Frame Compression

16110720018 张姝菡

Contents

- Algorithm and hardware design

→ Overall Architecture

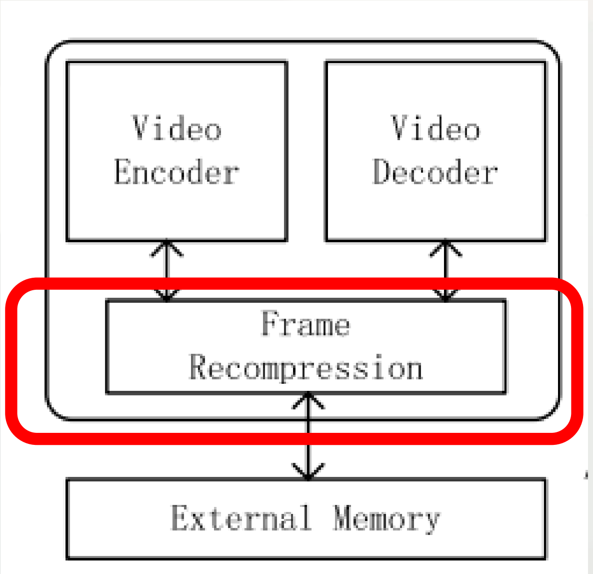
→ TBT(Tail Bit Truncation)

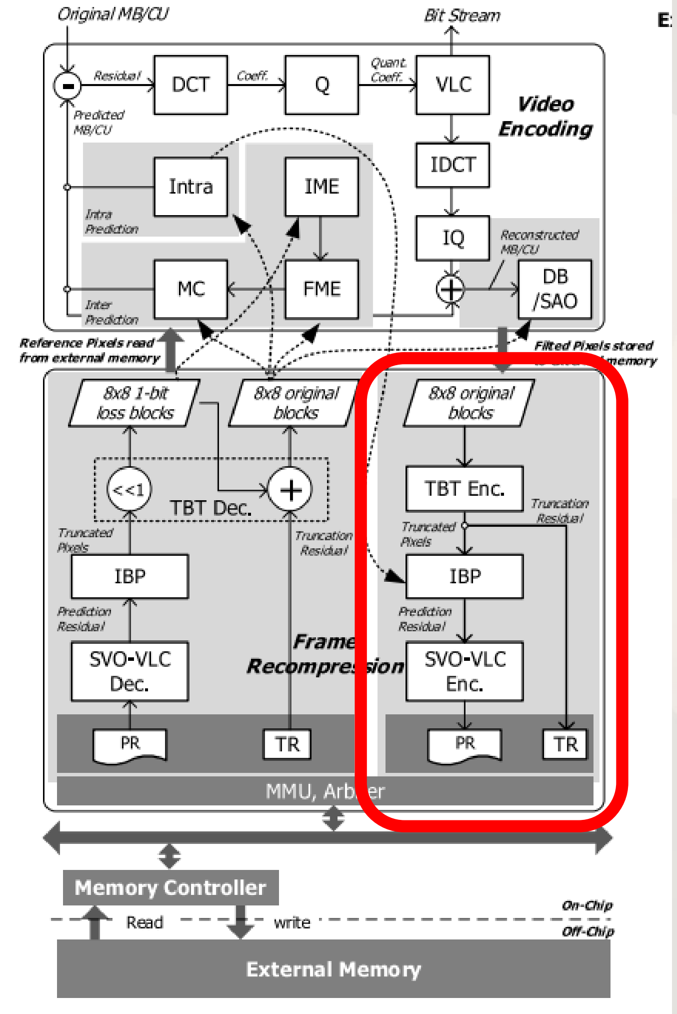
→ IBP(In Block Prediction)

→ VLC(Variable Length Coding)

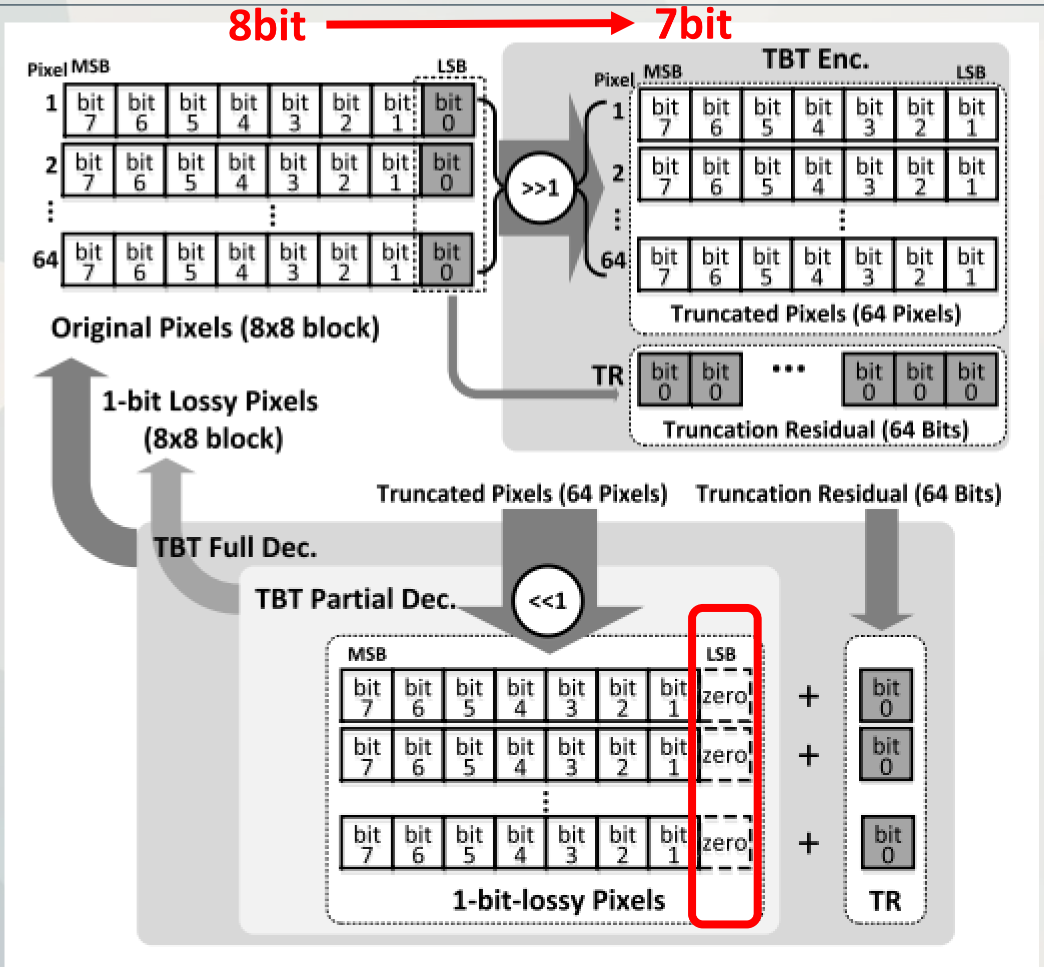
- Hardware architecture
- Experiment results

