/(X.shape[0]*X.shape[1]) + 1e-8
    log10 = lambda x:log(x)/log(10)
    return 10*log10(MAX**2/MSE)
```

图九 PSNR 函数对应的 python 代码

然而 PSNR 算法虽然作为 mode decision 算法性能，但是硬件实现过程中计算量大，尤其是 log10，用查找表的方式构造的话，占用资源很大，但是对应的使用频率却不高，因此，PSNR 对于硬件实现是不合适的。

根据图五对应的残差编码方式可以看出，每一个 4*4 像素块编码方式的决定性因素是残差绝对值的最大值，因此，从三张预测图片中选择最合适的预测图，最关键的因素是这张预测图对应的残差绝对值最大值是三张图中最小的，相应的 python 算法实现代码见图十。

```
def IBP(X, intra_mode, l, printall = False):
    n, m = X.shape
    res = np.zeros(X.shape,dtype=np.int8)

    # IP
    IP = X[0,0]

    # BP
    for i in range(1,n):
        res[i,0] = X[i,0] - X[i-1,0]
    for i in range(1,m):
        res[0,i] = X[0,i] - X[0,i-1]

    # NP
    old_X = X[:,:]
    X = X.flatten()
    modelist = domain[intra_mode]
    predicts = zeros([len(modelist),n-1,m-1])
    BP_mode = None
    best_predict = None
    p = 255
    for k in range(len(modelist)):
        for i in range(1,n):
            for j in range(1,m):
                idx = i*m+j
