VLSI Project2

Report_Project02_김두현

1. Overview

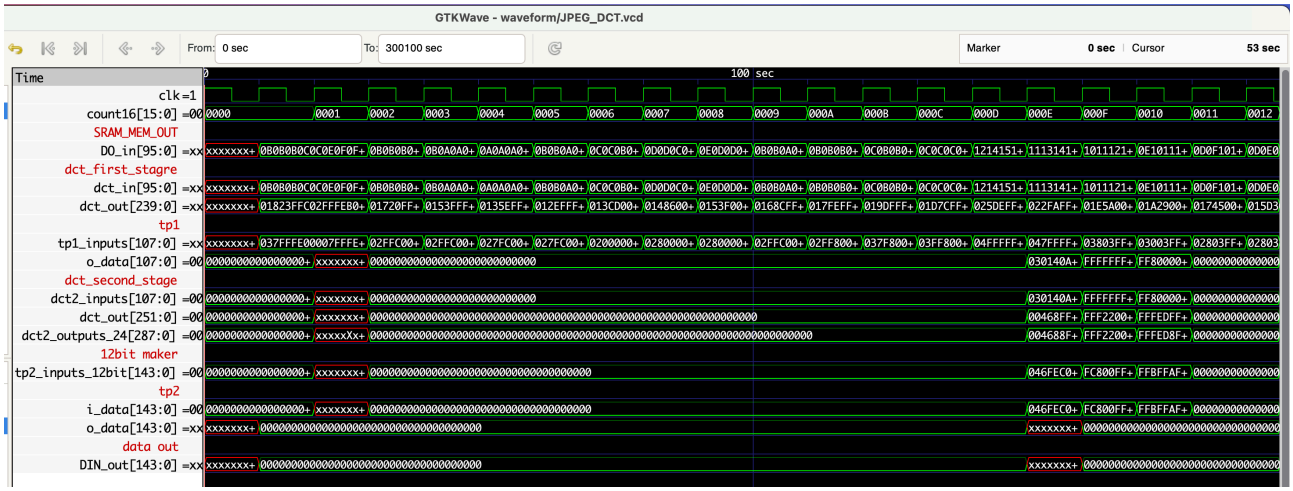
이번 실험에서 우리는 JPEG이나 동영상 압축에서 사용되는 DCT 의 forward process에 대한 모듈을 구성한다.
이때 area와 power 관점에서 회로의 효율을 최대화하는 과정을 설명한다.
이번 보고서에는 DCT의 면적을 줄이기 위해서 다양한 방법들을 시도한다.

2. Waveform

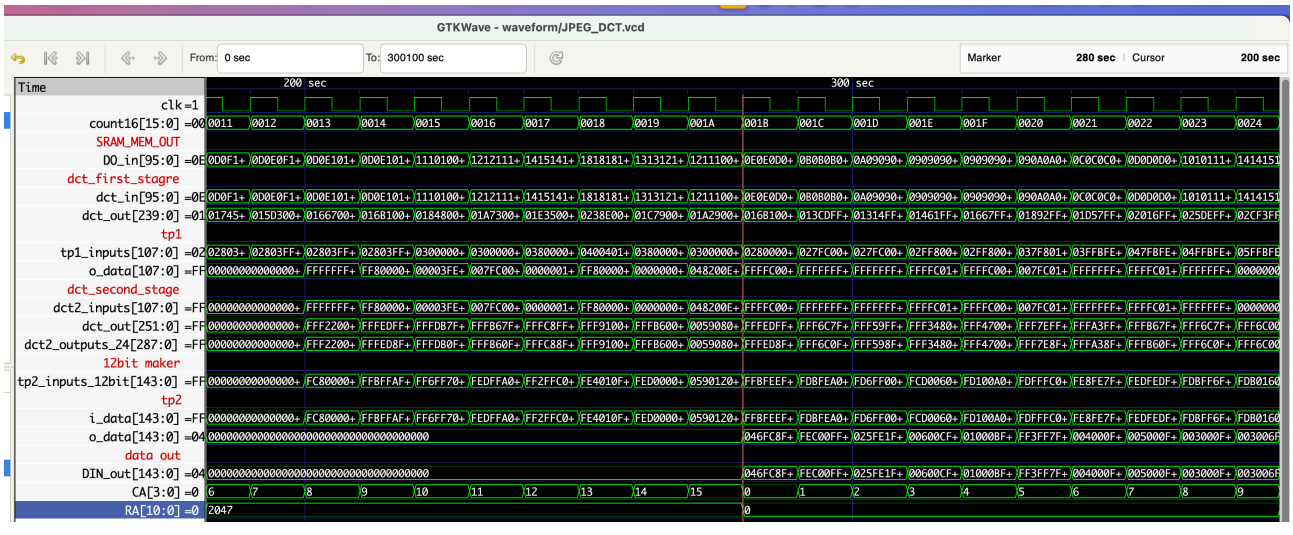
조건

1) coefficient quantization 8bit
2) Middle BW = 9bit

DFF 없는 최소 면적 wave

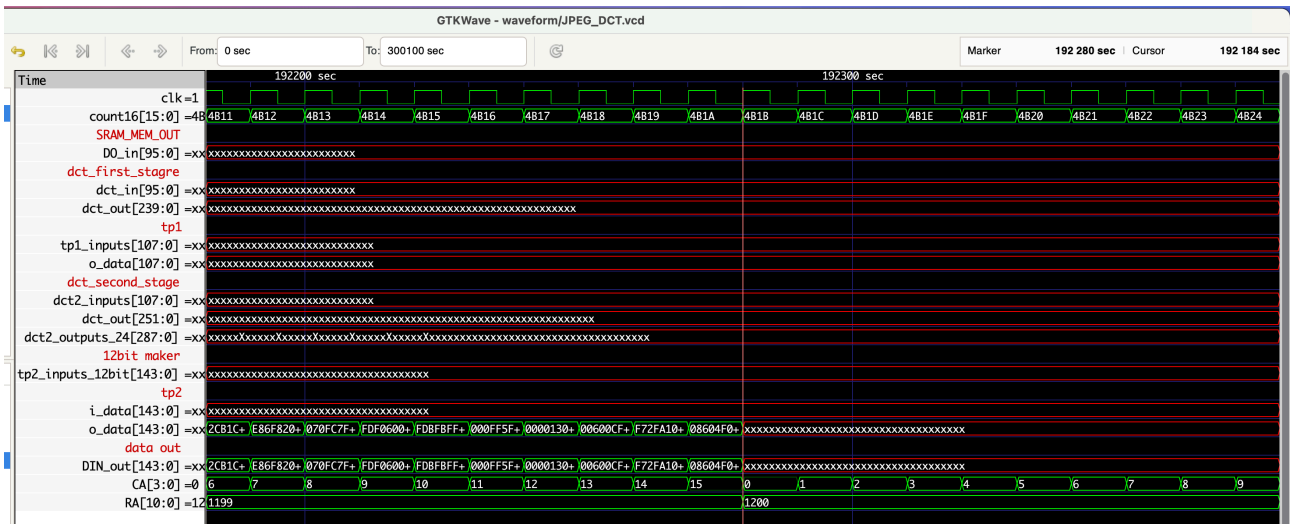


real data out point

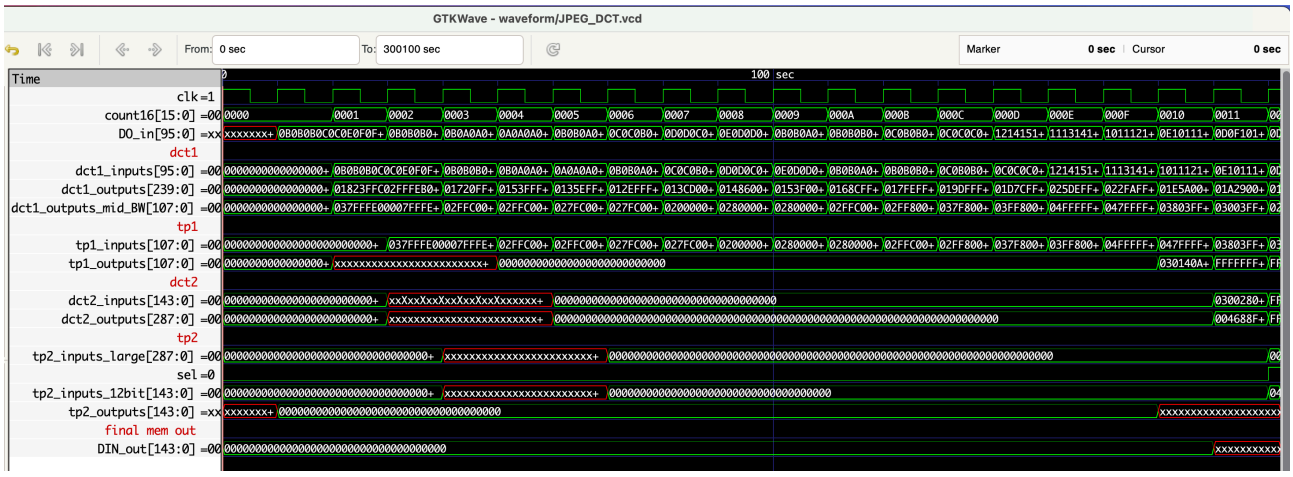


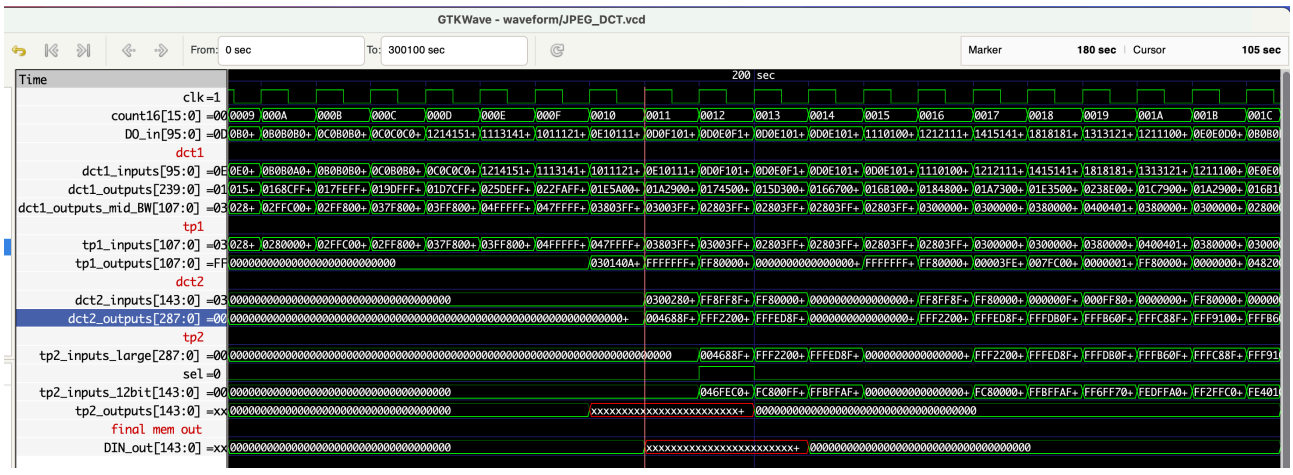
Last output Memory write

VLSI Project2

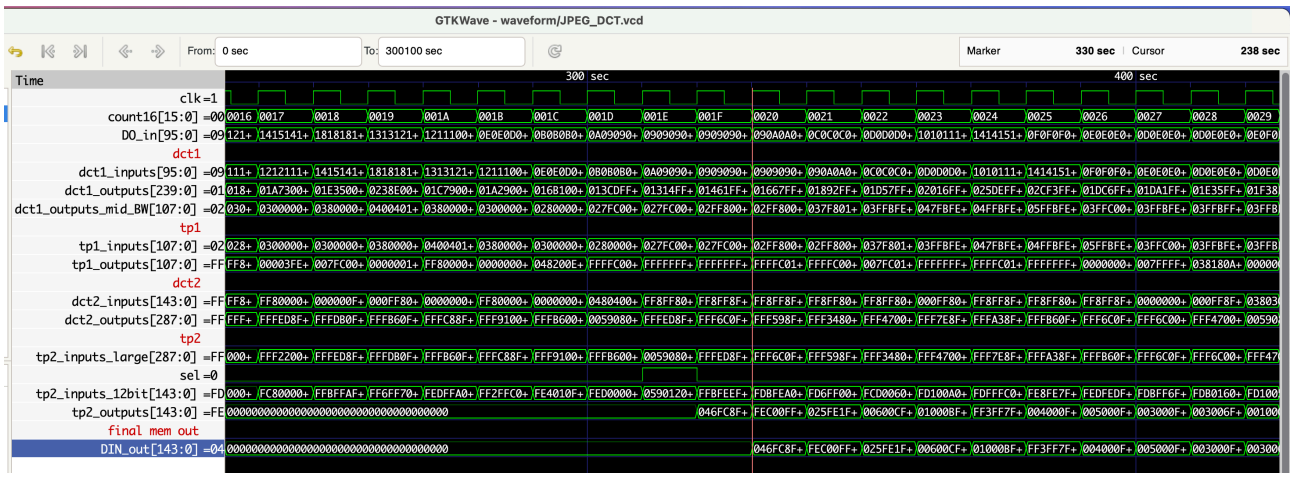


DFF 있는 version





VLSI Project2



3. Synthesis

(최종)

Area

Project2 > thanh > txt > report_area.txt

```
design_vision-xg-t> report_area

****************************************
Report : area
Design : TOP_JPEG_2D_DCT_synthesis
Version: Z-2007.03-SP4
Date   : Fri Jun 14 14:24:41 2024
****************************************

Library(s) Used:

    lec25dscc25_SS (File: /home/admin/lib/lec25/lec25dscc25_SS.db)

Number of ports:              242
Number of nets:              1648
Number of cells:               19
Number of references:          17

Combinational area:       709868.033863
Noncombinational area:    688146.185760
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          1398014.250000
Total area:                 undefined
1
```

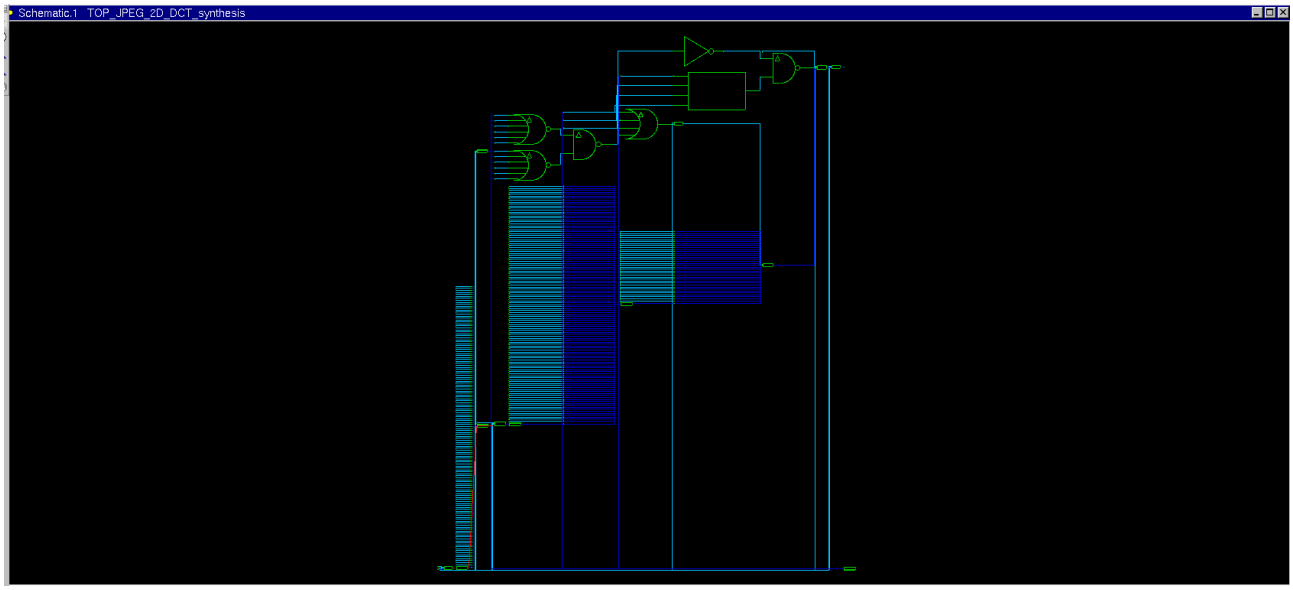
Power

```
Library(s) Used:

    lec25dscc25_SS (File: /home/admin/lib/lec25/lec25dscc25_SS.db)


Operating Conditions: nom_pvt   Library: lec25dscc25_SS
Wire Load Model Mode: top


Global Operating Voltage = 2.25
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  42.3857 mW   (81%)
  Net Switching Power  =  10.0876 mW   (19%)
                         ---------
Total Dynamic Power    =  52.4733 mW  (100%)

Cell Leakage Power     = 532.6506 uW