Overall Architecture





TBT

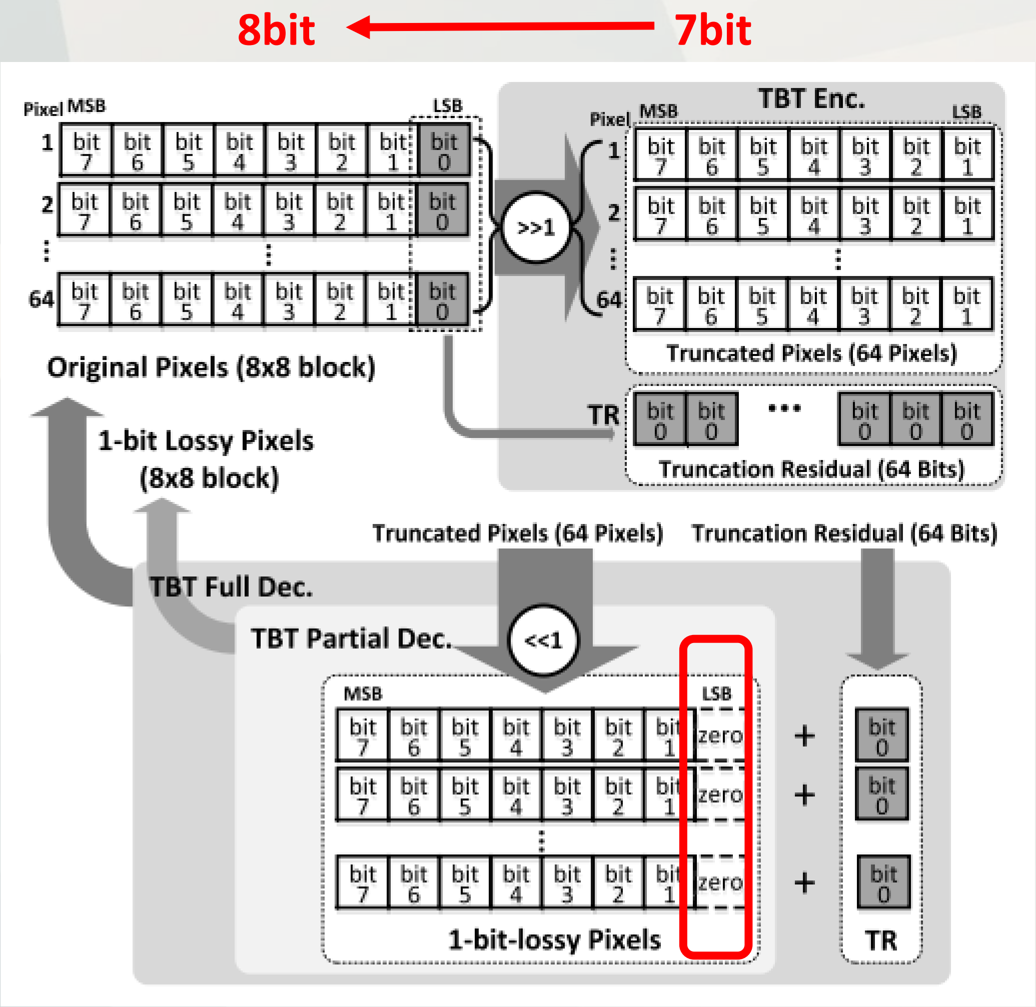



```python
import numpy as np

def TBT(X,l):
    B = 2**l - 1
    n, m = X.shape

    # residual
    TR = np.zeros(n,m)
    for i in range(n):
        for j in range(m):
            TR[i,j] = X[i,j] & B
    # new X after truncation
    for i in range(n):
        for j in range(m):
            X[i,j] = X[i,j] >> l
    return X, TR
```