                r1 = X[idx-1]
                r2 = X[idx-1-m]
                r3 = X[idx-m]
                r4 = X[idx-m+1]
                predicts[k,i-1,j-1] = ibp_mode[modelist[k]](r1,r2,r3,r4)
        if printall:
            print k
            print predicts[k]

        current = abs(old_X[1:,1:] - predicts[k]).max()
        if current < p:
            p = current
            best_predict = k
            BP_mode = modelist[k]

    res[1:,1:] = old_X[1:,1:] - predicts[best_predict]
    return IP, BP_mode, res
```

图十 最终 IBP 模块对应的 python 代码实现

3. VLC 部分编码方式的修改

VLC 部分原本的编码方式，需要先行比较得到 4*4 区块内残差绝对值，才能够进行数值编码，这样带来的问题是，在编码前至少需要等待 3*8+4=28 个时钟周期，才能

够进入到编码阶段，而在 IBP 模块按照原本的 mode decision 方式，也需要将所有的预测值全部求出，才能够整体确定需要编码的偏差 8*8 区块，进而传给下一个模块进行编码，这样一来，IBP 和 VLC 模块单是花费在等待上的时间就有 64+28=92 个时钟周期，同时因为要进行控制逻辑区块编码的选择以及编码模式的选择，由此产生的控制逻辑更加复杂，这样给硬件编码带来的压力，会一定程度上直接导致电路性能降低。因此，这里另外的选择了一个编码方式：指数哥伦布编码方式(Exp-Golomb)。

指数哥伦布的编码方式求解方式，具体来说如下：

$$\mathrm{m} = \log_2(|num| + 1)$$