```

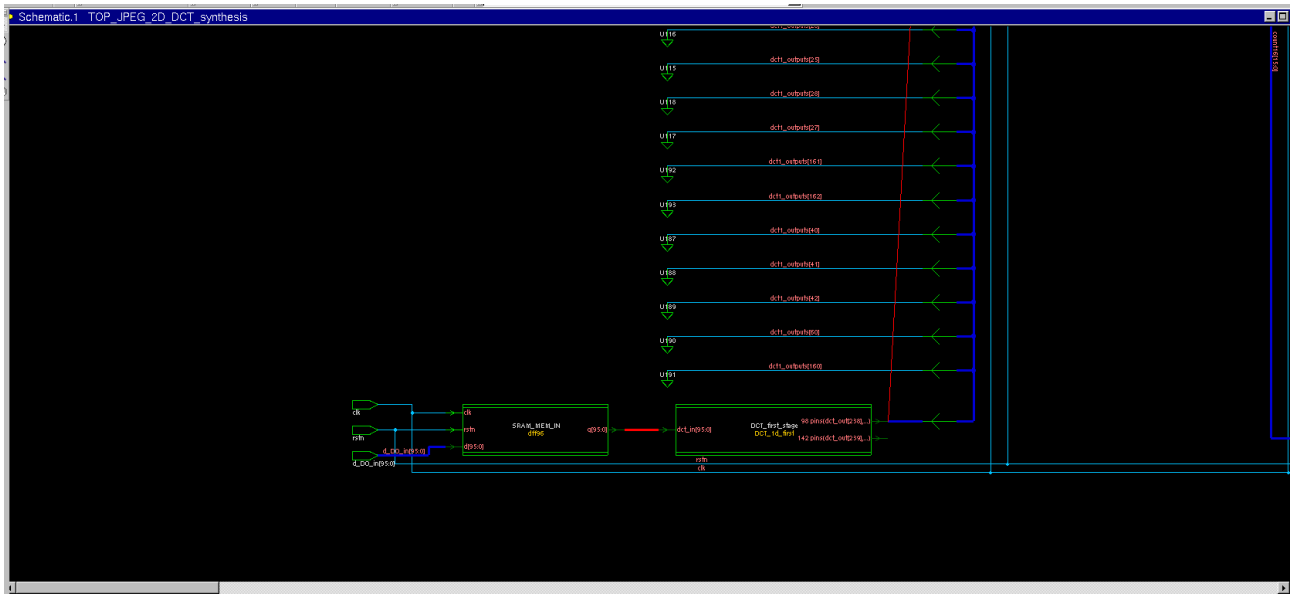
Timing

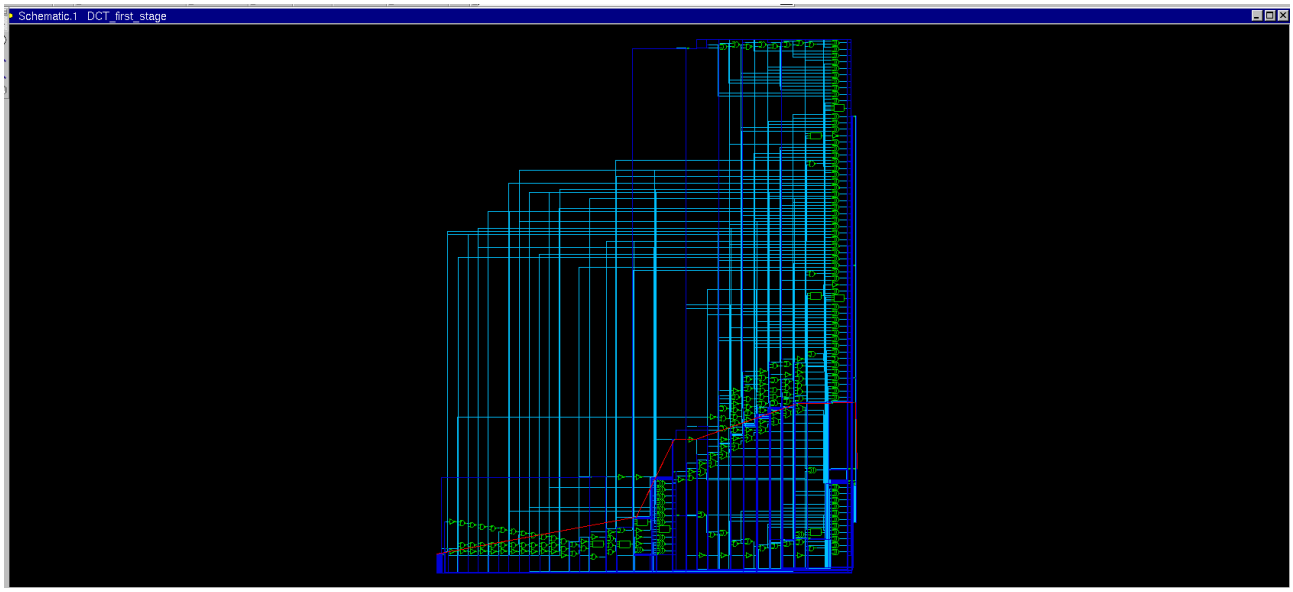
VLSI Project2

```
                                                  0.00       7.74 f
  DCT_first_stage/z5_c11_shifter_adder/U19/OUTC (fadd1s3)
                                                  0.34       8.08 r
  DCT_first_stage/z5_c11_shifter_adder/U41/Q (xor2s2)     0.35       8.42 r
  DCT_first_stage/z5_c11_shifter_adder/U58/Q (and2s2)     0.18       8.60 r
  DCT_first_stage/z5_c11_shifter_adder/U44/Q (or2s2)      0.23       8.83 r
  DCT_first_stage/z5_c11_shifter_adder/U30/OUTS (fadd1s3)
                                                  0.62       9.45 f
  DCT_first_stage/z5_c11_shifter_adder/add_0_root_add_32_2/U1_14/OUTC (fadd1s3)
                                                  0.44       9.89 f
  DCT_first_stage/z5_c11_shifter_adder/add_0_root_add_32_2/U1_15/OUTC (fadd1s3)
                                                  0.38      10.27 f
  DCT_first_stage/z5_c11_shifter_adder/add_0_root_add_32_2/U1_16/OUTC (fadd1s3)
                                                  0.38      10.65 f
  DCT_first_stage/z5_c11_shifter_adder/add_0_root_add_32_2/U1_17/OUTS (fadd1s3)
                                                  0.84      11.49 r
  DCT_first_stage/z5_c11_shifter_adder/out[17] (c1_c3_c5_c7_c9_c11_sa_2)
                                                  0.00      11.49 r
  DCT_first_stage/dct_out[137] (DCT_1d_first)      0.00      11.49 r
  internal_BW_maker/in[137] (BW_maker)             0.00      11.49 r
  internal_BW_maker/out[61] (BW_maker)             0.00      11.49 r
  tp_memory_1/i_data[61] (TP_MEM_mreged_BW9)       0.00      11.49 r
  tp_memory_1/U2672/Q (nnd2s3)                     0.20      11.68 f
  tp_memory_1/U187/Q (nnd2s1)                      0.32      12.00 r
  tp_memory_1/array_reg[5][7]/DIN (dffles1)        0.00      12.00 r
  data arrival time                                          12.00

  clock clk (rise edge)                           12.50      12.50
  clock network delay (ideal)                      0.00      12.50
  tp_memory_1/array_reg[5][7]/CLK (dffles1)        0.00      12.50 r
  library setup time                              -0.50      12.00
  data required time                                         12.00
  --------------------------------------------------------------------------
  data required time                                         12.00
  data arrival time                                         -12.00
  --------------------------------------------------------------------------
  slack (MET)                                                 0.00
```



VLSI Project2



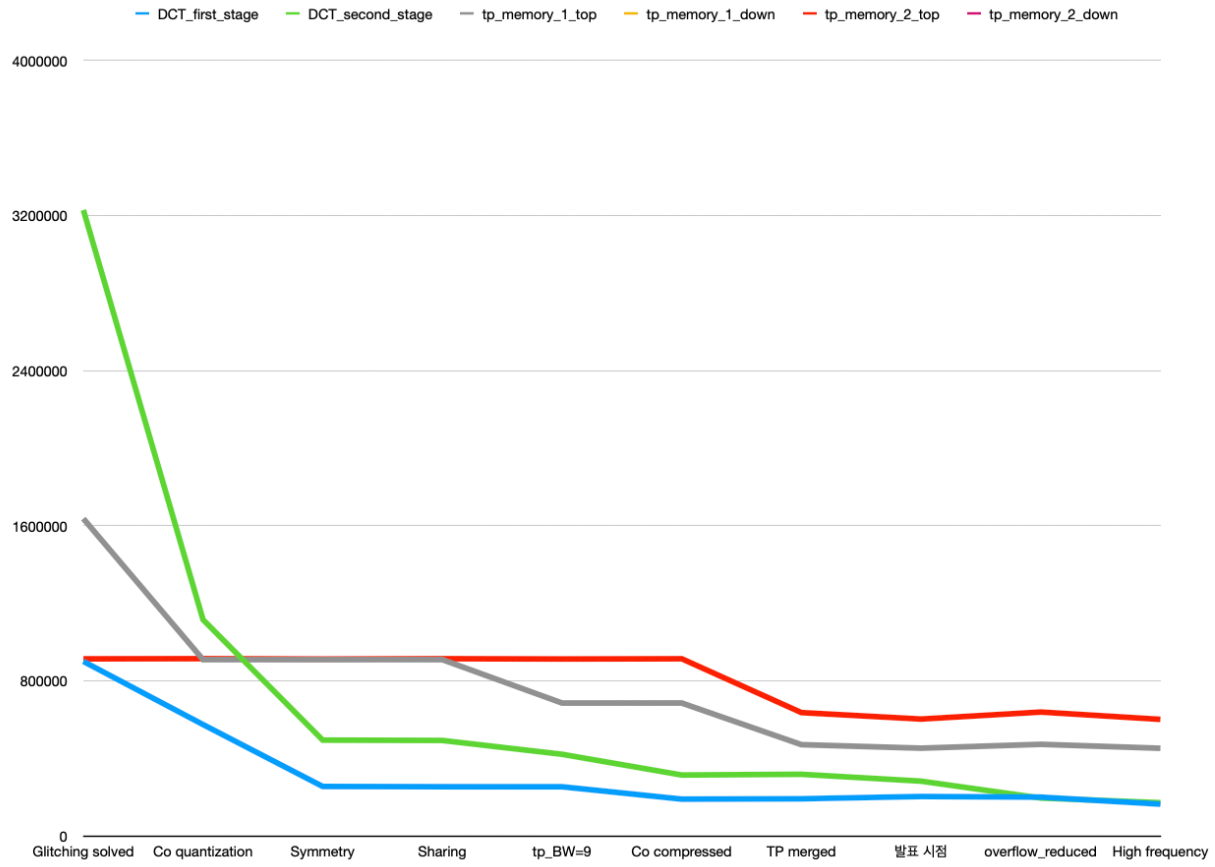


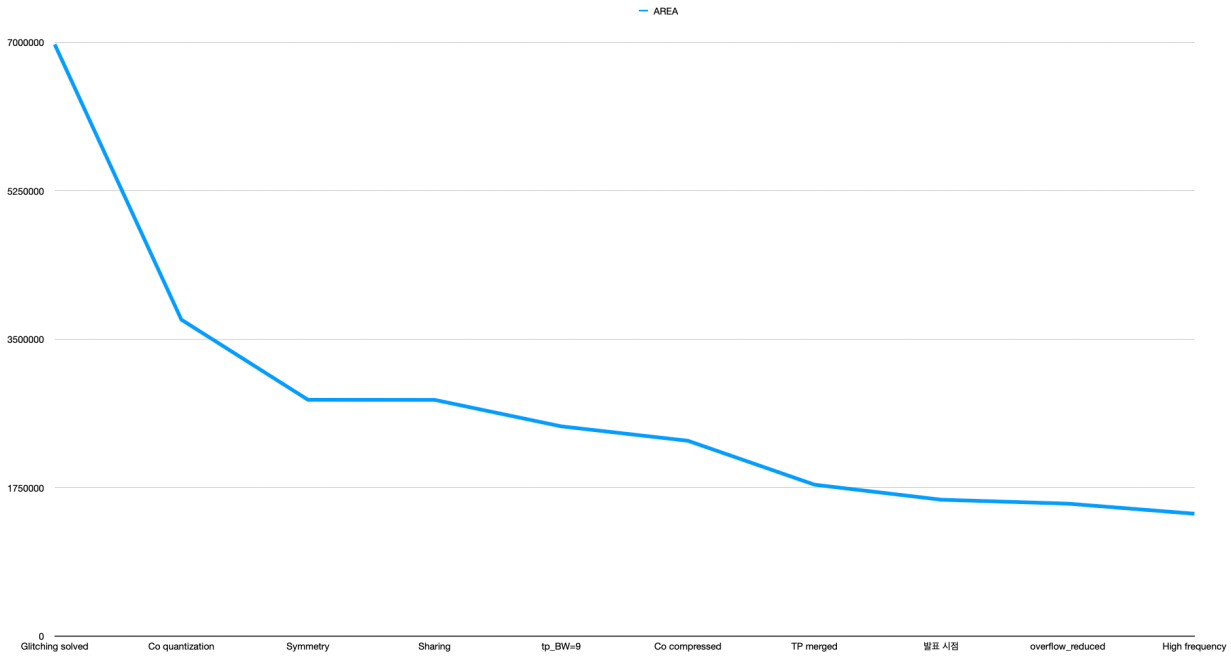
Critical path는 아래 합성결과에서 나오지만, 당연하게 연산이 가장 많은 DCT stage를 지나는 경로로 나타난다.

(과정)

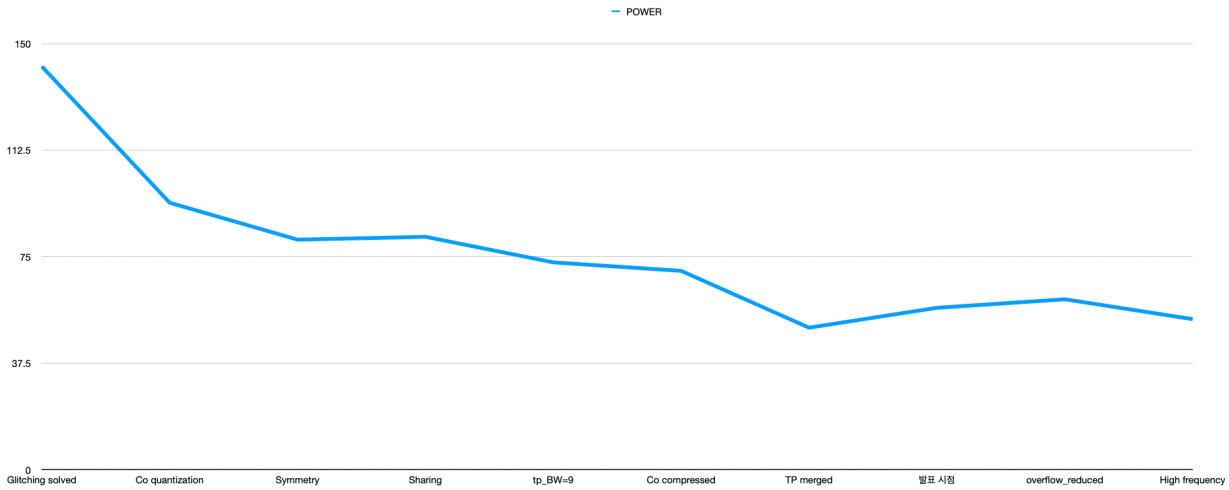
	Glitching solved	Co quantization	Symmetry	Sharing	tp_BW=9	Co compressed	TP merged	발표 시점	overflow_reduced	High frequency
DCT_first_stage	900605	575021	255781	254572	254339	190588	192363	204174	200914	164701
DCT_second_stage	3227965	1115634	494894	493156	421764	314821	318677	283267	196195	172249
tp_memory_1_top	1636641	909818	909793	909793	685996	686104	471420	453243	473186	452754
tp_memory_1_down										
tp_memory_2_top	913949	914811	912821	914629	913219	914281	636736	603389	639516	601791
tp_memory_2_down										

