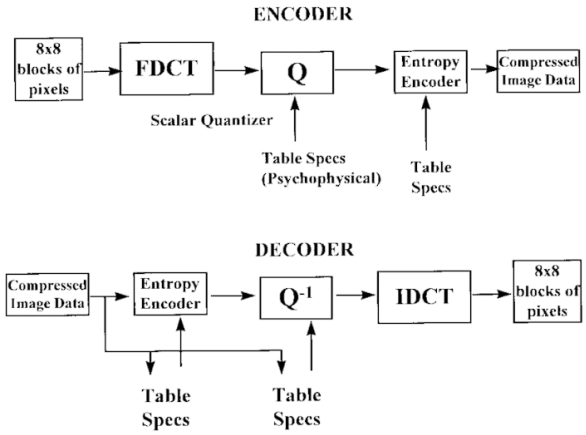
Project2 Report 2017160111 안진모

(1) Experimental Goal

이번 실습은 JPEG codec(이미지를 저장하기 위해 만들어진 압축 표준)을 회로로 구성하는 것이다. 이는 아래와 같은 구조를 갖는다.

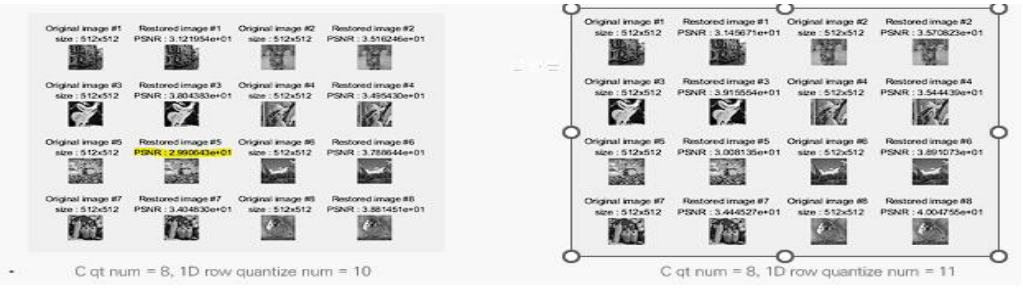


이 때, 각 픽셀에 대한 모든 데이터를 가지고 있는 원본 데이터에 대하여 색공간 변환(DCT) – 양자화(각 성분을 상수로 나누고 몫을 취함)- 엔트로피 부호화 가정을 거쳐 압축된 JPEG data가 만들어진다.

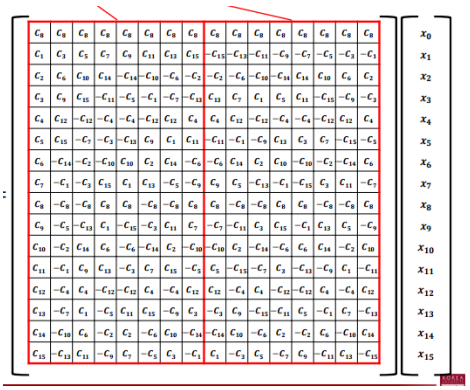
JPEG 이미지는 손실된 이미지이기에, 손실 정도를 가늠할 수 있는 factor가 필요하며, 대표적인 factor로 우리는 이번 실습에서 PSNR(신호 대 잡음의 비율)을 이용한다.

(2) Theoretical basis

실습에서 나는 C_quantization num =8로 설정했는데, 이는 hexadecimal로 보기에도 좋으며 실제로 bit가 커질수록 더욱 품질은 훌륭해지지만 계산량이 늘고, 계산량이 적으면서 적당히 쓸 수 있는 bit가 8bit으로 생각했기 때문이다. 그리고, 1D row quantization num =11로 설계하였는데, 이는 10으로 했을 때에는 PSNR이 30이 넘지 않는 image가 존재하기 때문이다 이를 첨부하면 아래와 같다.