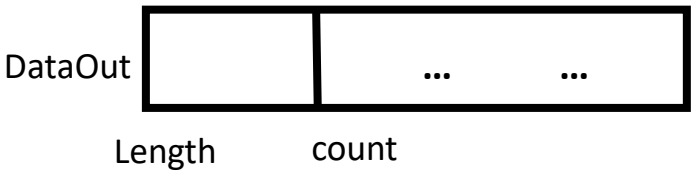
$$\mathrm{r} = |num|\&(2^m - 1)$$

初始的 m 位 1 加上 0，再加上 r 以及 num 对应的符号的二进制表达，以 0~11 为例：

0	0	±6	11011S
±1	100S	±7	1110000S
±2	101S	±8	1110001S
±3	11000S	±9	1110010S
±4	11001S	±10	1110011S
±5	11010S	±11	1110100S

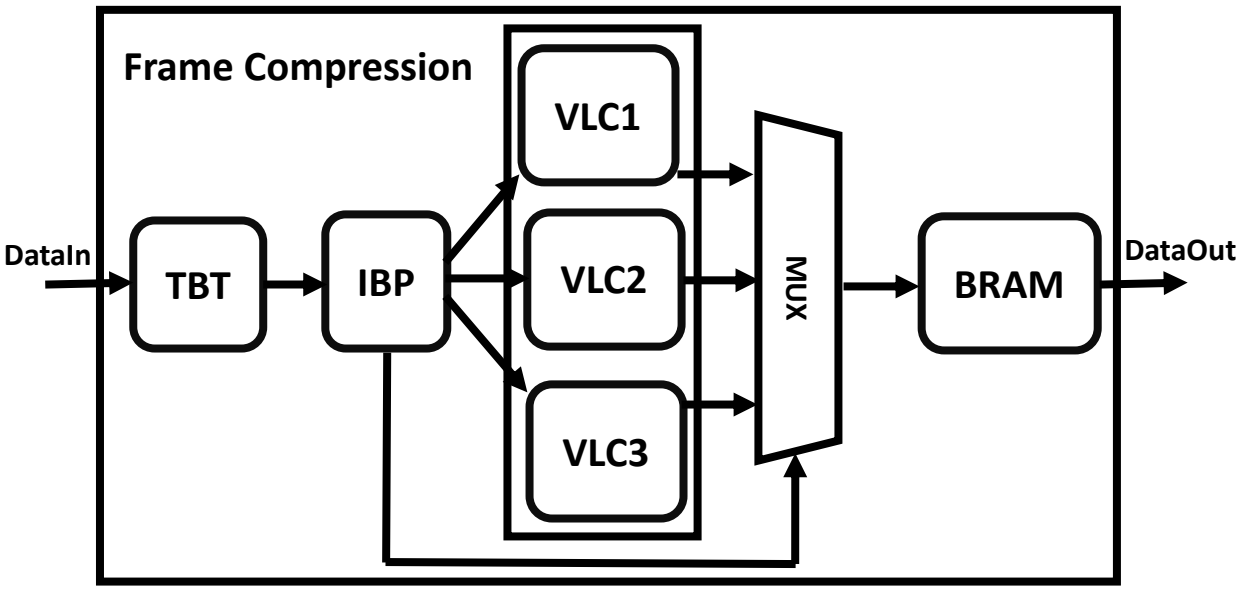
针对编码的问题，在硬件实现过程中遇到了一个难题，如何将不定长的编码结果，存进接口为定长的存储器，以 16bit 为例，针对这个问题，初始尝试了两种方法：a. 用 count 来记录当前在 DataOut 这个变量中存入的数据长度，一旦 count 大于 16，就将 DataOut 中的这部分数据存入内存，并将剩余数据右移 16bit 位，这个方法理论可行，但是在 verilog 的语法设计中，wire 和 register 这样的变量都不能够放在坐标框内，坐标框内必须是常量；b. 用 count 来记录当前 DataOut 中已经放置的数据长度，每读入一个数据，当即可以算出对应的编码 k 的占位长度 n，于是将 DataOut <= (DataOut<<n)+k，count <= count+n，这样当所有的数据编码完毕，将 DataOut 一次性移至最左端，然后每个时钟周期输出 16bit 数据到内存，而在 verilog 语法中，将变量放置在移位符号右侧，是合法的。

按照以上设计方案，当数据输入完毕，只需要几个时钟周期，就同时完成了三张残差图的编码，而与此同时 IBP mode 也得到了结果，这样就可以直接选择需要输出的残差编码到外部存储器，存起来。



图十一 VLC 编码输出到内存的方式

三、硬件架构



图十二 整体硬件架构

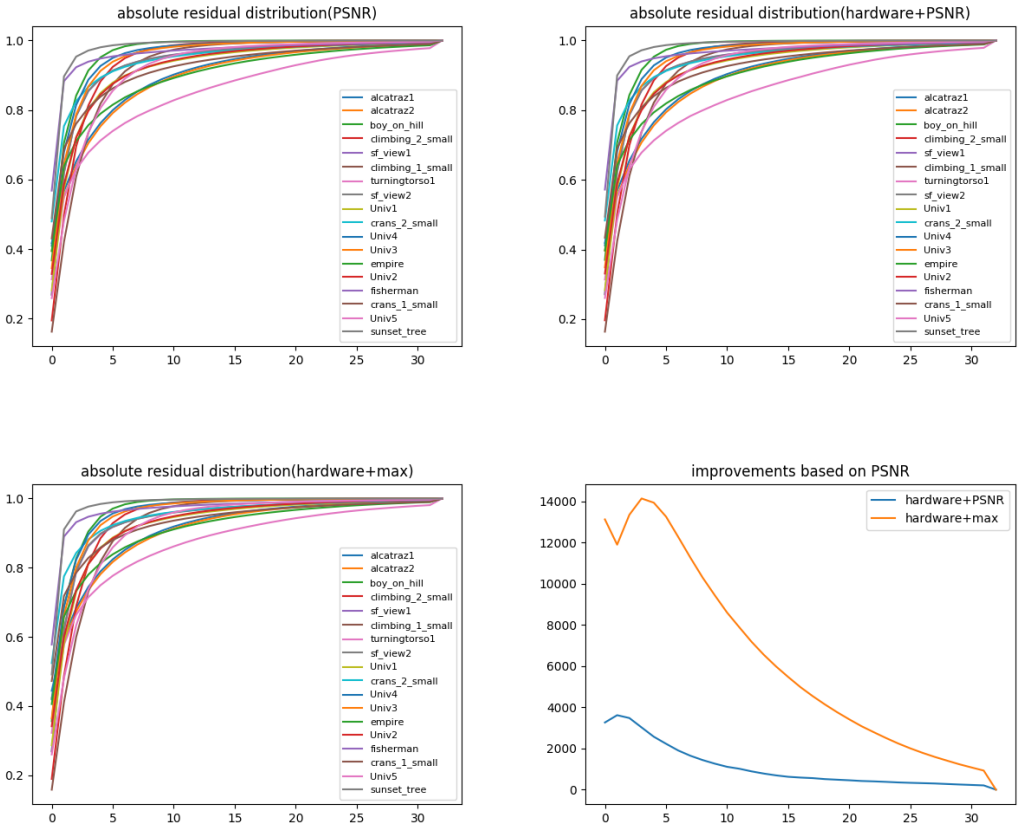