TBT decompression

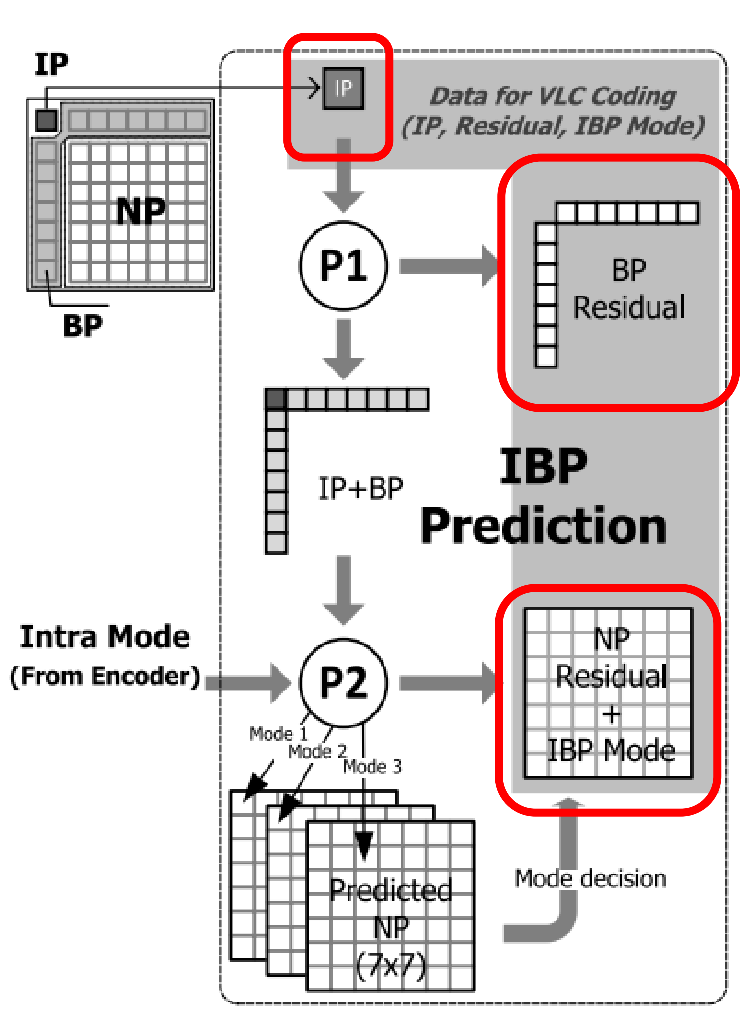


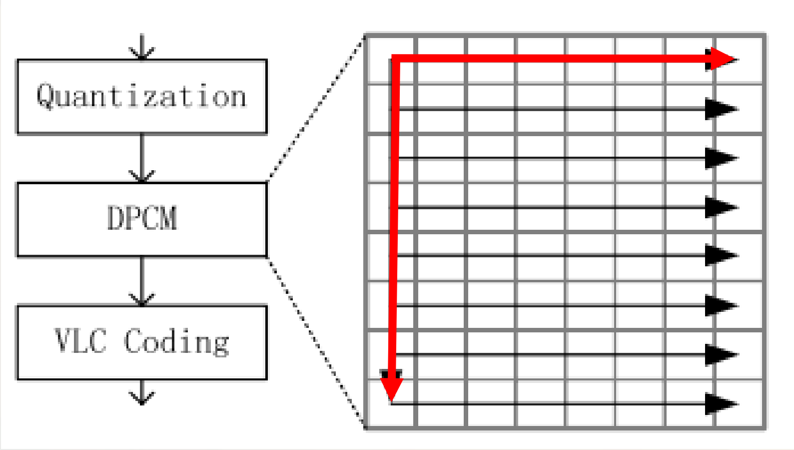

```python
import numpy as np
from math import log

def TBT_decode(X, TR, l):
    n,m = X.shape
    # initialize output frame
    new_X = np.array(X.shape,dtype=np.uint8)

    # decode
    for i in range(n):
        for j in range(m):
            new_X[i,j] = (X[i,j] << l) + TR[i,j]
    return new_X
```

IBP





BP residual

```
# BP residual
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]
```

IBP

After 1bit TBT, the BP mode computation won't overflow.

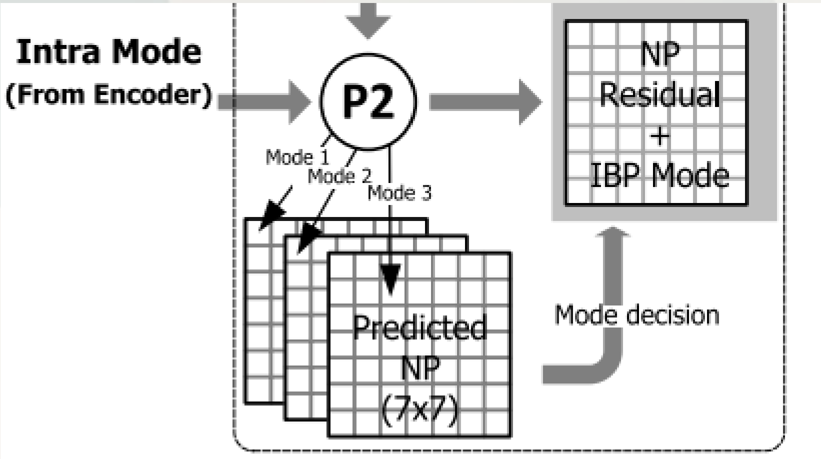


TABLE I
IBP MODE

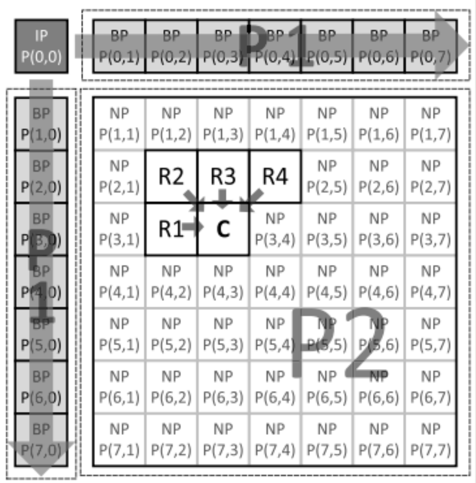
Mode	Prediction (C=)
0	R1
1	R3
2	(R1+R2)/2
3	(R3+R4)/2
4	(R1+R4)/2
5	(R1+R3)/2
6	((R1+R2)/2+R3)/2
7	((R1+R2)/2+(R3+R4)/2)/2

```python
import numpy as np
from math import log

# BP mode
mode0 = lambda r1,r2,r3,r4: r1
mode1 = lambda r1,r2,r3,r4: r3
mode2 = lambda r1,r2,r3,r4: (r1+r2)>>1
mode3 = lambda r1,r2,r3,r4: (r3+r4)>>1
mode4 = lambda r1,r2,r3,r4: (r1+r4)>>1
mode5 = lambda r1,r2,r3,r4: (r1+r3)>>1
mode6 = lambda r1,r2,r3,r4: (((r1+r2)>>1)+r3)>>1
mode7 = lambda r1,r2,r3,r4: (((r1+r2)>>1)+((r3+r4)>>1))>>1
ibp_mode = [mode0,mode1,mode2,mode3,mode4,mode5,mode6,mode7]
domain = [[4,5,7],[4,5,7],[0,4,5],[0,2,6],[1,5,6],[1,3,4]]
```

TABLE II
MODE MAPPING FOR HEVC INTRA MODE AND IBP MODE

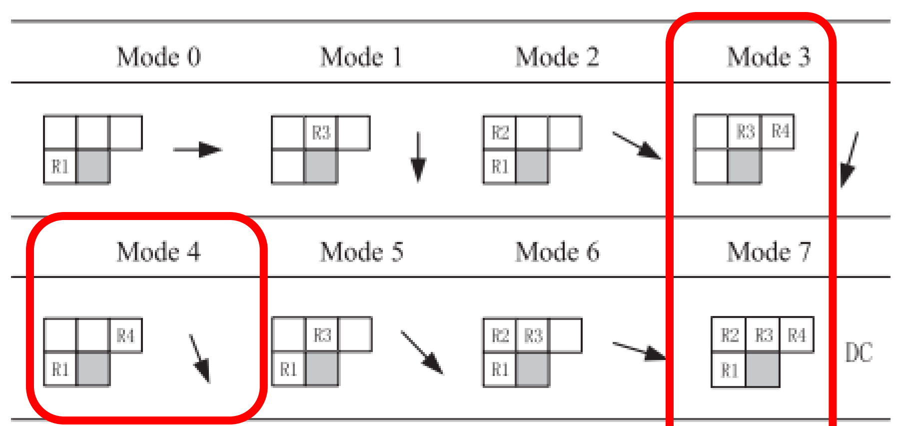
Intra Mode	IBP Mode	Intra Mode	IBP Mode
Planner	4,5,7	11-18	0,2,6
DC	4,5,7	19-26	1,5,6
2-10	0,4,5	27-34	1,3,4



C: Predicted Pixel Rx: Reference Pixels

```python
# 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)
```