VLSI Project2

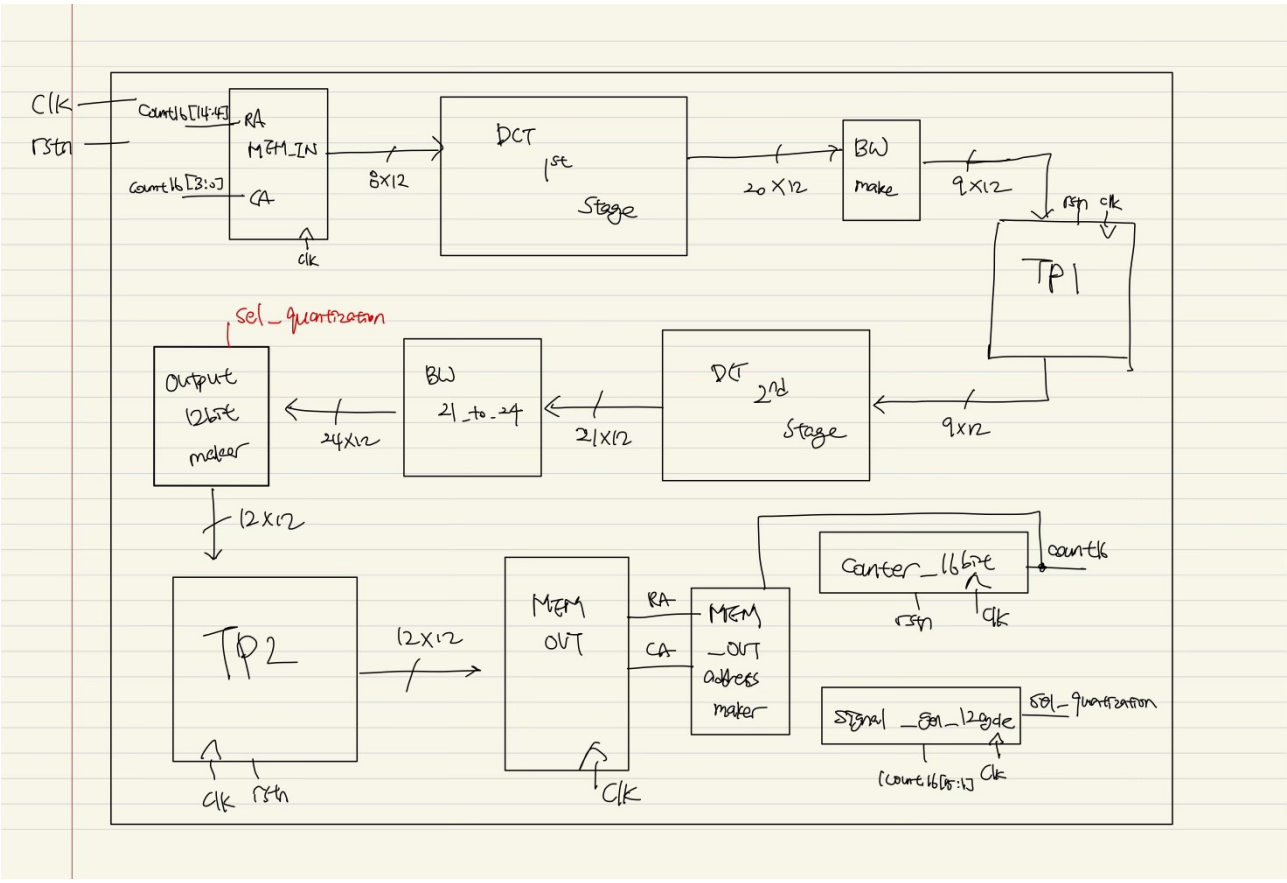




VLSI Project2



4. 사용한 DCT 면적 최소화 기술들



1. Coefficient quantizatoin bit + Result 1D DCT quantization bit

이는 Matlab에서 알고리즘 레벨에서 확인이 가능하다.

10bit일 때,

8bit 일때,

이를 통해서 우리는 최적의 coefficient quantization bit와 result 1d act quantization bit를 설정한다.
PSNR을 30이상으로 약간의 margin을 두는 조합은 (8, 9)이다.
이때, 우리는 PSNR이 가장 취약한 풍차 사진을 중점으로 최적화를 진행해 나간다.
Coefficient quantizatoin bit = 8 / Result 1D DCT quantization bit = 9

2. TP Memory Merging

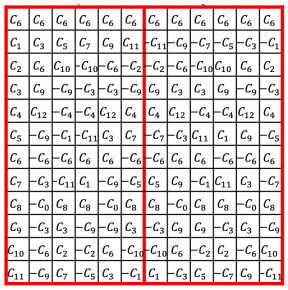
기존의 TP memory는 12cycle 동안 row방향으로 write 후, 다음 12cycle동안 column방향으로 read를 하는 구조였다. 이는 1D DCT의 결과를 12개 마다 TP_MEM을 switching해가면서 사용해야되는 비효율이 있다.
따라서 이번 실험에서 TP Memory를 작게 만드는 것은 면적을 최소화하는 가장 중요한 지점이였다.
이는 TP Memory가 reg를 사용하기 때문이다. Reg는 데이터를 저장하기 위해 면적을 많이 사용한다. 우리가 사용하는 데이터가 12x12xBW의 bit를 가지기 때문에 TP_MEM을 두개 활용하는 것은 엄청난 면적에서의 손해를 야기한다.
TP Memory를 하나로 합치는 방법은 간단하다. 우리가 기존에 12cycle마다 toggle하면서 사용했던 TP Memory의 throughput을 만족시키기 위해서는 read와 write가 동시에 진행 될 수 밖에 없다. 이것이 가능한 이유는 dff의 clk에 synchronous한 특성 때문이다. Column 방향을 데이터를 posedge에 data_out reg에 넘길 때, 동시에 앞단 dff에서 i_data를 제공하여 read와 write를 동시에 하게 만든다.
이때 write가 이제 column방향으로 이루어지기 때문에, 다음 read에서는 row 방향으로 read를 하여야, 결과적으로 column방향을 읽는 효과를 가져온다. 우리는 지금 transpose memory를 다루기 때문에 무조건 들어온 데이터의 반대 방향으로 읽어야 한다.

코드에서 살펴보면, w_rowise(write row-wise) 핀을 활용하여서 write 작업을 case문으로 index별로 진행할 수 있게 해준다.

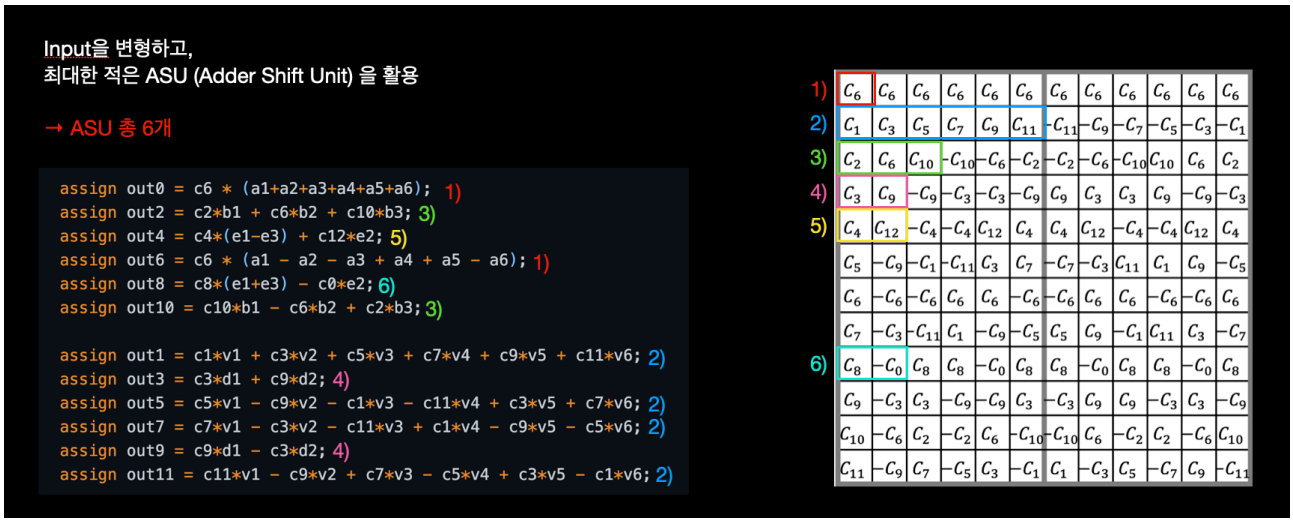
```
    else if(w_rowise)   array[index] <= i_data;
    else if(!w_rowise) begin
        case(index)
            4'b0000: begin
                array[0][12*BW-1:11*BW] <= i_data[12*BW-1:11*BW];
                array[1][12*BW-1:11*BW] <= i_data[11*BW-1:10*BW];
                array[2][12*BW-1:11*BW] <= i_data[10*BW-1:9*BW];
                array[3][12*BW-1:11*BW] <= i_data[9*BW-1:8*BW];
                array[4][12*BW-1:11*BW] <= i_data[8*BW-1:7*BW];
                array[5][12*BW-1:11*BW] <= i_data[7*BW-1:6*BW];
                array[6][12*BW-1:11*BW] <= i_data[6*BW-1:5*BW];
                array[7][12*BW-1:11*BW] <= i_data[5*BW-1:4*BW];
                array[8][12*BW-1:11*BW] <= i_data[4*BW-1:3*BW];
                array[9][12*BW-1:11*BW] <= i_data[3*BW-1:2*BW];
                array[10][12*BW-1:11*BW] <= i_data[2*BW-1:1*BW];
                array[11][12*BW-1:11*BW] <= i_data[1*BW-1:0*BW];
            end
            4'b0001: begin
                array[0][11*BW-1:10*BW] <= i_data[12*BW-1:11*BW];
                array[1][11*BW-1:10*BW] <= i_data[11*BW-1:10*BW];
                array[2][11*BW-1:10*BW] <= i_data[10*BW-1:9*BW];
                array[3][11*BW-1:10*BW] <= i_data[9*BW-1:8*BW];
                array[4][11*BW-1:10*BW] <= i_data[8*BW-1:7*BW];
```

→ 결과적으로 TP_MEM은 면적은 TP x4 = 16만 에서 TP x2 = 11만으로 대략 30% 정도 감소한 모습을 볼 수 있다.

3. Coefficient Symmetry property



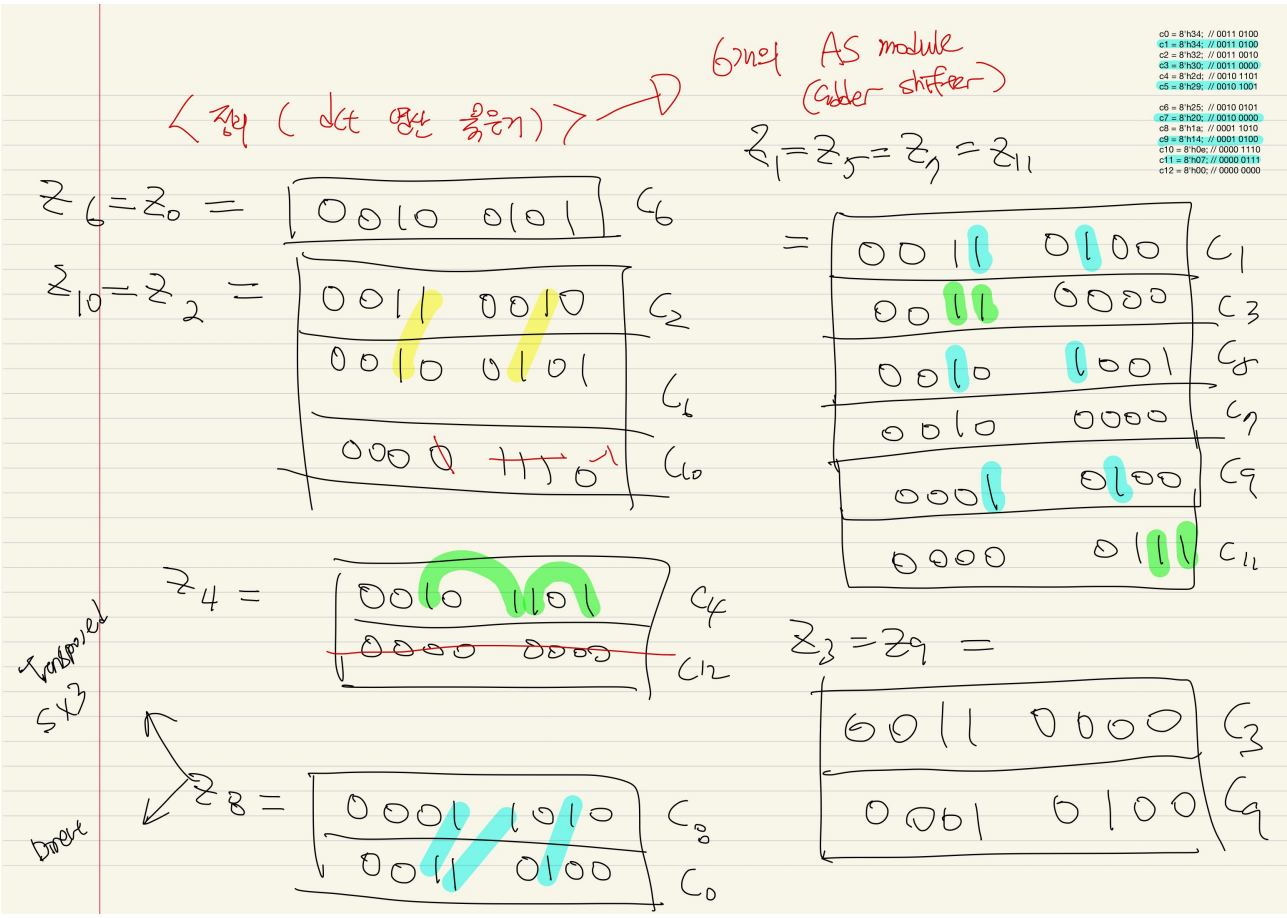
우리가 사용하는 12x12 coefficient matrix를 보면,
Even part와 odd part에서 각가 symmetry한 것을 볼 수 있다.
Odd의 경우에는 [C1, C3, C5, C7, C9, C11], [C3, C9]을 활용한 모듈을 만들 수 있다.
특히 Even의 경우에는 대칭이 두번까지도 가능하기 때문에 모듈을 더 반복해서 활용할 수 있다. [C6], [C2, C6, C10], [C4, C12], [C0, C8] 을 활용하면, input을 이용하여 통일화된 계산이 가능하다.
이러한 symmetry 특성은 multiplier 개수를 줄여주어, 면적에서 이득을 볼 수 있다.



여기 까지는 verilog module에서 노골적으로 C0=8'h34;와 같이 coefficient를 사용하여 compile을 하였다. 이 때, 직전에 coefficient만 8bit로 quantization 하였을 때에 비해서 100만 um^2 정도 줄어든 것을 볼 수 있다. 이는 멀티플라이어가 얼마나 면적에 취약한지를 확인할 수 있는 결과 였다.

VLSI Project2

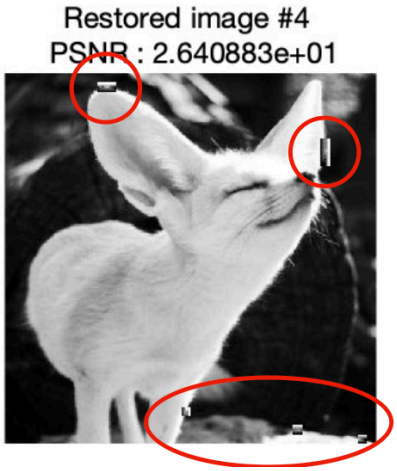
4. Coefficient Sub-expression sharing



우리는 3. Coefficient Symmetry property에서 6가지의 ASU(adder shifter unit)을 구성할 수 있게 묶었다. 이를 이용하여 sharing을 찾고, multiply를 shifting과 adder만을 가지고 효율적으로 표현할 수 있다.

하지만, synthesis 결과를 확인하여 보면 parameter를 직접적으로 C0 = 8'h34;와 같이 선언하였을 때와 크게 면적에서 차이가 나지 않음을 확인 할 수 있다. 이는 design compiler 자체에서 fix된 coefficient를 분석하여서 이미 최적화를 진행하여, 무작정 multiplier를 사용하지는 않았기 때문이라고 할 수 있다. 하지만 그럼에도 compiler와 수작업으로 진행한 모듈이라는 점을 고려하면 1000um^2 감소 또한 상당한
의미를 가진다고 생각된다.

5. Glitching Effect 관리



Glitching 같은 경우에는 비트연산에서 주의하여야 하는 overflow와 underflow에 연관이 되어 있다.
Overflow의 경우, 최상위 비트를 초과하여 값이 반전되는 상황을 말한다.
Underflow의 경우에도 마찬가지로 음의 방향으로 비트를 초과하여서 반전되는 상황이다.
우리는 DCT에서 이를 glitching으로 확인 할 수 있다. 특히 해당 basis와 비슷한 패턴이 나타나는 부분에서 내적값이 너무 폭발하여서 아애 패턴이 반전되어 나타나는 양상을 보인다. 이는 특히 여우 사진과 같은 horizontal하거나 vertical한 edge가 많은 사진에 나타난다.