相比于文章中针对压缩部分的硬件架构，少了在 IBP 模块内对于预测值的存储，直接通过传给下一级进行编码，在 verilog 代码试验中，为了验证 verilog 硬件的功能，因此把代表外部存储的 Bram 也放在了帧压缩模块中，实际完成压缩算法的模块有 TBT、IBP、VLC1、VLC2、VLC3 和 MUX，IBP 传出的三种残差矩阵，可以直接在 Top 层进行判断，在数据输入完成之后，MUX 的选择信号立即得到，IBP mode 也可以立即得到。这样在降低硬件开销的情况下，还降低了帧压缩的时钟长度。

四、算法验证

1. 对于链式处理 r4 和 mode decision 优化算法的验证

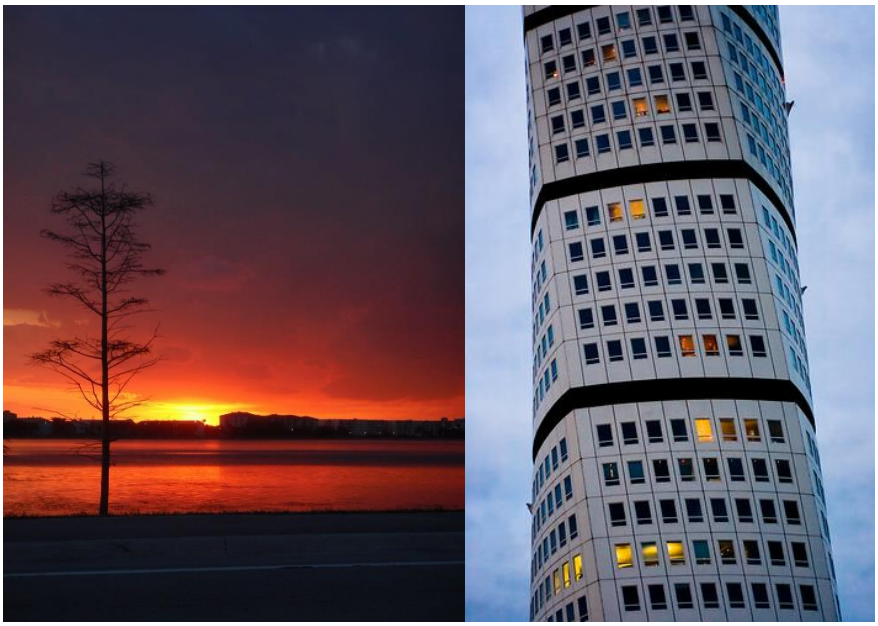
从网上随机下载了 18 张图片，进行压缩实验，并将残差记录下来，如图十三所示，模拟硬件架构的移位寄存器办法处理 r4，以及取残差最大绝对值最小的预测图作为 mode decision 方案(hardware+max)，相比于 PSNR 和 hardware+PSNR 得到的分布曲线更加尖锐，对应第四张图，以 PSNR 为比较基础，hardware+PSNR 在 0 和 1 的数值分

布明显增多，hardware+max 的方式对于性能的提升是最大的，相应的分布曲线也更加的尖锐。



图十三 算法验证

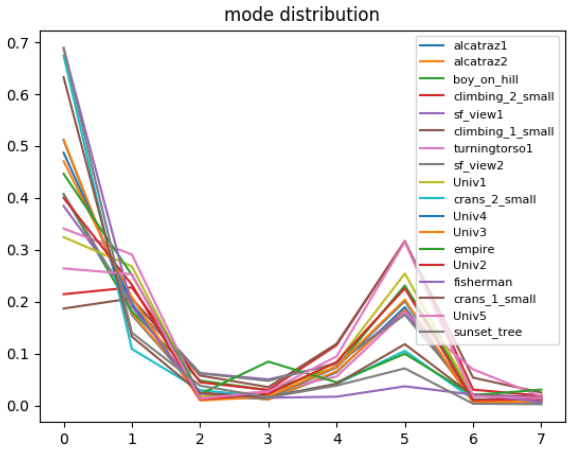
将图中曲线位置最高的灰色线和位置最低的粉色线相应的图片从数据集中拿出，进行对比，具体如图十四所示。发现灰色线对应的图片，大量的区域是渐变色的天空，颜色饱和的湖泊，以及曝光度低的公路，这样的构图直接带来的影响是，大片大片的空间颜色变化不大，对应空间裁剪出来的 8*8 像素块内，像素值的差别很小，因此 IBP mode 的预测结果与真实情况越接近，对应的残差矩阵绝对值分布也就更趋于分布在 0 和 1。粉色线条对应的图片，左右侧的天空，中间的办公楼，而每种色彩对比度都十分强烈，窗户与墙体，墙体与天空，窗户周围的黑条，楼上的黑条与浅色墙体产生强烈的色彩对比，在每一块内的残差矩阵，绝对值分布都更多分布在 0 和 1 部分，但是一旦 8*8 像素块出现跨边界的情况，尤其是出现浅色和黑色的对比，预测值与实际值的差值都会变得十分的大，高密度的窗格以及窗户旁边的黑条，都让这种处于边界的像素块，出现的比例更大，因此整体的残差绝对值分布中，相对于灰色线条线条在图中位置更低。