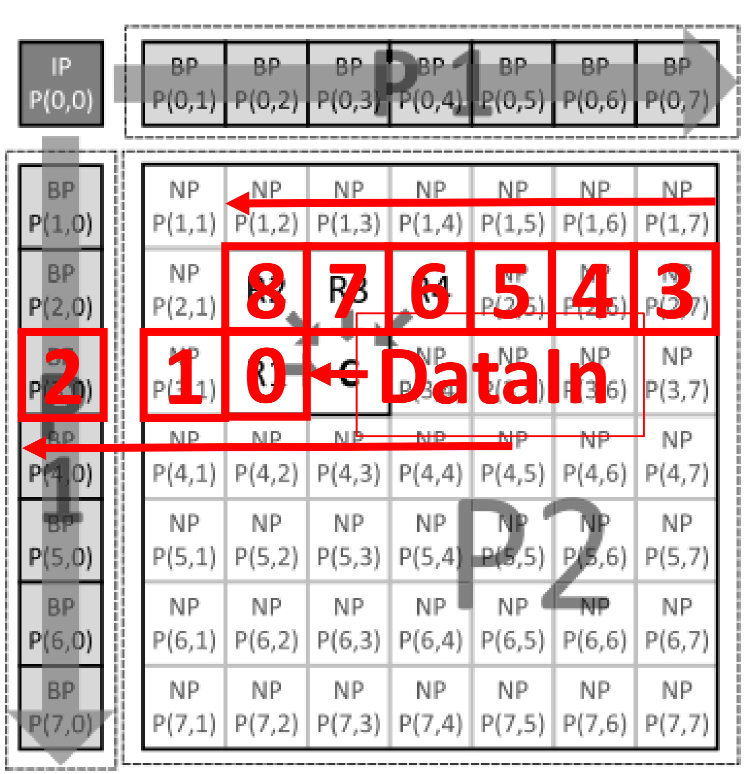
IBP



IBP modes selection

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



C: Predicted Pixel   Rx: Reference Pixels

```
//wire
wire [W-1:0] Out1, Out2, Out3, Out4;

assign Out1 = DataEn ? Mem[0] : 0;
assign Out2 = DataEn ? Mem[Length-1] : 0;
assign Out3 = DataEn ? Mem[Length-2] : 0;
assign Out4 = DataEn ? Mem[Length-3] : 0;
```

BP mode decision

In order to reduce the memory consumption after VLC, we need to make the maximum absolute value of the residual block as small as possible.

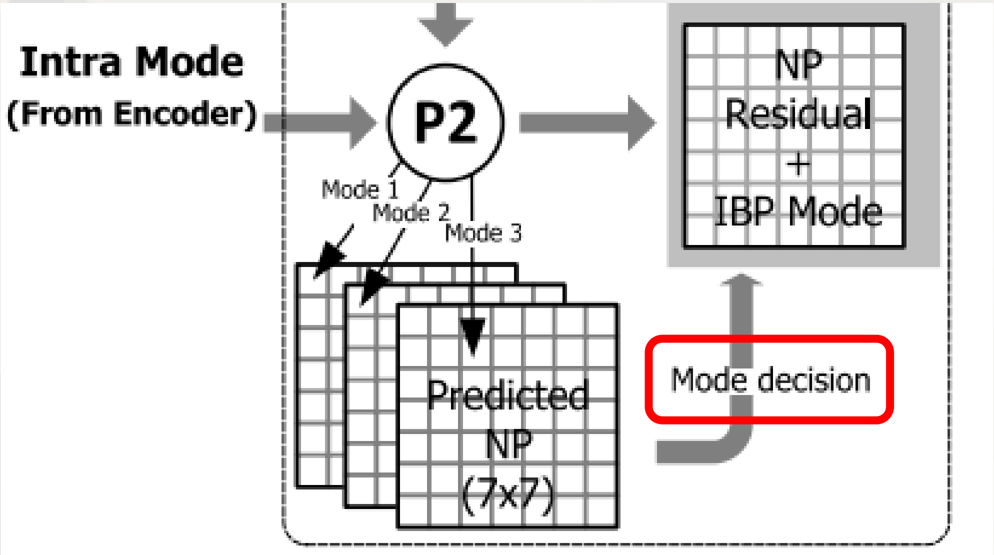
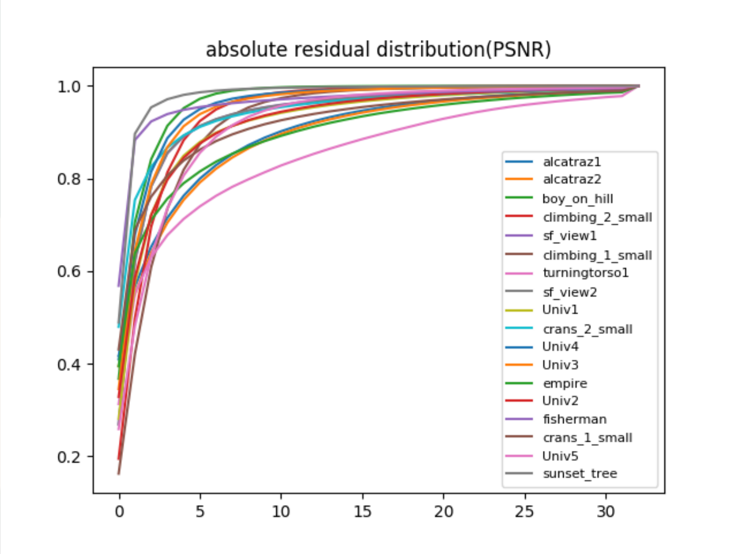
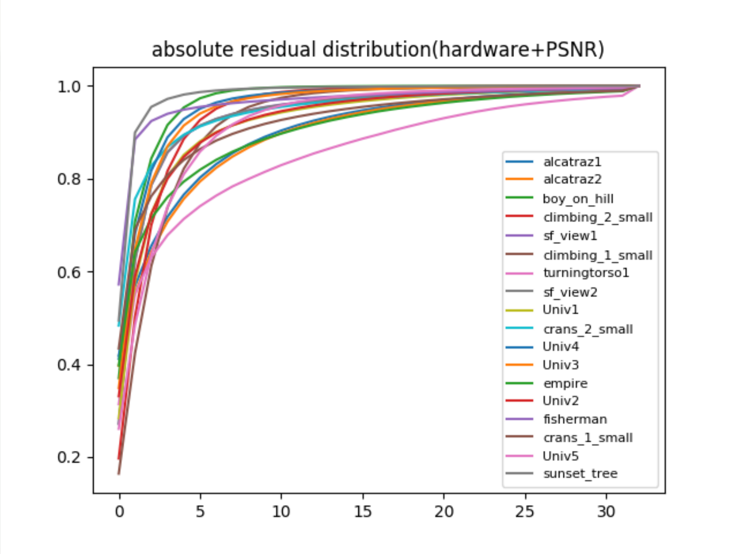