특히 DCT에서 해당 effect는 마지막 12bit truncation에서 발생한다고 볼 수 있다. 따라서 truncation 되기 전의 MSB(sign bit)와 truncation 이후의 MSB를 비교하여 overflow와 underflow를 확인하고, overflow의 경우 12bit 최대 양수인, 12'b0111_1111_1111으로

VLSI Project2

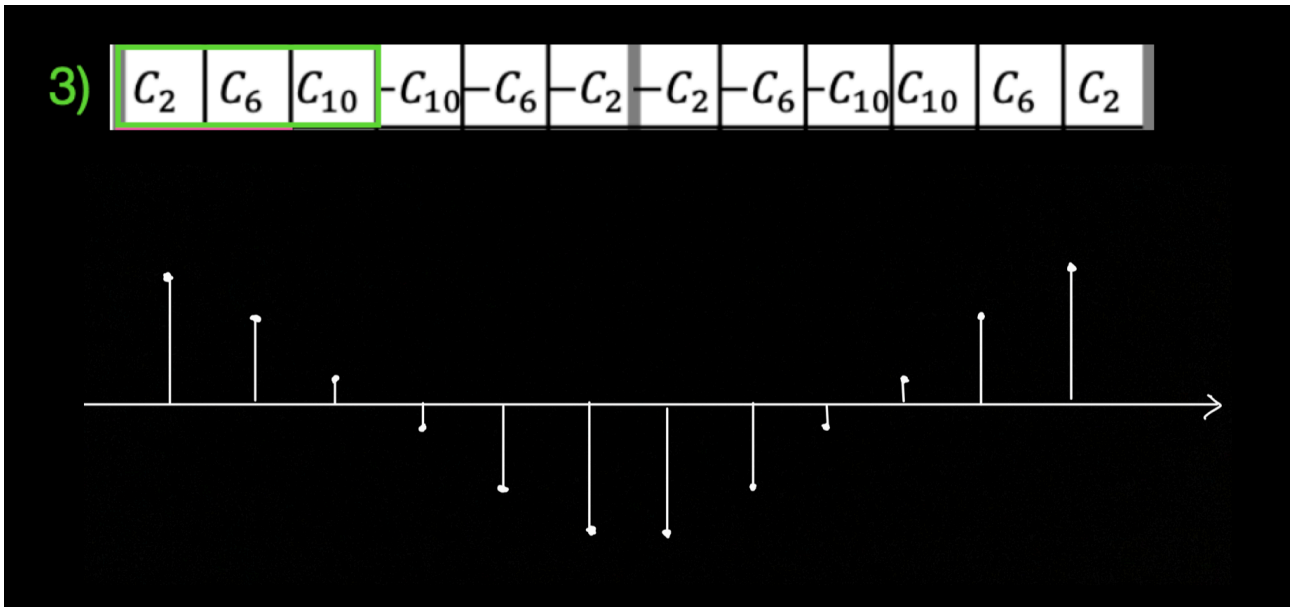
값을 할당하고, underflow의 경우 12bit 음수 최대값인 12'b1000_0000_0000으로 값을 대체하여 이를 해결 하

였다.

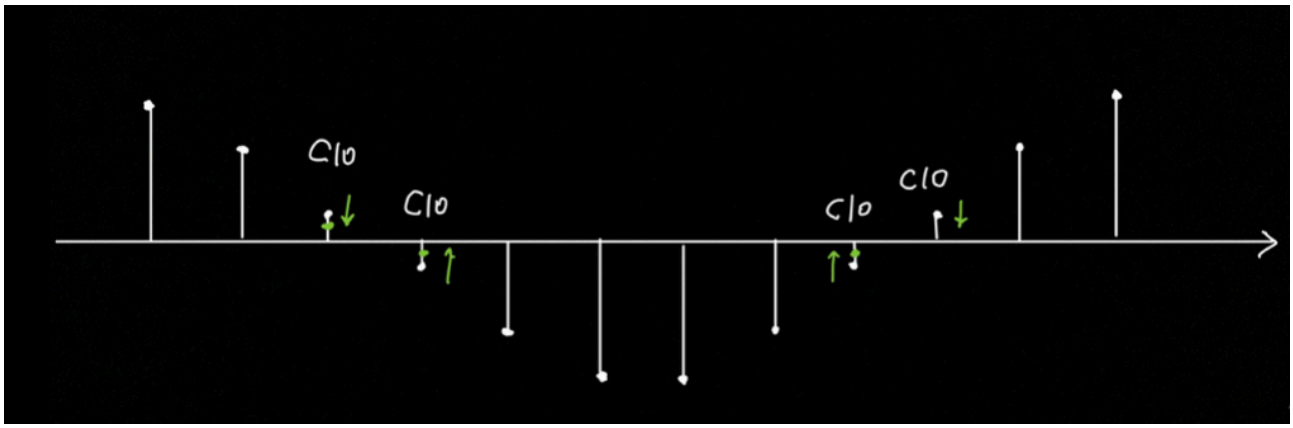
6. Coefficient Compression based on frequency basis

이 부분이 가장 System적인 부분을 반연하는 HW를 구성하는 방법이라는 생각이 든다.

먼저 3번 ASU를 보면, 해당 coefficient 행은 아래와 같은 cosine basis를 나타낸다.
우리는 DCT를 통해서 해당 frequency 성분을 cosine함수를 이용하여서 간접적으로 구하고자 하는 것이다.
Frequency의 측면에서 본다면 우리는 2가지에 집중하여야 한다. Peak부분과 zero-crossing부분의 정보는 잃으면 안된다는 것이다. 따라서 C2를 건드린다면 아마 DCT결과에 상당한 영향을 끼칠 것이다.



여기서 우리는 이제 PSNR을 망치지 않는 Coefficient를 발견해서 미세한 조정을 할 것이다.
이러한 미세한 조정은 sharing으로 묶이지 않는 coefficient part를 drop하여 연산자를 줄일 수도 있고, sharing을 만들 수 있도록 도와주는 역할을 할 수도 있다.
여기서는 C10을 8'b0e에서 8'b09로 바꿈으로서 sharing되지 않는 부분을 drop하면서 동시에 adder 연산을 하나 줄일 수 있다.



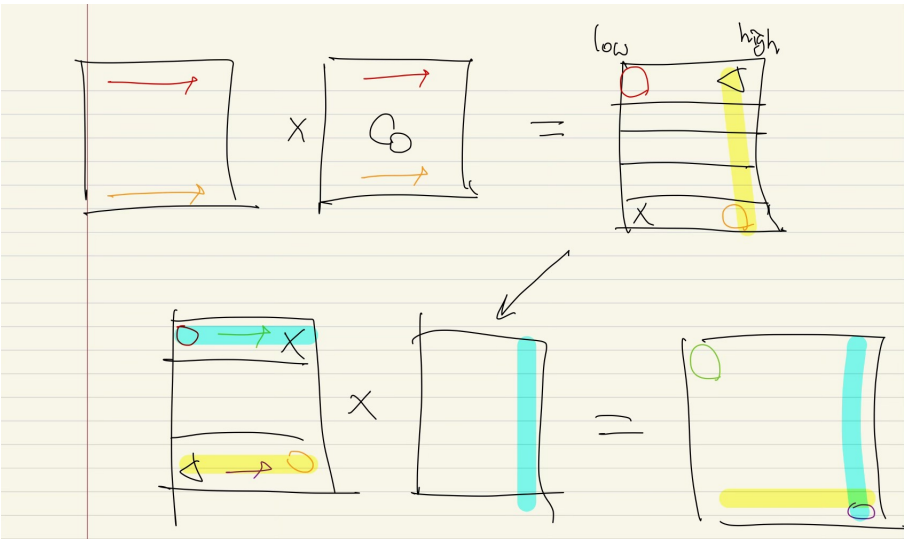
이러한 방식으로 각 ASU에 대해서 일일이 적용한 결과, 아래와 같은 coefficient의 compression이 가능했다.

```
// coefficient original
parameter signed c0 = 8'h34;
parameter signed c1 = 8'h30;        // 34 to 30
parameter signed c2 = 8'h32;
parameter signed c3 = 8'h30;
parameter signed c4 = 8'h2d;
parameter signed c5 = 8'h30;        // 29 to 30

parameter signed c6 = 8'h25;
parameter signed c7 = 8'h20;
parameter signed c8 = 8'h1a;
parameter signed c9 = 8'h18;        // 14 to 18
parameter signed c10 = 8'h09;       // 0e to 09
parameter signed c11 = 8'h06;       // 07 to 06
parameter signed c12 = 8'h00;
```

7. High frequency Computation reducing

그리고 우리는 DCT를 활용할 때 가장 큰 장점은 high frequency에 정보량이 많이 없기 때문에, pixel value로 저장하면 많은 공간을 차지할 데이터를 손쉽게 압축할 수 있다. 특히 high frequency성분을 직접적으로 masking한다면, 면적을 줄이면서도 psnr 성능을 유지할 수 있다는 실험적 결과를 얻었다.

이미 위의 과정들을 통해서 psnr 마진이 많지 않았던 상황에서,
First stage DCT와 Second stage DCT의 out11, out10을 0으로 선언하여도 psnr의 손실이 없음을 확인하였다.
이때 first satge와 second stage dct의 high freqency 성분을 표시해보면 아래와 같다.



우리는 dct1과 dct2를 동일하게 high frequency를 제거해야, 최종적인 2D DCT에서의 동일한 frequency 성분이 통일성 있게 지워질 수 있음을 확인 할 수 있다.
out9 결과까지 0으로 치환하면 풍차그림에서 psnr이 29로 떨어져 out10까지만 0으로 선언하였다.

추가적으로 tp_mem에서도 해당 배열에 해당하는 부분을 0으로 값을 지정하도록 하여 reg가 필요없음을 명시적으로 보여주려고 하였다.

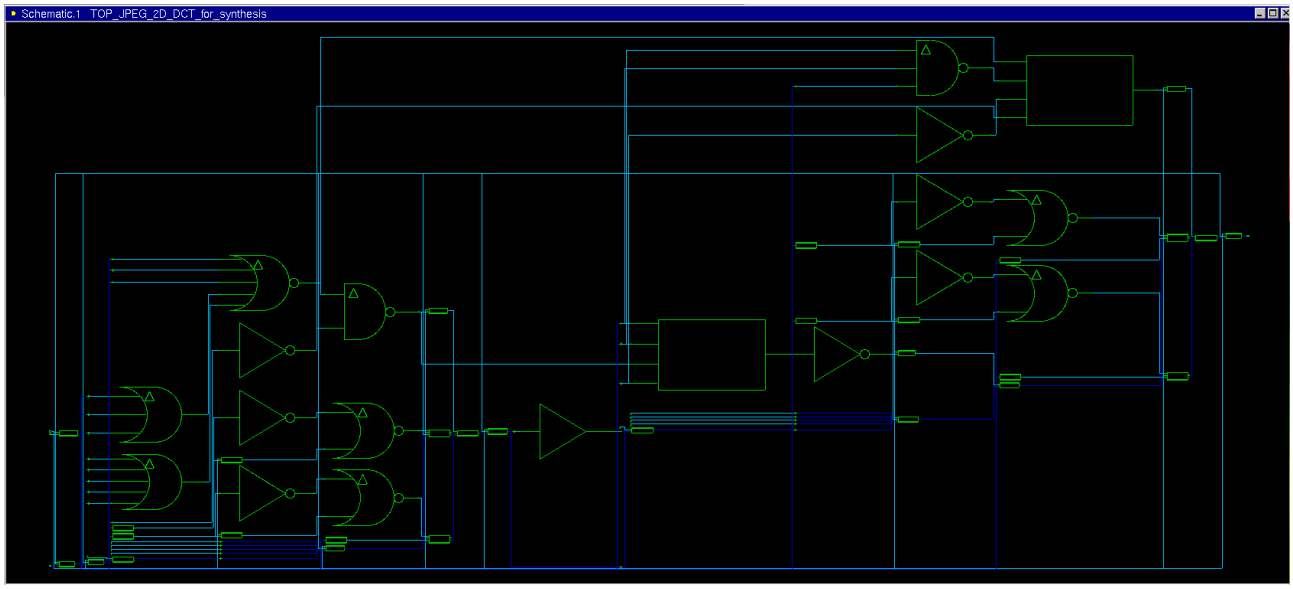
VLSI Project2

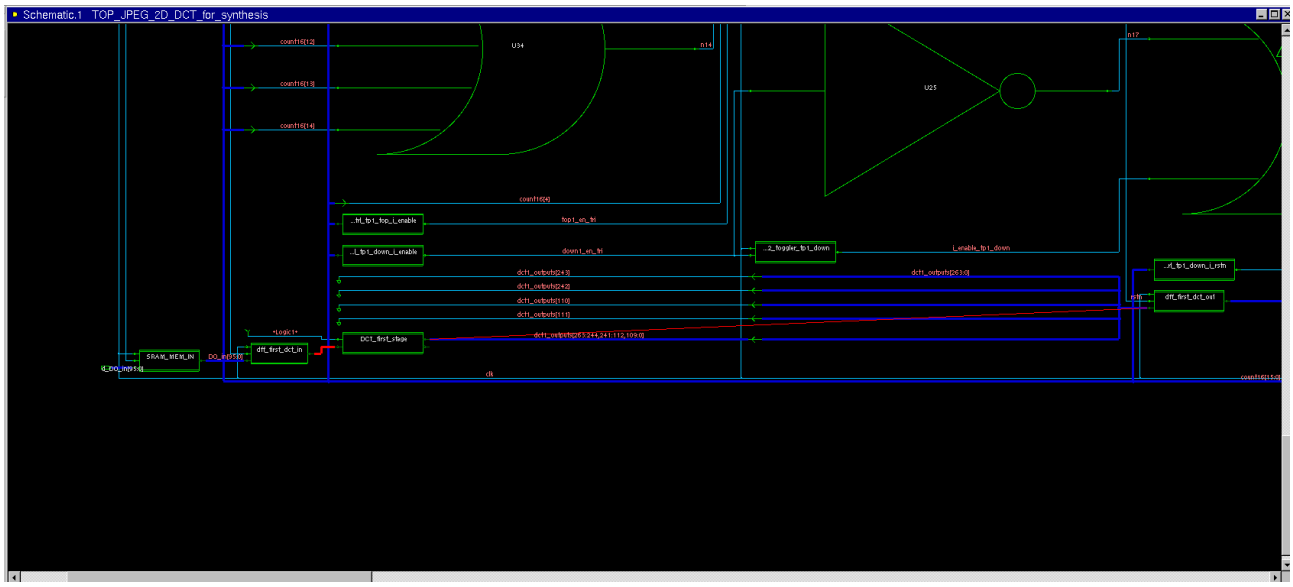
```
    end
    4'b0111: begin
        array[0][5*BW-1:4*BW] <= i_data[12*BW-1:11*BW];
        array[1][5*BW-1:4*BW] <= i_data[11*BW-1:10*BW];
        array[2][5*BW-1:4*BW] <= i_data[10*BW-1:9*BW];
        array[3][5*BW-1:4*BW] <= i_data[9*BW-1:8*BW];
        array[4][5*BW-1:4*BW] <= i_data[8*BW-1:7*BW];
        array[5][5*BW-1:4*BW] <= i_data[7*BW-1:6*BW];
        array[6][5*BW-1:4*BW] <= i_data[6*BW-1:5*BW];
        array[7][5*BW-1:4*BW] <= i_data[5*BW-1:4*BW];
        array[8][5*BW-1:4*BW] <= i_data[4*BW-1:3*BW];
        array[9][5*BW-1:4*BW] <= i_data[3*BW-1:2*BW];
        array[10][5*BW-1:4*BW] <= {BW{1'b0}};
        array[11][5*BW-1:4*BW] <= {BW{1'b0}};
    end
    4'b1000: begin
        array[0][4*BW-1:3*BW] <= i_data[12*BW-1:11*BW];
        array[1][4*BW-1:3*BW] <= i_data[11*BW-1:10*BW];
        array[2][4*BW-1:3*BW] <= i_data[10*BW-1:9*BW];
        array[3][4*BW-1:3*BW] <= i_data[9*BW-1:8*BW];
        array[4][4*BW-1:3*BW] <= i_data[8*BW-1:7*BW];
        array[5][4*BW-1:3*BW] <= i_data[7*BW-1:6*BW];
        array[6][4*BW-1:3*BW] <= i_data[6*BW-1:5*BW];
        array[7][4*BW-1:3*BW] <= i_data[5*BW-1:4*BW];
        array[8][4*BW-1:3*BW] <= i_data[4*BW-1:3*BW];
        array[9][4*BW-1:3*BW] <= i_data[3*BW-1:2*BW];
        array[10][4*BW-1:3*BW] <= {BW{1'b0}};
        array[11][4*BW-1:3*BW] <= {BW{1'b0}};
    end
```

```
assign col[ 0] = {A_00 [ 0], A_01 [ 0], A_02 [ 0], A_03 [ 0], A_04 [ 0], A_05 [ 0], A_06 [ 0], A_07 [ 0], A_08 [ 0], A_09 [ 0], A_10 [ 0], A_11 [ 0]};
assign col[ 1] = {A_00 [ 1], A_01 [ 1], A_02 [ 1], A_03 [ 1], A_04 [ 1], A_05 [ 1], A_06 [ 1], A_07 [ 1], A_08 [ 1], A_09 [ 1], A_10 [ 1], A_11 [ 1]};
assign col[ 2] = {A_00 [ 2], A_01 [ 2], A_02 [ 2], A_03 [ 2], A_04 [ 2], A_05 [ 2], A_06 [ 2], A_07 [ 2], A_08 [ 2], A_09 [ 2], A_10 [ 2], A_11 [ 2]};
assign col[ 3] = {A_00 [ 3], A_01 [ 3], A_02 [ 3], A_03 [ 3], A_04 [ 3], A_05 [ 3], A_06 [ 3], A_07 [ 3], A_08 [ 3], A_09 [ 3], A_10 [ 3], A_11 [ 3]};
assign col[ 4] = {A_00 [ 4], A_01 [ 4], A_02 [ 4], A_03 [ 4], A_04 [ 4], A_05 [ 4], A_06 [ 4], A_07 [ 4], A_08 [ 4], A_09 [ 4], A_10 [ 4], A_11 [ 4]};
assign col[ 5] = {A_00 [ 5], A_01 [ 5], A_02 [ 5], A_03 [ 5], A_04 [ 5], A_05 [ 5], A_06 [ 5], A_07 [ 5], A_08 [ 5], A_09 [ 5], A_10 [ 5], A_11 [ 5]};
assign col[ 6] = {A_00 [ 6], A_01 [ 6], A_02 [ 6], A_03 [ 6], A_04 [ 6], A_05 [ 6], A_06 [ 6], A_07 [ 6], A_08 [ 6], A_09 [ 6], A_10 [ 6], A_11 [ 6]};
assign col[ 7] = {A_00 [ 7], A_01 [ 7], A_02 [ 7], A_03 [ 7], A_04 [ 7], A_05 [ 7], A_06 [ 7], A_07 [ 7], A_08 [ 7], A_09 [ 7], A_10 [ 7], A_11 [ 7]};
assign col[ 8] = {A_00 [ 8], A_01 [ 8], A_02 [ 8], A_03 [ 8], A_04 [ 8], A_05 [ 8], A_06 [ 8], A_07 [ 8], A_08 [ 8], A_09 [ 8], A_10 [ 8], A_11 [ 8]};
assign col[ 9] = {A_00 [ 9], A_01 [ 9], A_02 [ 9], A_03 [ 9], A_04 [ 9], A_05 [ 9], A_06 [ 9], A_07 [ 9], A_08 [ 9], A_09 [ 9], A_10 [ 9], A_11 [ 9]};
assign col[10] = {BLOCK_SIZE*BW{1'b0}};
assign col[11] = {BLOCK_SIZE*BW{1'b0}};
```