图十四 左图对应灰色线，右图对应粉色线

2. IBP mode 的使用频率

在八种 IBP mode 中做选择，用 18 张图片来做测试，记录每种 IBP mode 成为最优选择的频率，并绘成图片，具体见图十五。在图中可以看出，与预测的情况一致，IBP mode 6 和 IBP mode 7 的使用频率很低，使用频率最高的分别是 IBP mode 0、IBP mode 1 和 IBP mode 5，这说明相对简单的预测方案，就能够得到很好的效果。相应的线条对应的图像如图十六和图十七所示。

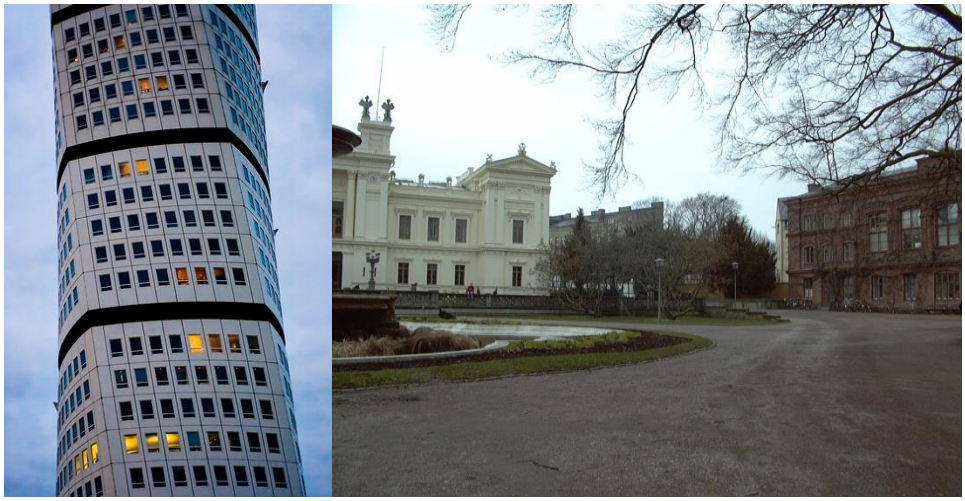


图十五 最优 IBP mode 的使用频率分布

图十六中左右图分别对应在 IBP mode 0 和 IBP mode 1 分布频率最高的浅蓝色和紫色线条，左图大量的空间对应天空、楼宇、山脉、树林，每一块的像素点变化都不大，楼宇部分虽然棱角更多，但是也存在大量墙体、窗户颜色基本一致，右图大量黑

色低曝光度的背景，以及中心部分橘黄和浅黄的商标，这样的两张图因为存在大量色块，在色块内部基本没有太大颜色变化，对应的 IBP mode 0 和 IBP mode 1 就可以很好的完成预测工作。

图十六 浅蓝和紫色线条对应的图片，最优 IBP mode 分布在 IBP mode 0 和 IBP mode 1



图十七 粉色线条和大红色线条，最优 IBP mode 大量分布在 IBP mode 5

在图十七中，左图存在大量高色彩对比度的部分，窗户和墙体，墙体和天空，墙体黑条和浅色墙体，这些都会造成局部像素点波动很大，右图因为枝丫的黑色和天空的白色造成巨大的像素点差异，同样带来局部像素点波动大的问题，在这种情况下 (r1+r4)>>1 的预测效果，因为综合了 r1 和 r4 的值，相对于其他预测方案会更好，可以看出的是 IBP mode 4、IBP mode 6 和 IBP mode 7 在这两张图中的出现比例也更高，但是 IBP mode 5 的效果相对更好。