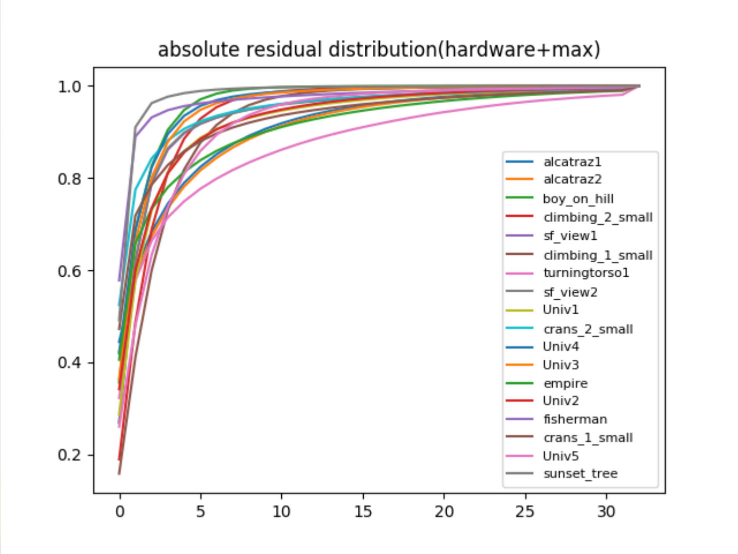
TABLE III
PROPOSED VLC TABLE

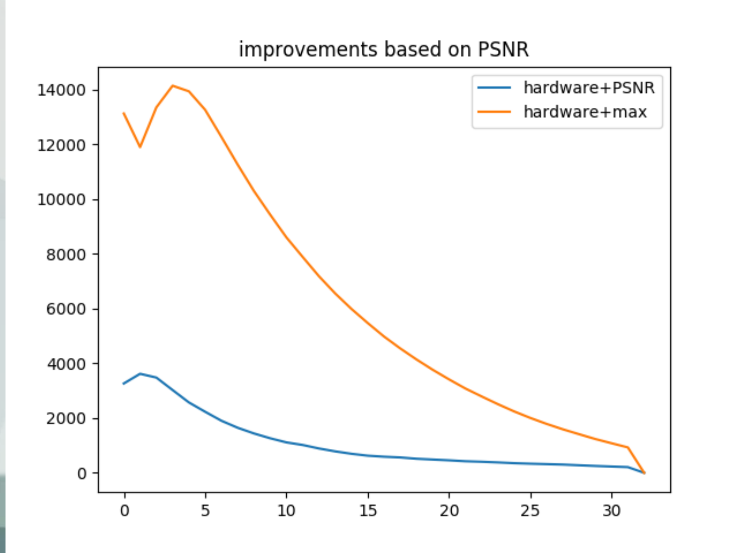
	01	**10**	**00**	**110**
Max. value	*0*	*1*	*2*	*3~4*
0	-	1	01	001
±1		0S	1S	01S
±2			00S	10S
±3				11S
±4				000S
	1110	**11110**	**111110**	**111111**
Max. value	*5~8*	*9~16*	*17~32*	*>32*
0	0001	00001	00001	
±1	001S	0001S	0001S	
±2	010S	0010S	0010S	
…	…	…	…	
±7	111S	0111S	0111S	
±8	0000S	1000S	1000S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

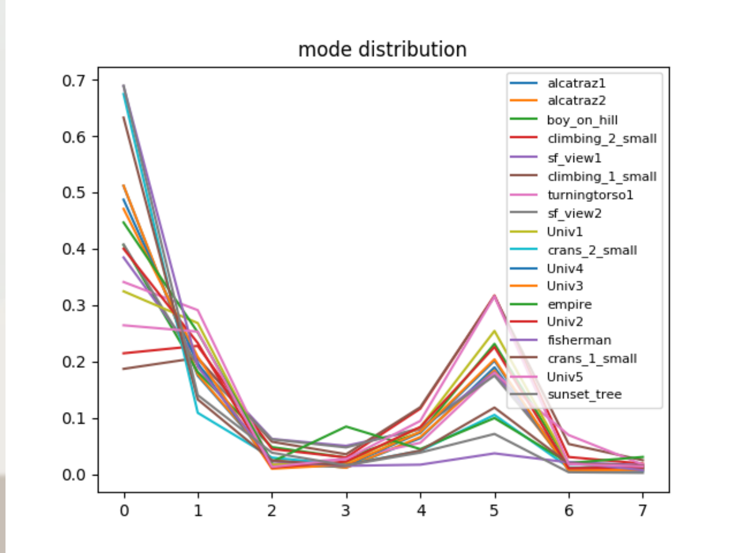
IBP experiments



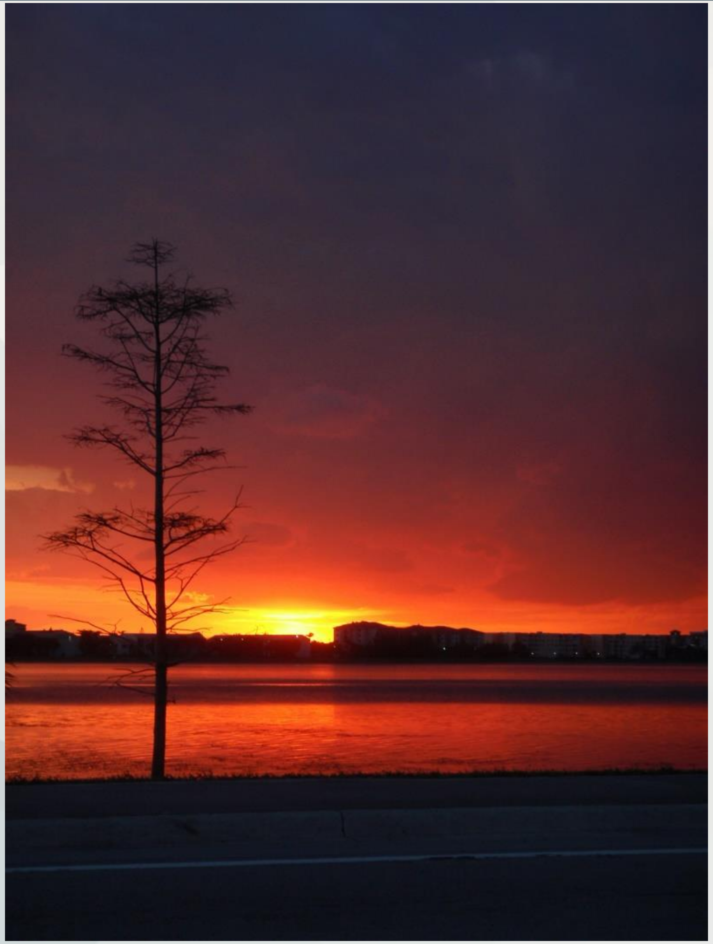








IBP experiments



gray line

pink line

IBP code

```python
from recompression import TBT
from pylab import *
from PIL import Image

def IBP_decode(IP, BP_mode, res):
    X = zeros(res.shape, dtype=np.uint8)
    n,m = res.shape

    # IP
    X[0,0] = IP

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

    # NP
    X = X.flatten()
    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]
            X[idx] = ibp_mode[BP_mode](r1,r2,r3,r4) + res[i,j]
    X = X.reshape(res.shape)
    return X
```

```python
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
```