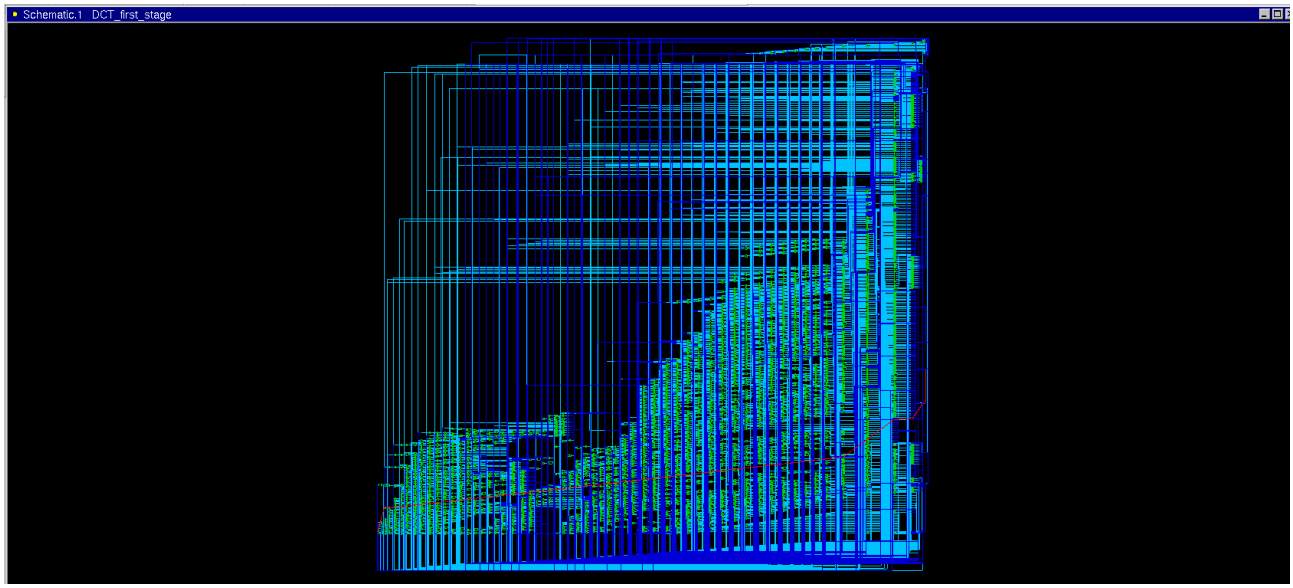
5. Schematics

10 / 22 / glitching removed



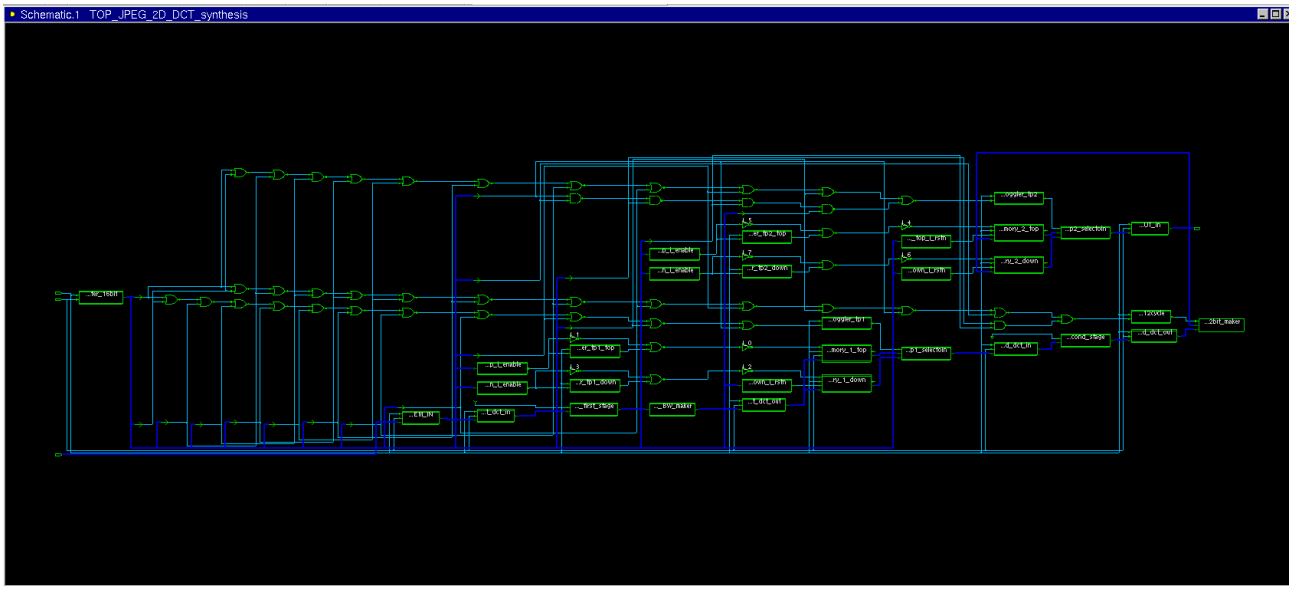


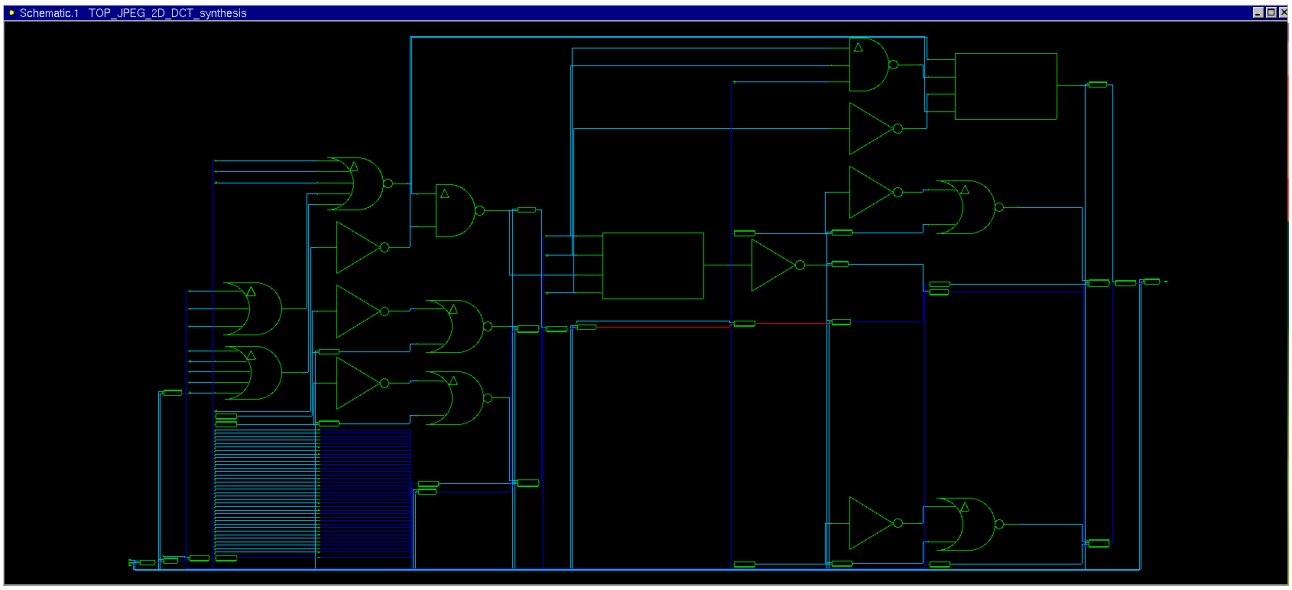
dct_first_stage

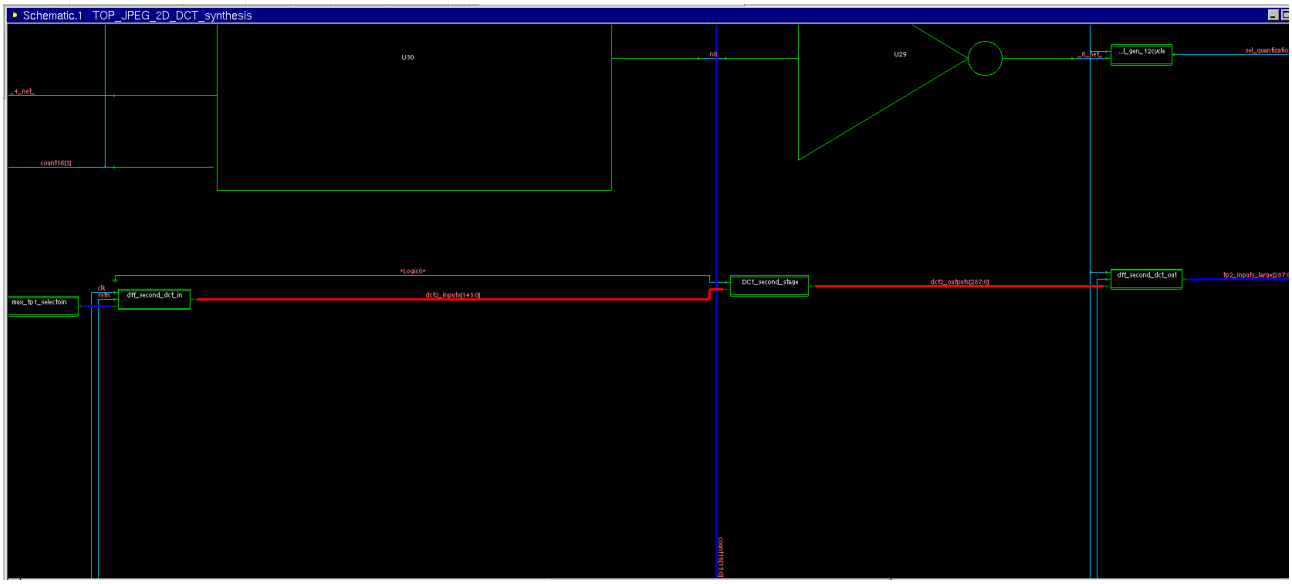


VLSI Project2

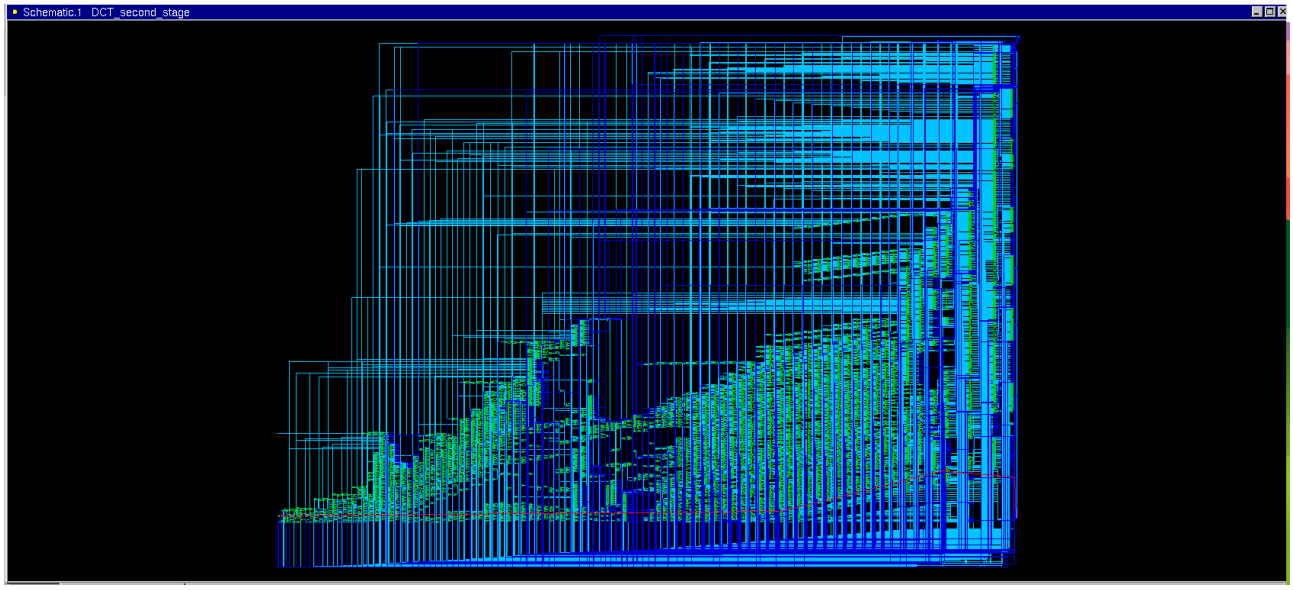
8/9





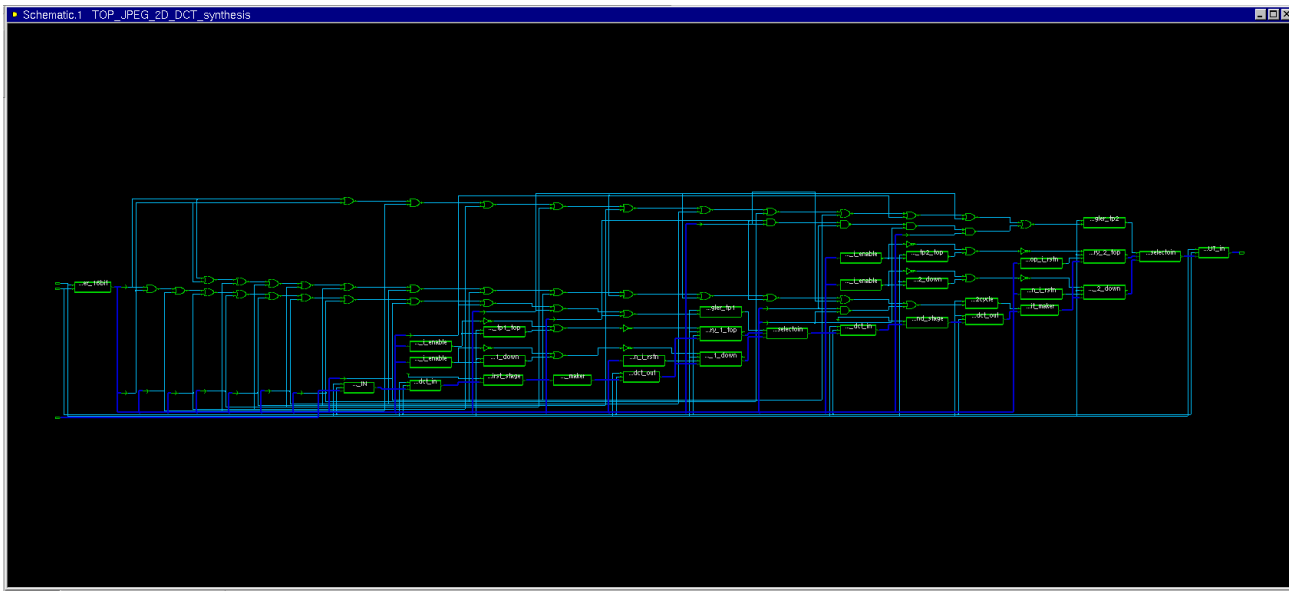


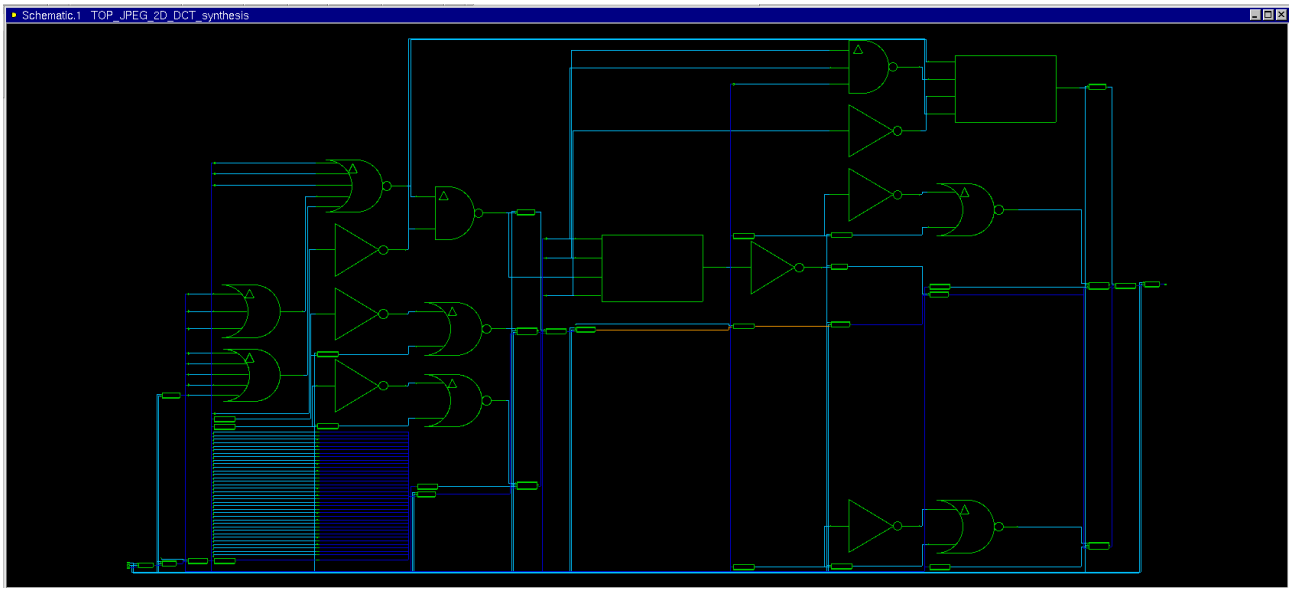
VLSI Project2

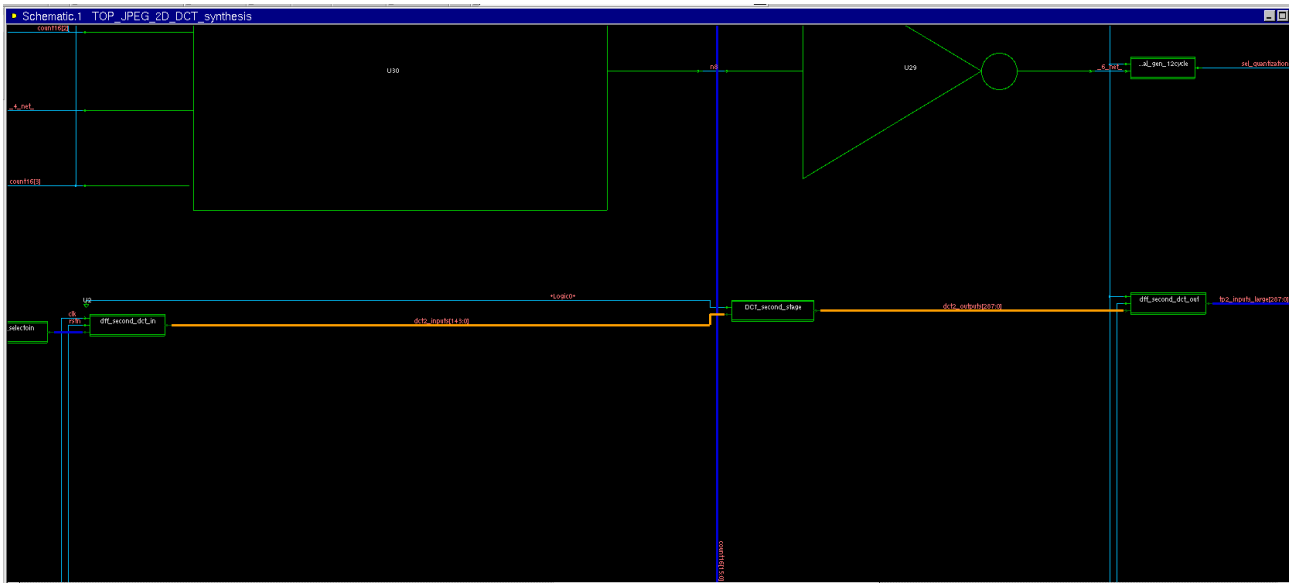


VLSI Project2

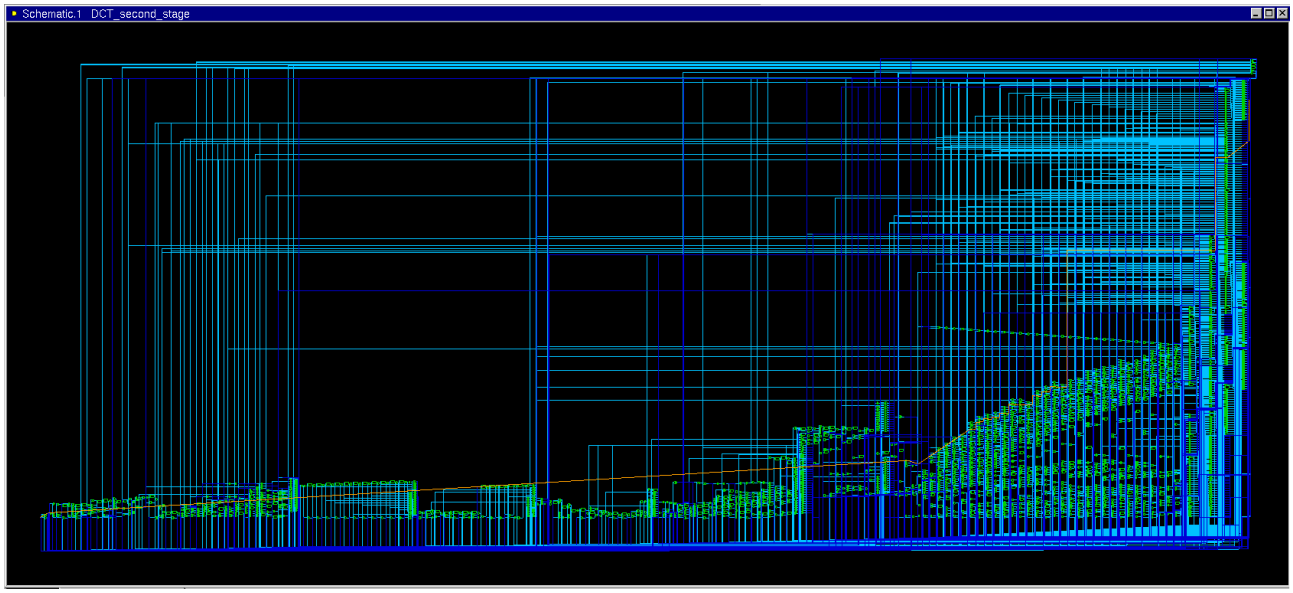
8 / 9 / symmetric





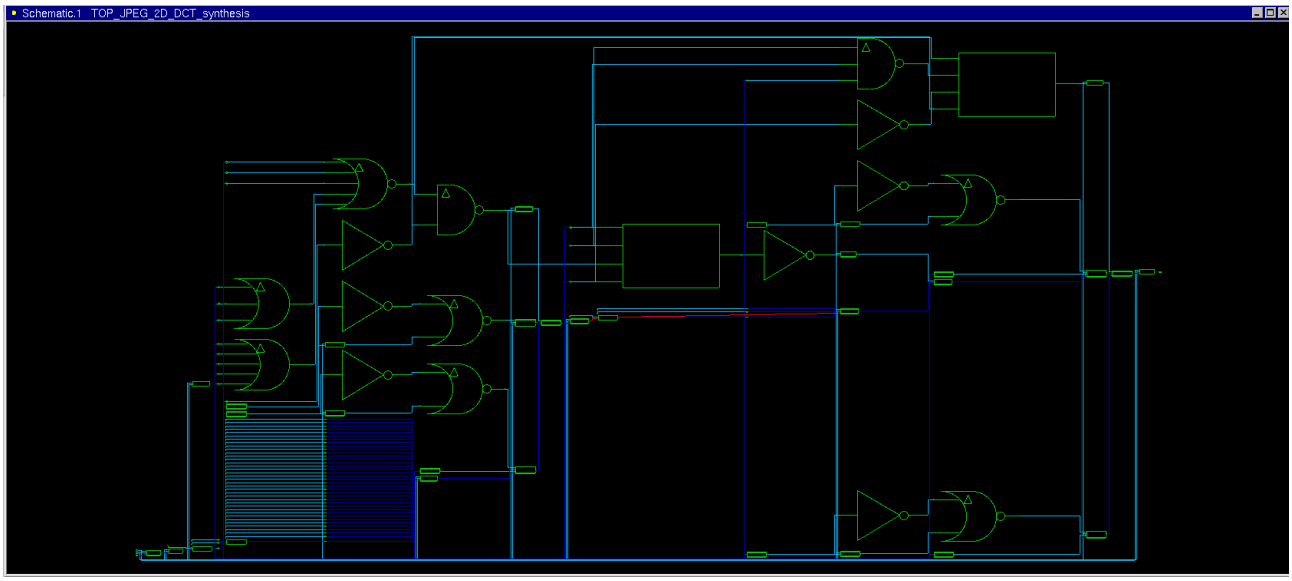


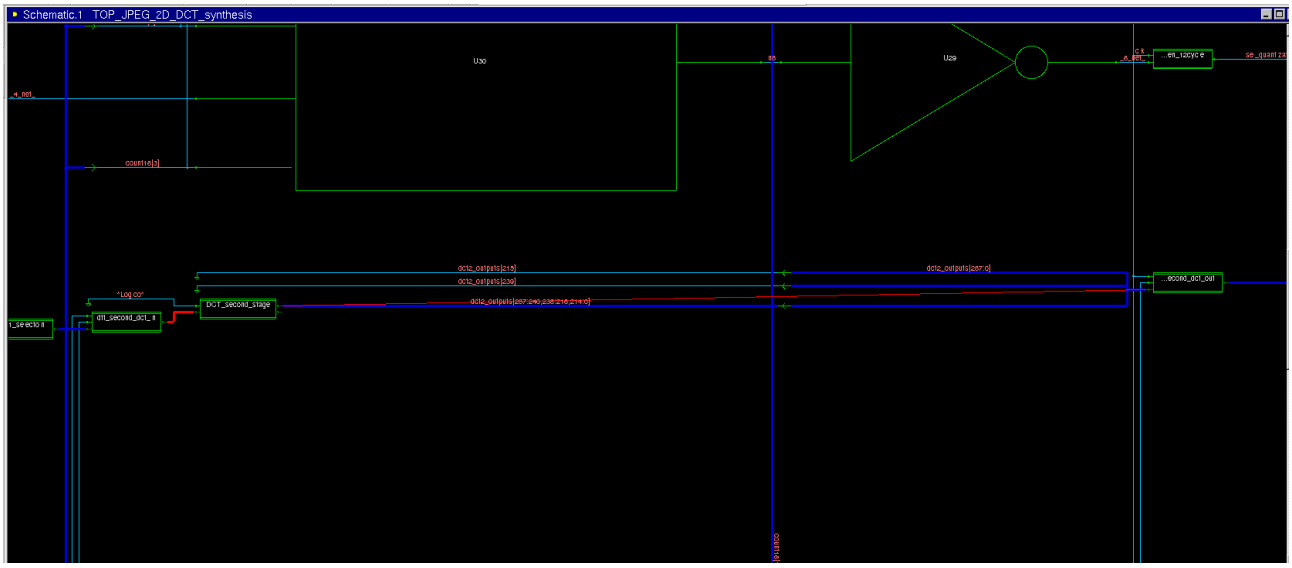
VLSI Project2

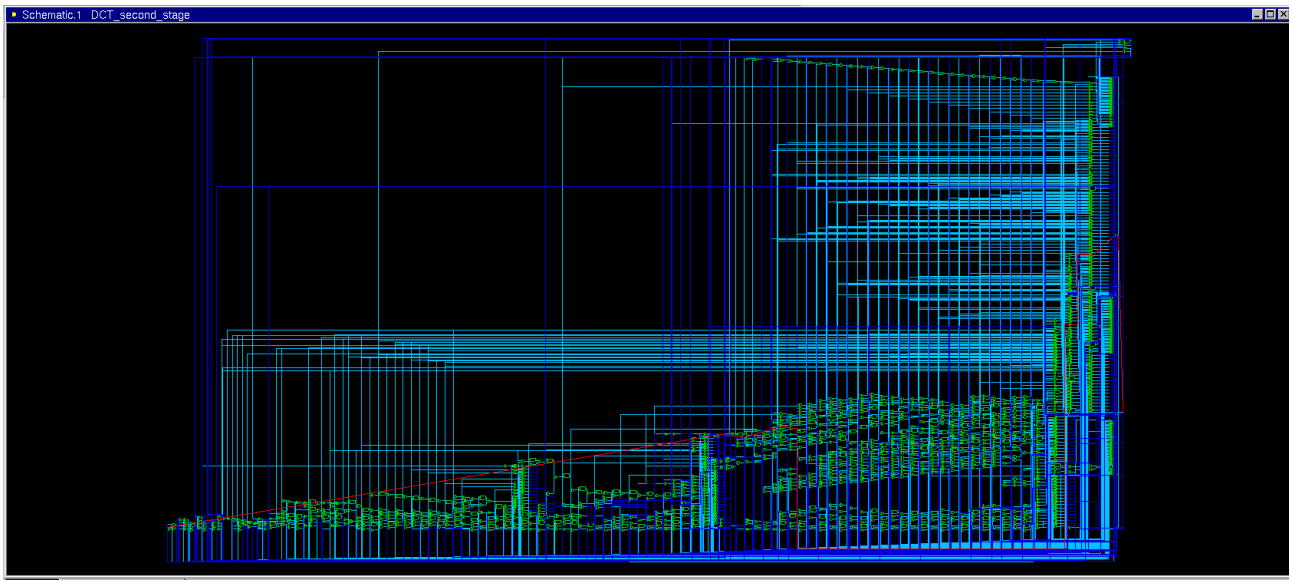


VLSI Project2