从以上可以看出，IBP mode 0 和 IBP mode 1 针对的是存在大块像素变化不大的区域的图片，IBP mode 4、IBP mode 5、IBP mode 6 和 IBP mode 7 主要是针对在一定区域内，想去对比度较大的图片，这样的经验可以帮我们去选择 intra mode，DC、Planner 主要是针对色彩对比度大的图片，2-10、11-18、19-26 和 27-34 都考虑同时照顾一张图片内有大量单一色块和对比度较大的区域，2-10、11-18 更趋向于在横向色彩几乎没

变吧，变化梯度方向处于纵向的图片，例如夕阳西下的那张图，19-26 和 17-34 更趋向于处理色彩变化在横向发生梯度变化，而在纵向几乎没有什么变化，这样的图片相对比例更小，因为一般来说，天空和大地楼宇产生的色彩对比强度是要高于横向色彩对比的。在我们的数据集中相对比较单一，大量都是在纵向存在更大色彩对比度，因此 IBP mode 0 比 IBP mode 1 的分布频率要高的多。

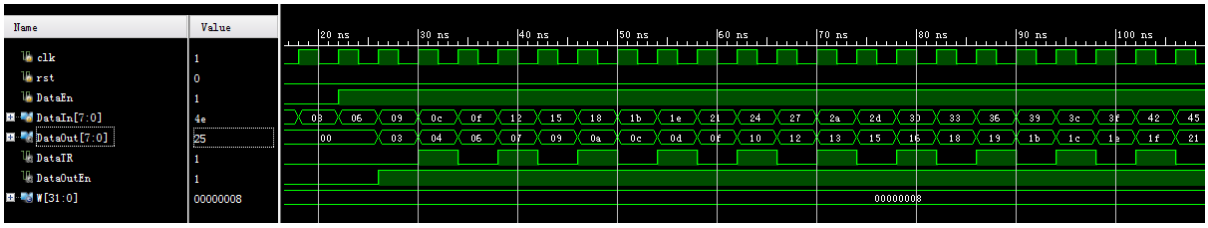
3. 将图片进行解压缩验证算法

通过编程，将初始图片进行压缩后再解压缩，对比原图与解压后的图片，来判断压缩算法的正误，具体如图十八所示，发现几乎没有什么变化，因此算法验证正确可行。

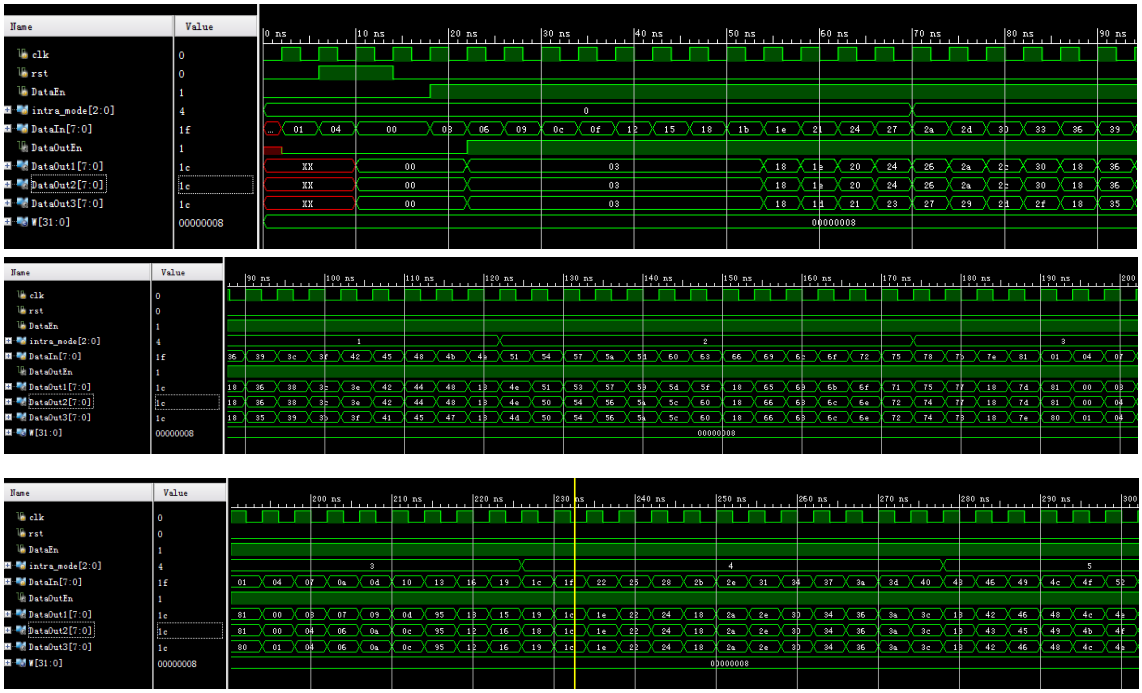
图十八　原图与解压缩后的图片

五、硬件仿真结果

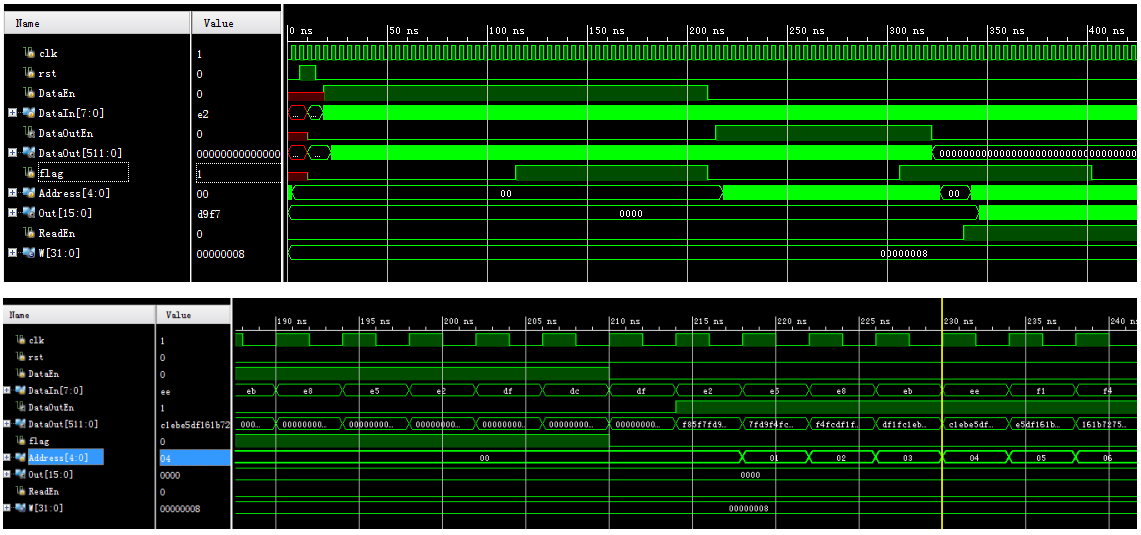
1. TBT 模块



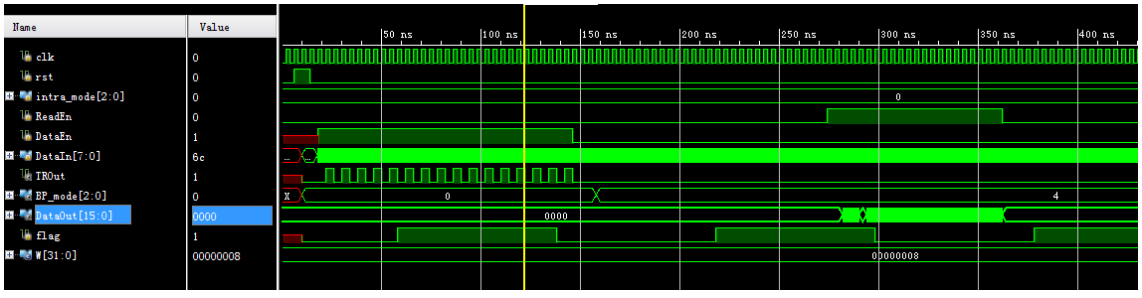
2. IBP 模块

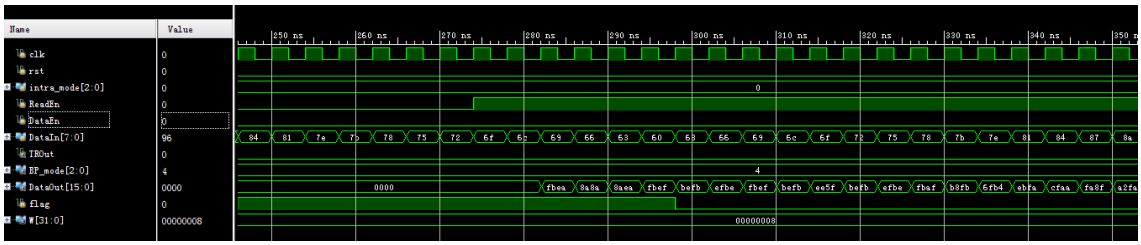


3. VLC 模块



4. Top 模块





5. 资源消耗

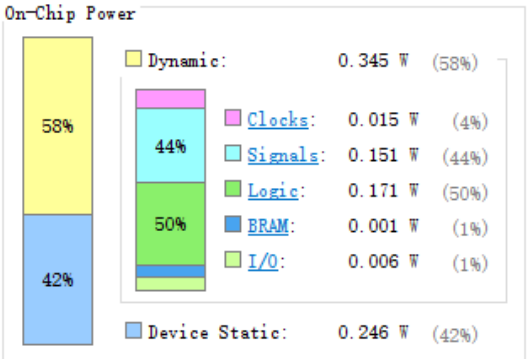
Utilization – Post-Synthesis

Resource	Estimation	Available	Utilization %
LUT	20025	303600	6.60
LUTRAM	7	130800	0.01
FF	1673	607200	0.28
IO	35	600	5.83
BUFG	1	32	3.13

6. 功耗

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.59 W**
Junction Temperature:	**25.8 °C**
Thermal Margin:	59.2 °C (40.8 W)
Effective ϑJA:	1.4 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



7. 时钟

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.029 ns	Worst Hold Slack (WHS):	-0.004 ns	Worst Pulse Width Slack (WPWS):	7.720 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-0.030 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	7	Number of Failing Endpoints:	0
Total Number of Endpoints:	3395	Total Number of Endpoints:	3395	Total Number of Endpoints:	1683

Timing constraints are not met.

引用：

[1]. Y. Fan, Q. Shang and X. Zeng, "In-Block Prediction-Based Mixed Lossy and Lossless Reference Frame Recompression for Next-Generation Video Encoding," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 1, pp. 112-124, Jan. 2015.