Results

testing code

```
from recompression import *
from PIL import Image
from pylab import *
from scipy import misc

im = array(Image.open('empire.jpg').convert('L'))
# original picture
misc.toimage(im, cmin=0, cmax=255).save('gray_empire.jpg')
n, m = im.shape

new_im = zeros([n,m])
for i in range(n/8):
    for j in range(m/8):
        X = im[i*8:(i+1)*8, j*8:(j+1)*8]
        X, TR = TBT(X, 1)
        IP, BP_mode, res = IBP(X, 0, 7)
        X = IBP_decode(IP, BP_mode, res)
        X = TBT_decode(X, TR, 1)
        new_im[i*8:(i+1)*8, j*8:(j+1)*8] = X

# image after decompression
misc.toimage(new_im, cmin=0, cmax=255).save('new_empire.jpg')
```

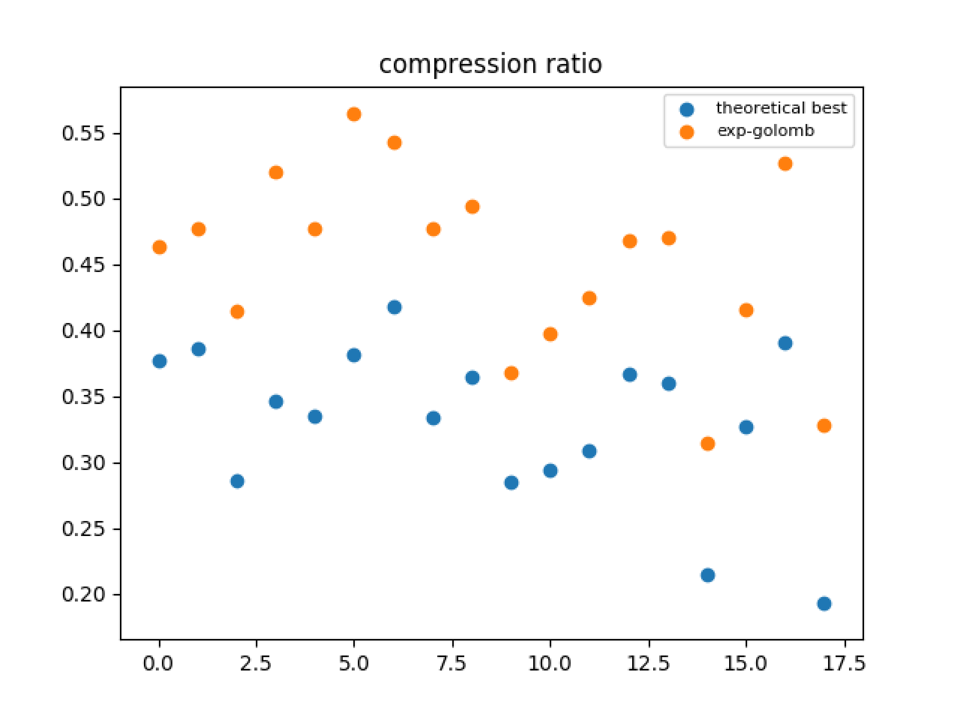
original image

image after decompression

VLC

- The proposed VLC table on the paper requires the maximum absolute residual. Thus, 28(3*8+4) clock time is required before compression.
- The division of 8*8 block and compression mode increase the control logic complexity. This can deteriorate the circuit’s performance.
- If only one large residual exists in the 4*4 block, the compression rate the whole block will be increased.
- Instead of proposed VLC, I choose to compress residuals with Exp-golomb.

Exp-golomb



DataOut | | …… ……

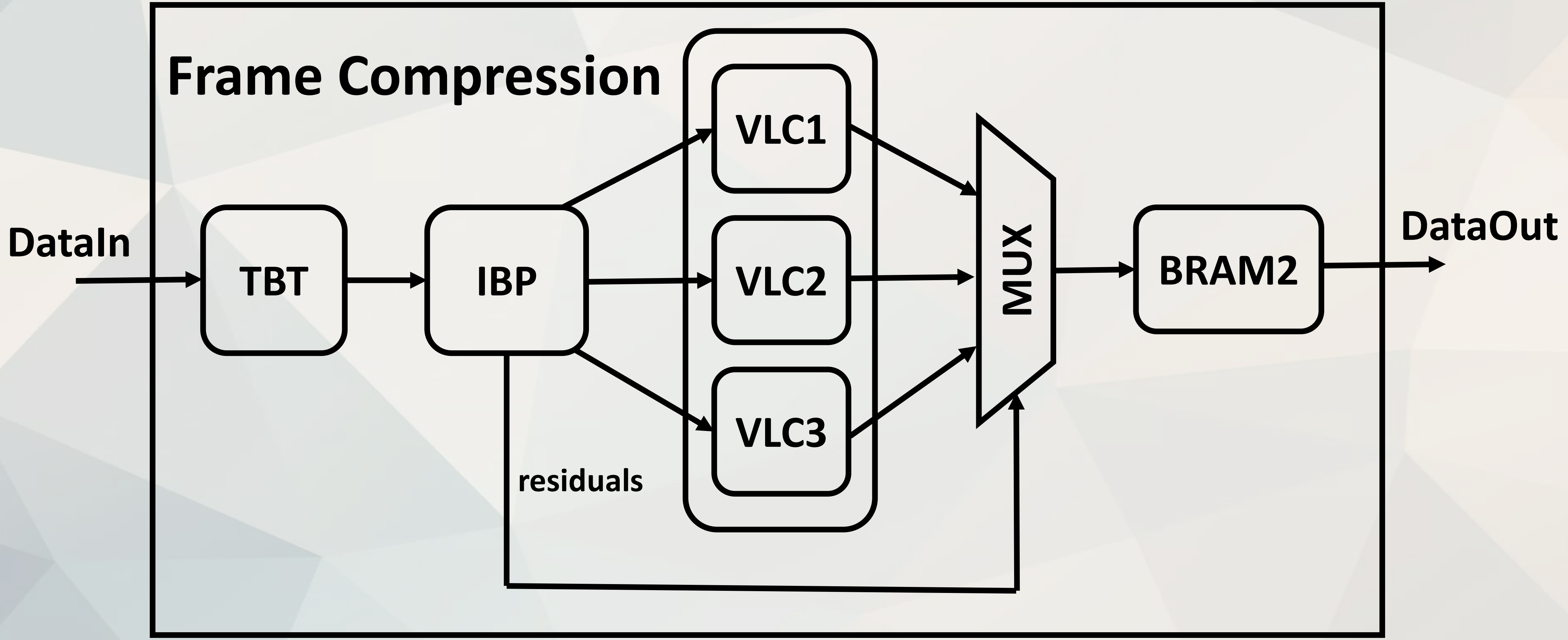
length **count**

Methods:

1) Let count to memorize the coordinate, and output DataOut if count equals or exceeds memory band width.
2) Use count to memorize the coordinate. After compression, left shift DataOut (length-count) bit. Then, sequentially output DataOut.