8 / 9 / symmetric ver2

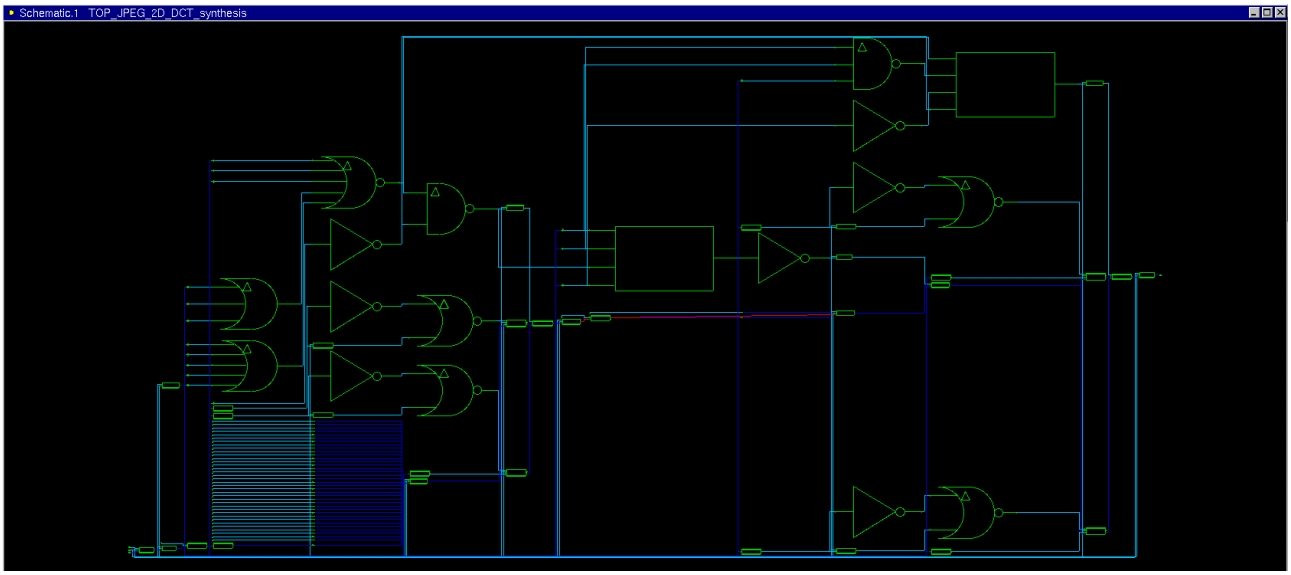


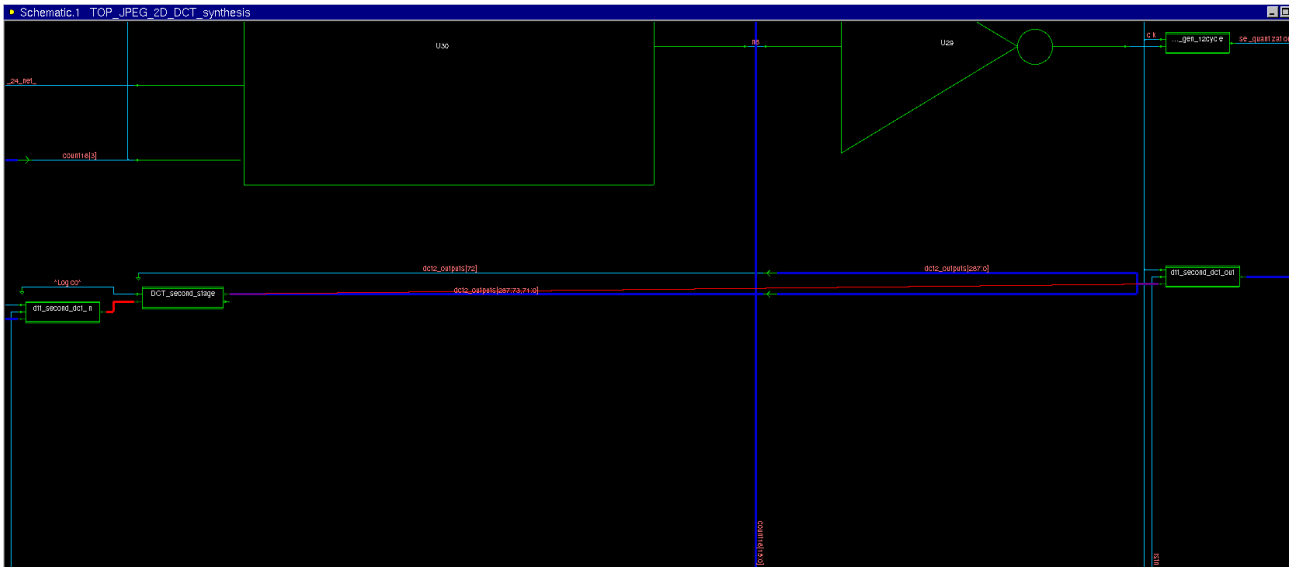


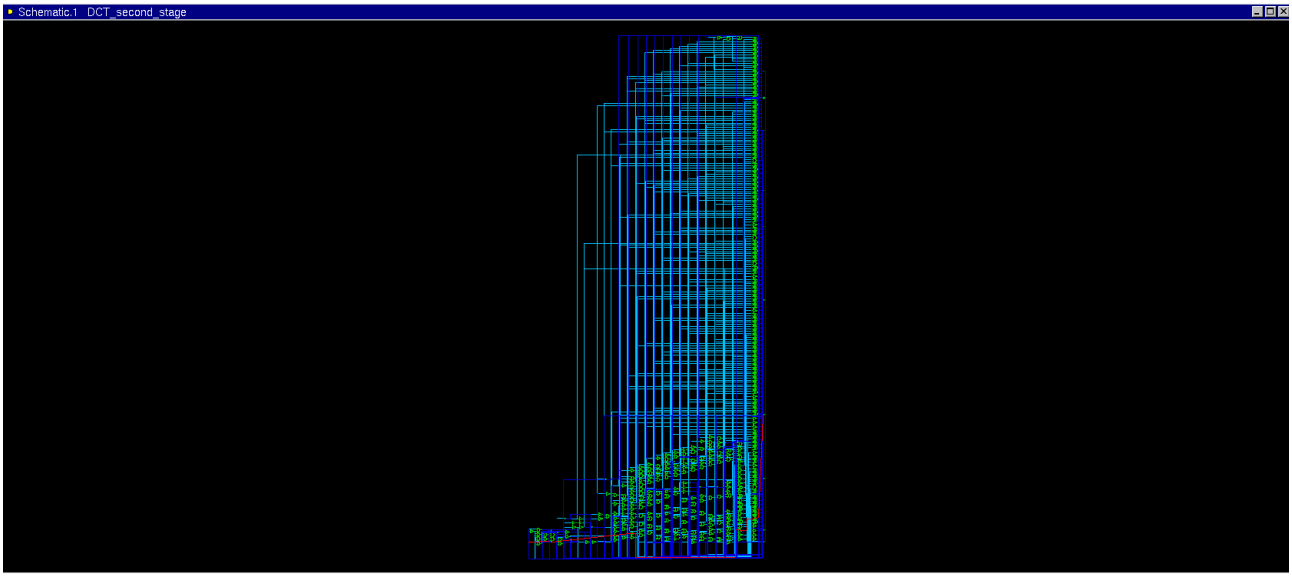


VLSI Project2

8 / 9 / sharing_exact

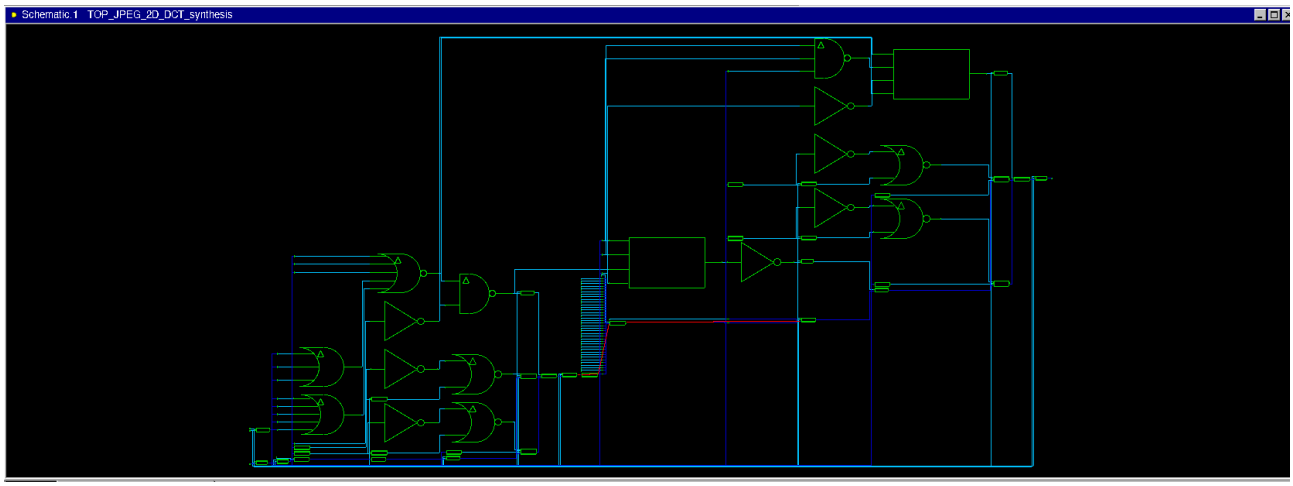






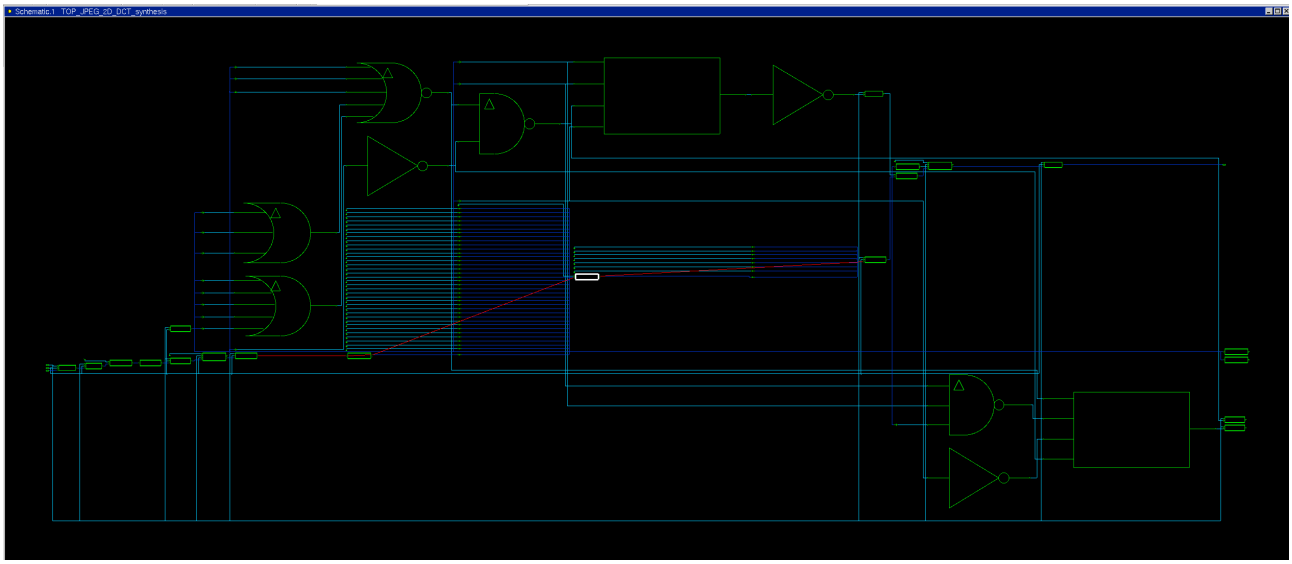
VLSI Project2

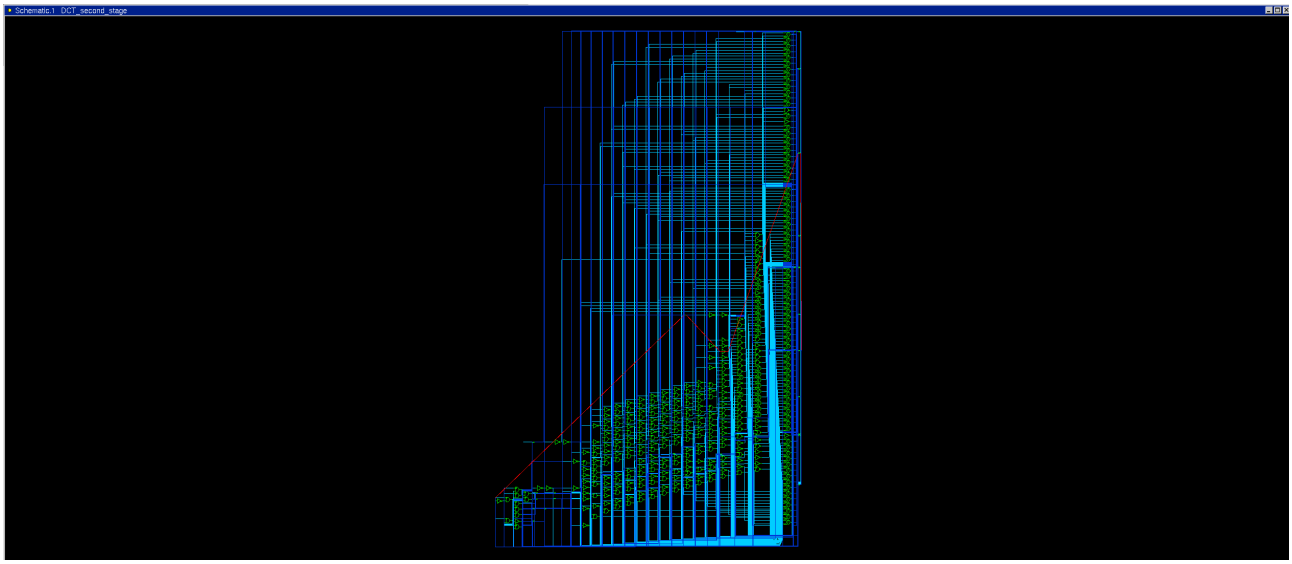
8 / 9 / tp mem first BW=9

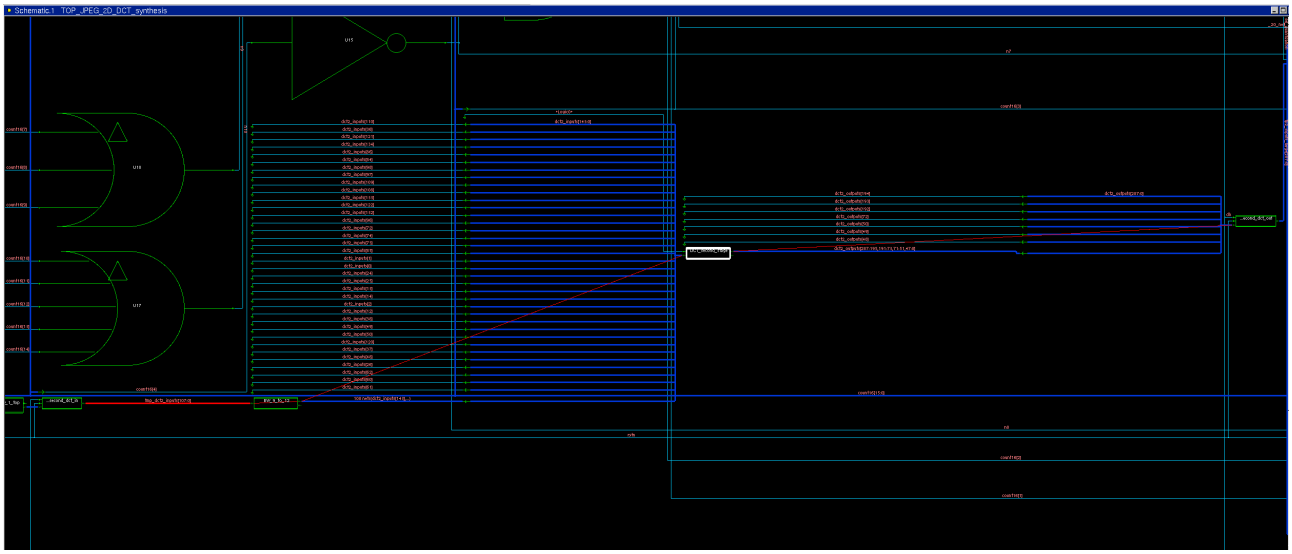


VLSI Project2

8 / 9 / sharing compressed + tp men 단일화

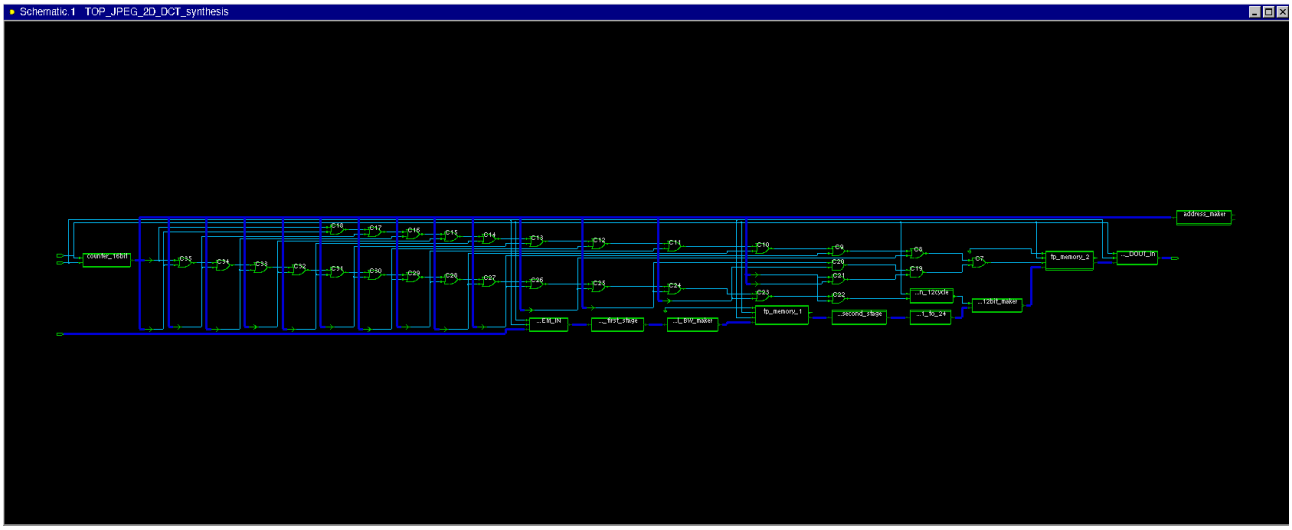


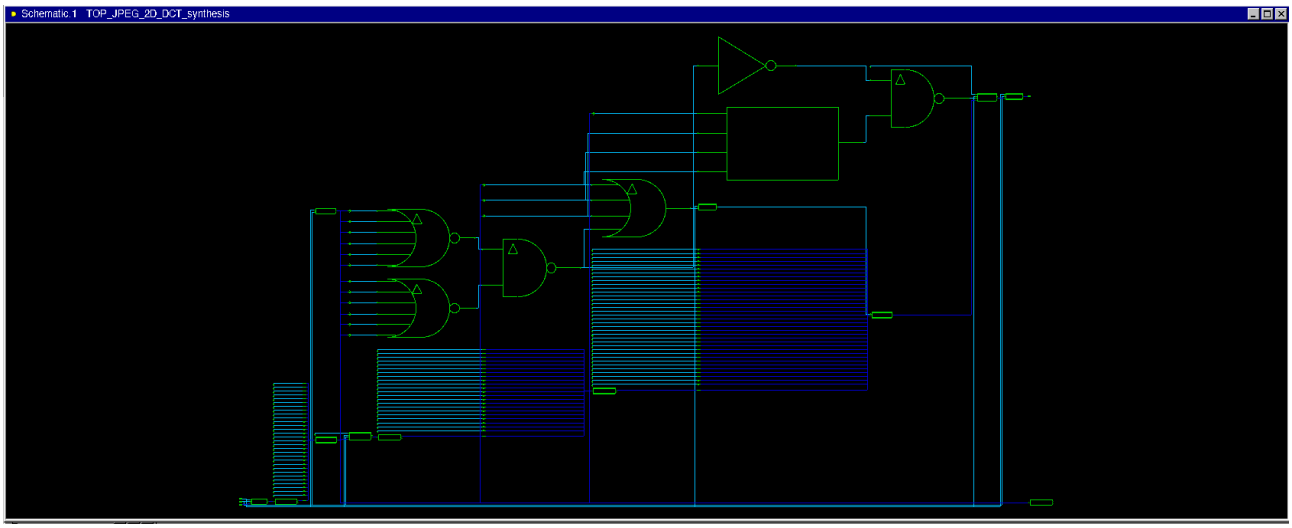


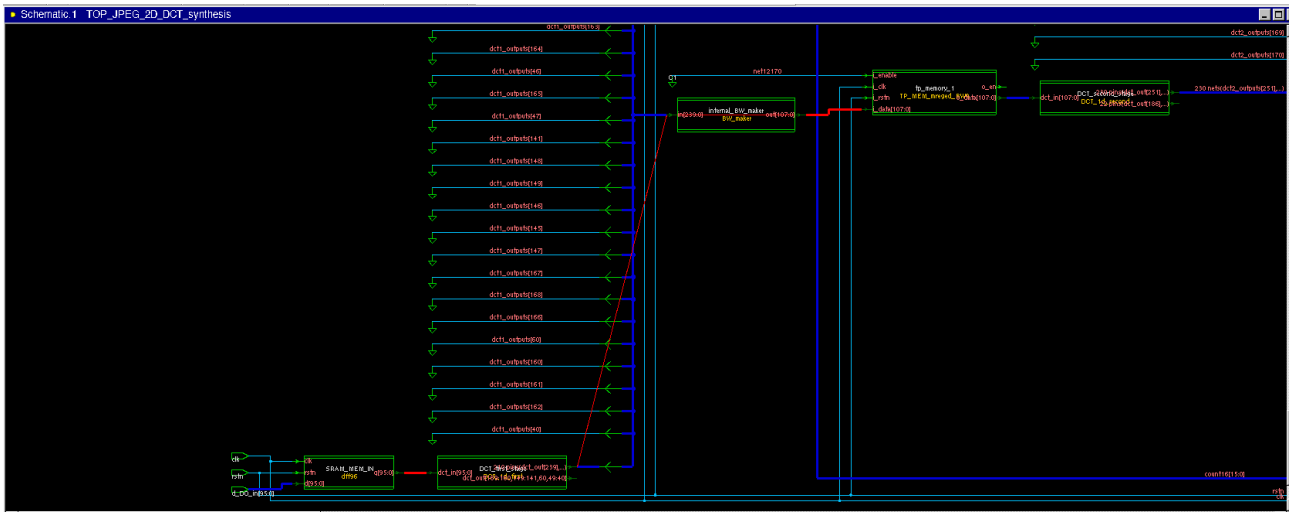


VLSI Project2

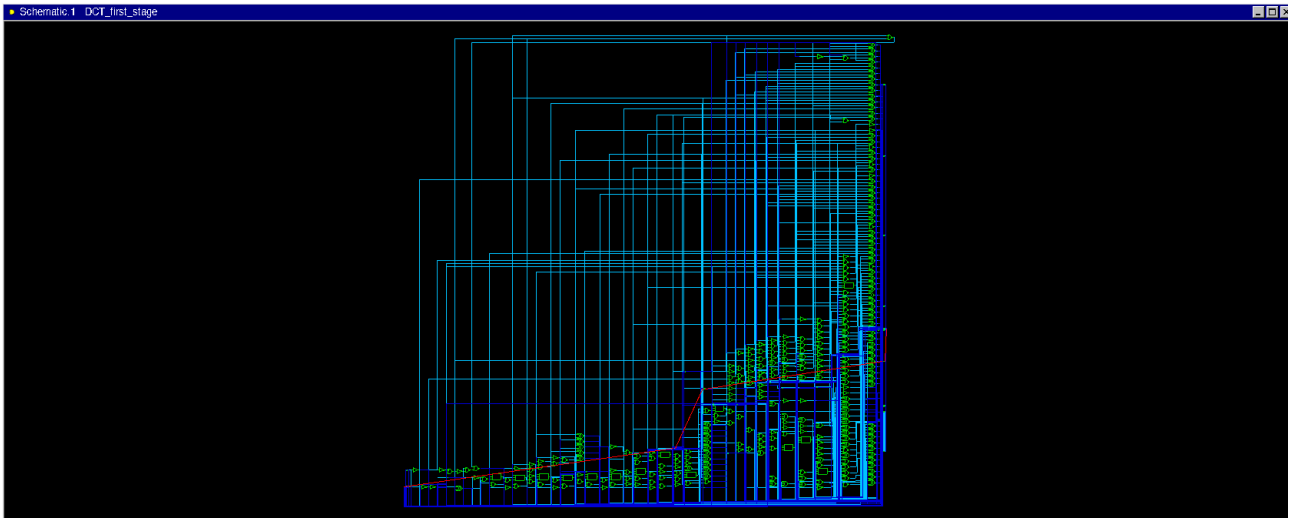
8 / 9 / dct2 BW optimize / dct1, 2 sign extension separation