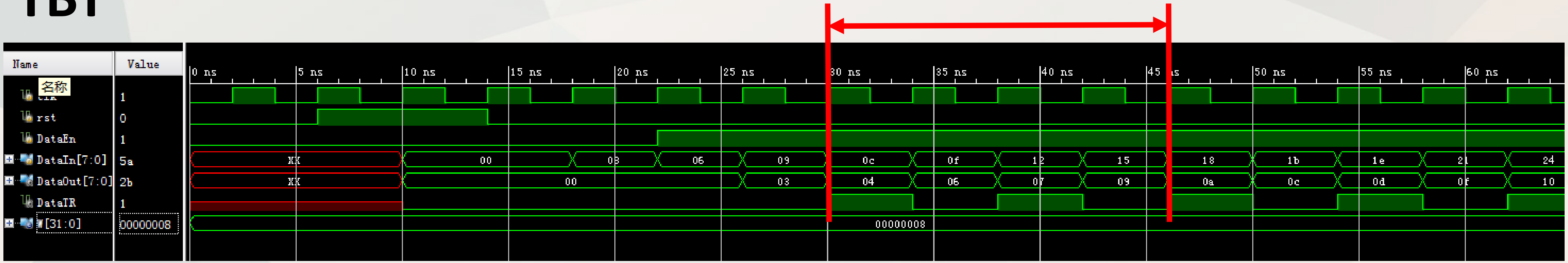
Due to the fact that we cannot use register or wire to represent coordinates, method 1 is not possible.

Hardware Architecture

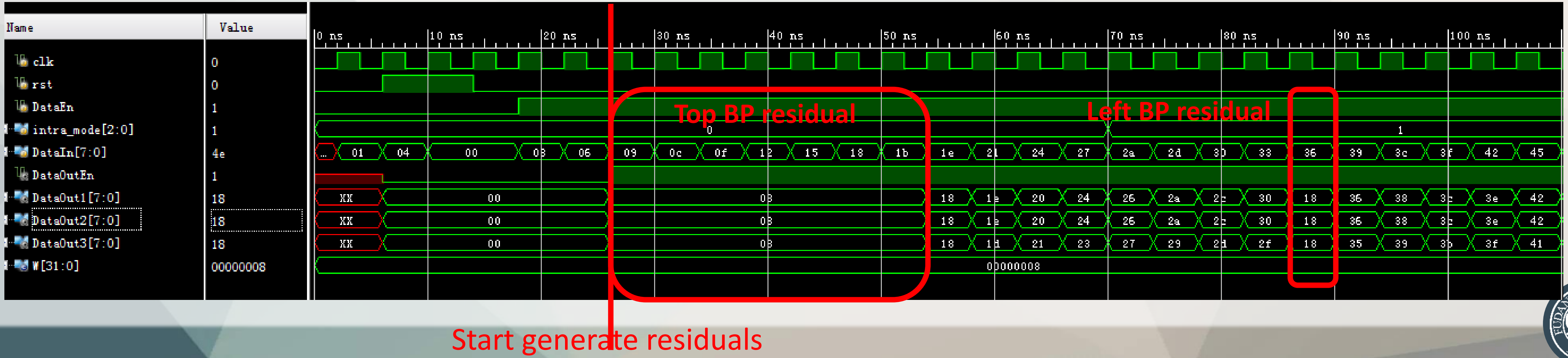


Results

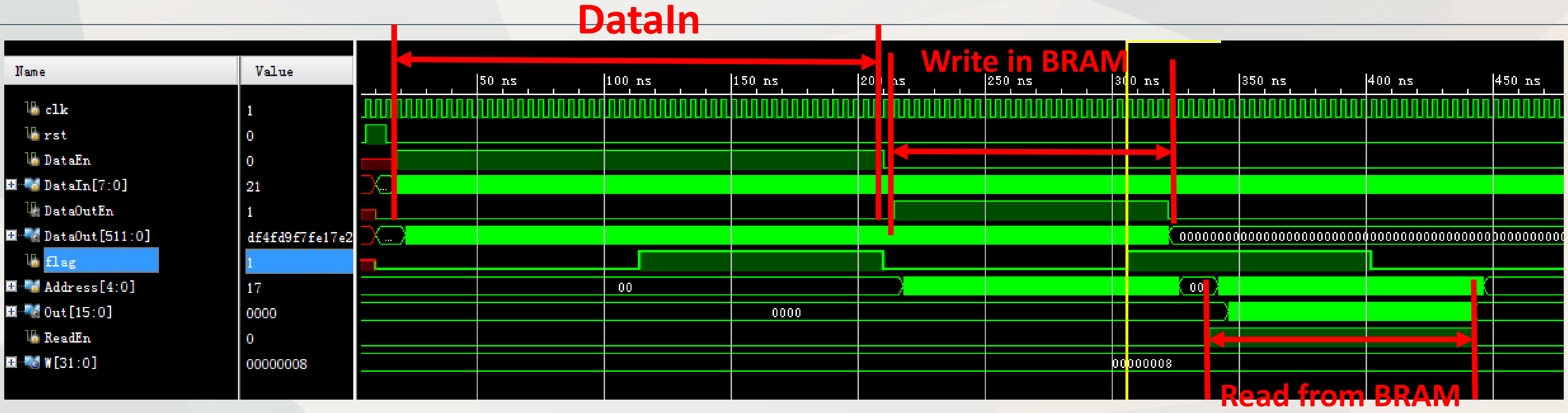
TBT

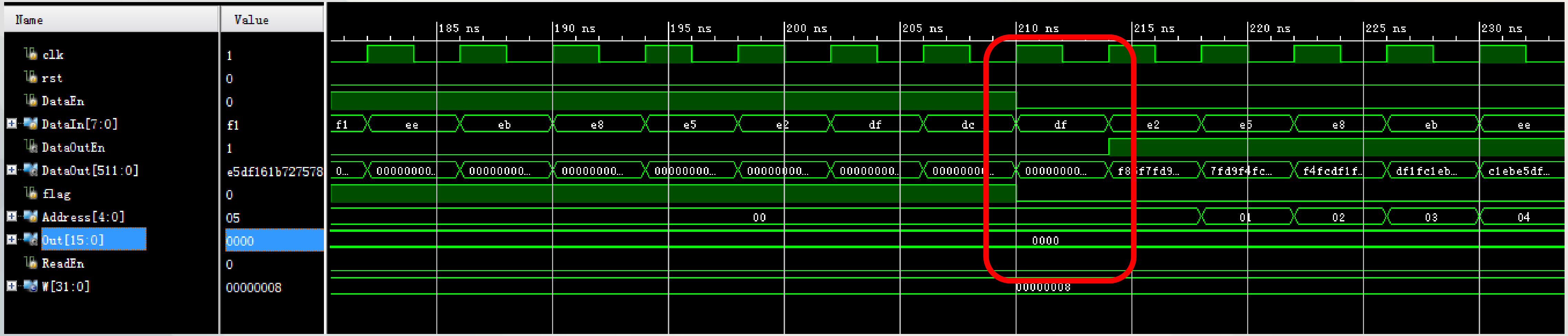


IBP

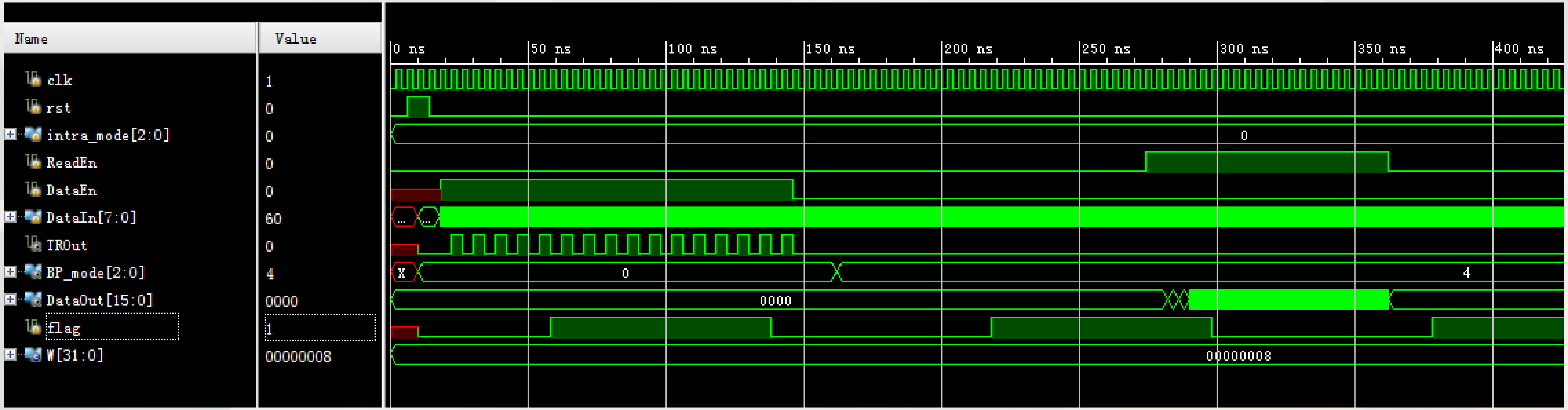


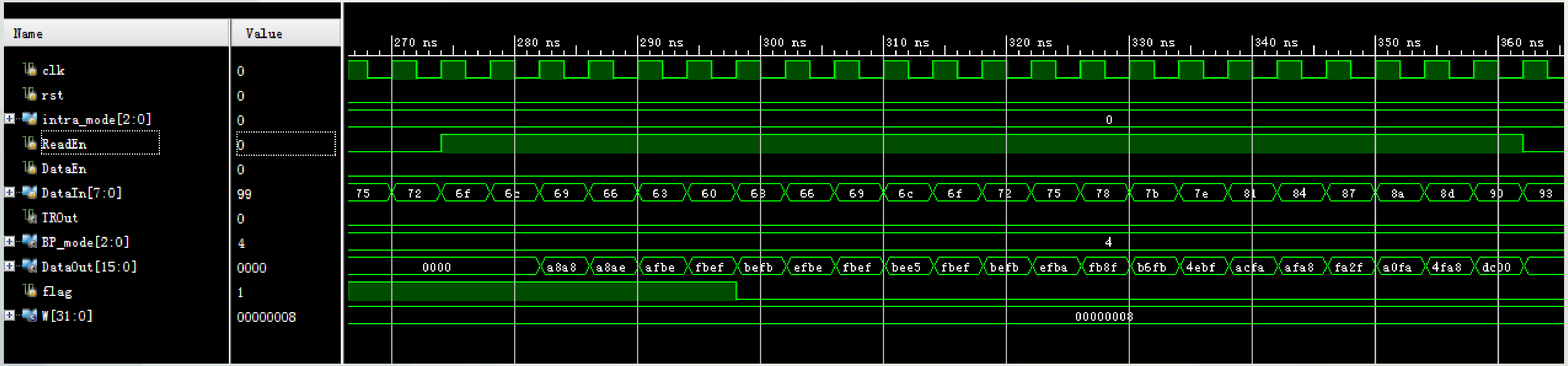
Results: VLC


DataIn
Write in BRAM
Read from BRAM
Name
Value
clk
rst
DataEn
DataIn[7:0]
DataOutEn
DataOut[511:0]
flag
Address[4:0]
Out[15:0]
ReadEn
W[31:0]
1
0
0
21
1
df4fd9f7fe17e2
1
17
0000
0
00000008
50 ns
100 ns
150 ns
250 ns
350 ns
400 ns
450 ns
00
0000
00000008


Name
Value
clk
rst
DataEn
DataIn[7:0]
DataOutEn
DataOut[511:0]
flag
Address[4:0]
Out[15:0]
ReadEn
W[31:0]
1
0
0
f1
1
e5df161b727578
0
05
0000
0
00000008
185 ns
190 ns
195 ns
200 ns
205 ns
210 ns
215 ns
220 ns
225 ns
230 ns
f1
ee
eb
e8
e5
e2
df
dc
df
e2
e5
e8
eb
ee
00
01
02
03
04
0000
00000008

Results: Top module





Results

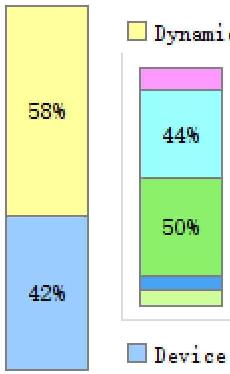
Utilization – Post-Synthesis

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

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 ℃**
Thermal Margin:	59.2 ℃ (40.8 W)
Effective ϑJA:	1.4 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power



Dynamic:	0.345 W	(58%)
Clocks:	0.015 W	(4%)
Signals:	0.151 W	(44%)
Logic:	0.171 W	(50%)
BRAM:	0.001 W	(1%)
I/O:	0.006 W	(1%)
Device Static:	0.246 W	(42%)

Setup

Worst Negative Slack (WNS):	3.029 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	3395

Hold

Worst Hold Slack (WHS):	-0.004 ns
Total Hold Slack (THS):	-0.030 ns
Number of Failing Endpoints:	7
Total Number of Endpoints:	3395

Pulse Width

Worst Pulse Width Slack (WPWS):	7.720 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	1683

Timing constraints are not met.

Thank you!