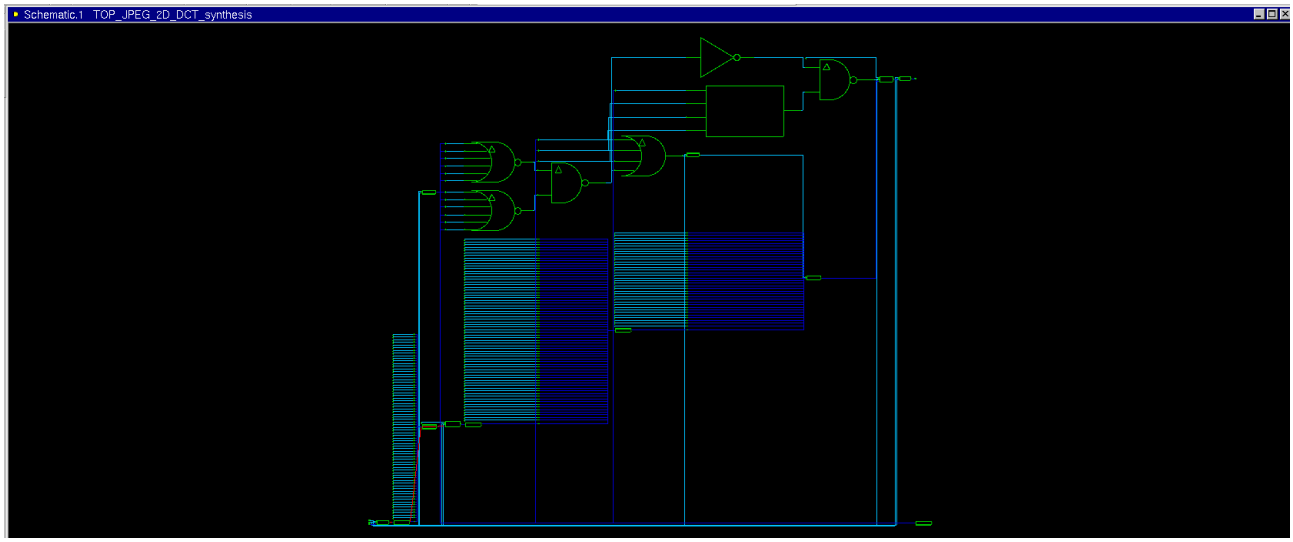


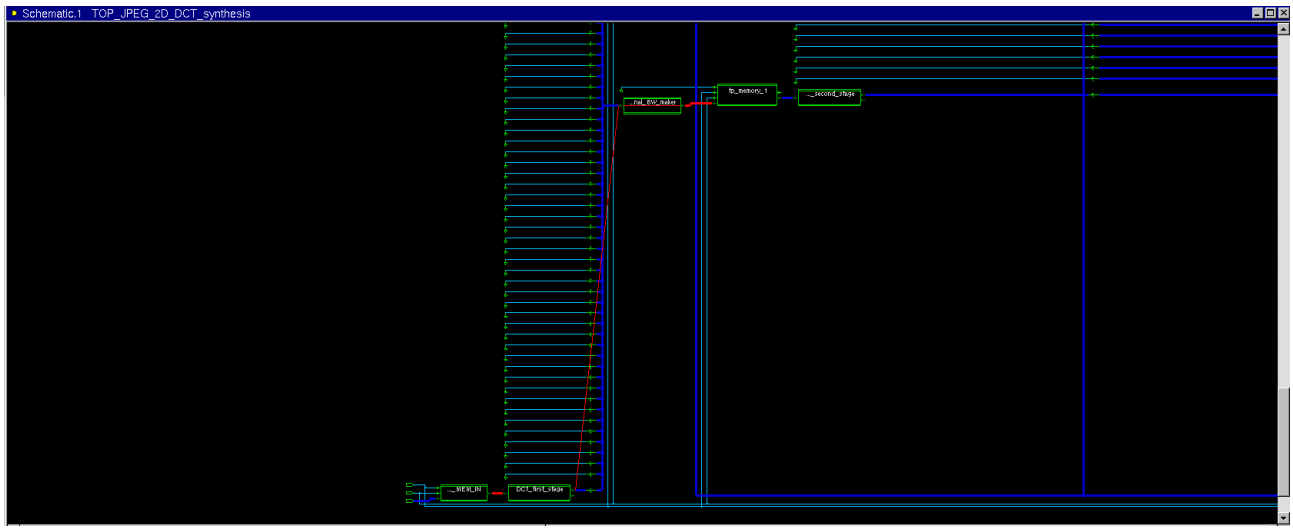
VLSI Project2

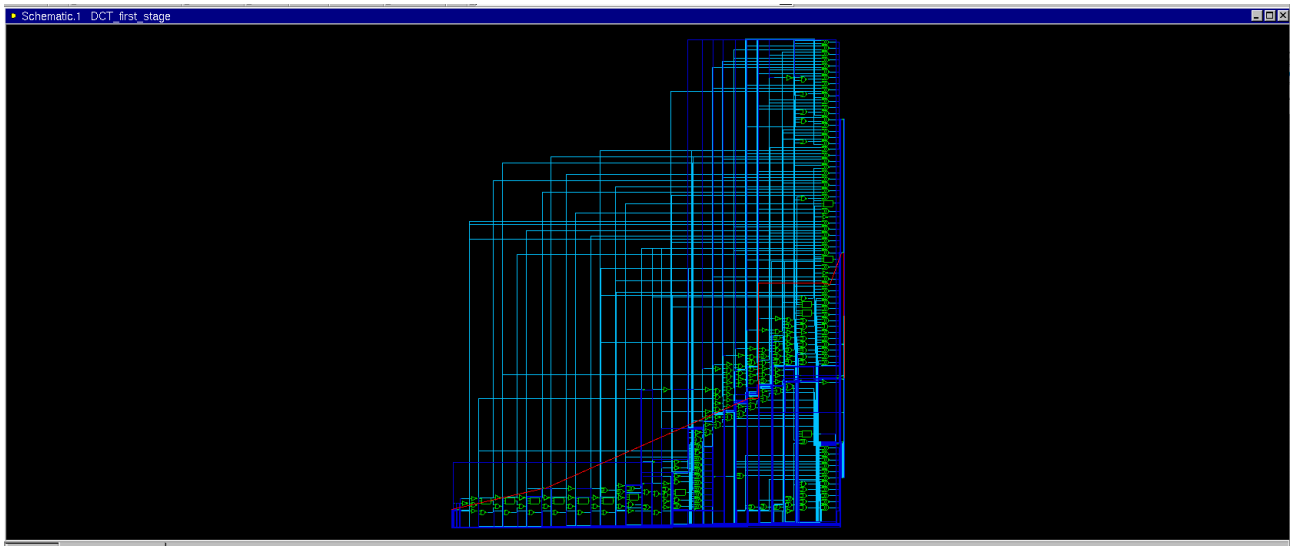


VLSI Project2

8 / 9 / high frequency computation removed







6. Processing log

Trial1. Overflow occurred

Trial2. Overflow fixed

Trial3. Coefficient 8 / Result_1D_DCT_quantization_bit = 9

VLSI Project2

Trial4. Coefficient Matrix symmetry property used

Trial5. Coefficient sharing

VLSI Project2

Trial6. TP_MEM 1st stage BW=9

Trial7. Coefficient compressed base on frequency basis

VLSI Project2

Trial8. Coefficient compressed + sharing renewal

Trial9. TP_MEM merged

VLSI Project2

Trial11. High frequency removed

Final hig frequency memory deleted

VLSI Project2

Original image #1
size : 480x480

Restored image #1
PSNR : 3.112170e+01

Original image #2
size : 480x480

Restored image #2
PSNR : 3.343087e+01

Original image #3
size : 480x480

Restored image #3
PSNR : 3.445786e+01

Original image #4
size : 480x480

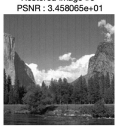
Restored image #4
PSNR : 3.438522e+01

Original image #5
size : 480x480

Restored image #5
PSNR : 3.032773e+01

Original image #6
size : 480x480

Restored image #6
PSNR : 3.458065e+01



Original image #7
size : 480x480

Restored image #7
PSNR : 3.287761e+01

Original image #8
size : 480x480

Restored image #8
PSNR : 3.497577e+01