우리는 Discrete cosine transform을 이용하며, 해당 image file을 압축하며, 이 때의 cosine transform의 수식은



좌측과 같다. Cosine fun은 even한 periodic한 function임을 알 수 있는데, 이를 대칭성으로 이용하면 쉽게 계산을 줄일 수가 있을 것으로 판단했다. 먼저 기본적인 대칭성만 이용하고, naive하게 설계한 DCT 모듈은 아래와 같다.

```
assign a0 = c8 * (sum0 + sum1 + sum2 + sum3 + sum4 + sum5 + sum6 + sum7);
assign a1 = (c1 * sub0) + (c3 * sub1) + (c5 * sub2) + (c7 * sub3) + (c9 * sub4) + (c11 * sub5) + (c13 * sub6) + (c15 * sub7);
assign a2 = (c2 * sum0) + (c6 * sum1) + (c10 * sum2) + (c14 * sum3) - (c14 * sum4) - (c10 * sum5) - (c6 * sum6) - (c2 * sum7);
assign a3 = (c3 * sub0) + (c9 * sub1) + (c15 * sub2) - (c11 * sub3) - (c5 * sub4) - (c1 * sub5) - (c7 * sub6) - (c13 * sub7);
assign a4 = (c4 * sum0) + (c12 * sum1) - (c12 * sum2) - (c4 * sum3) - (c4 * sum4) - (c12 * sum5) + (c12 * sum6) + (c4 * sum7);
assign a5 = (c5 * sub0) + (c15 * sub1) - (c7 * sub2) - (c3 * sub3) - (c13 * sub4) + (c9 * sub5) + (c1 * sub6) + (c11 * sub7);
assign a6 = (c6 * sum0) - (c14 * sum1) - (c2 * sum2) - (c10 * sum3) + (c10 * sum4) + (c2 * sum5) + (c14 * sum6) - (c6 * sum7);
assign a7 = (c7 * sub0) - (c11 * sub1) - (c3 * sub2) + (c15 * sub3) + (c1 * sub4) + (c13 * sub5) - (c5 * sub6) - (c9 * sub7);
assign a8 = (c8 * sum0) - (c8 * sum1) - (c8 * sum2) + (c8 * sum3) + (c8 * sum4) - (c8 * sum5) - (c8 * sum6) + (c8 * sum7);
assign a9 = (c9 * sub0) - (c5 * sub1) - (c13 * sub2) + (c1 * sub3) - (c15 * sub4) - (c3 * sub5) + (c11 * sub6) + (c7 * sub7);
assign a10 = (c10 * sum0) - (c2 * sum1) + (c14 * sum2) + (c6 * sum3) - (c6 * sum4) - (c14 * sum5) + (c2 * sum6) - (c10 * sum7);
assign a11 = (c11 * sub0) - (c1 * sub1) + (c9 * sub2) + (c13 * sub3) - (c3 * sub4) + (c7 * sub5) + (c15 * sub6) - (c5 * sub7);
assign a12 = (c12 * sum0) - (c4 * sum1) + (c4 * sum2) - (c12 * sum3) - (c12 * sum4) + (c4 * sum5) - (c4 * sum6) + (c12 * sum7);
assign a13 = (c13 * sub0) - (c7 * sub1) + (c1 * sub2) - (c5 * sub3) + (c11 * sub4) + (c15 * sub5) - (c9 * sub6) + (c3 * sub7);
assign a14 = (c14 * sum0) - (c10 * sum1) + (c6 * sum2) - (c2 * sum3) + (c2 * sum4) - (c6 * sum5) + (c10 * sum6) - (c14 * sum7);
assign a15 = (c15 * sub0) - (c13 * sub1) + (c11 * sub2) - (c9 * sub3) + (c7 * sub4) - (c5 * sub5) + (c3 * sub6) - (c1 * sub7);
```

이를 Cannonical signed digit method를 이용하면

```
assign o0 = ((ppp03+ppp12)<<5);
assign o1 = ((sub0<<5) +(sub0<<3)+(sub0<<2)+sub0) + ((sub1<<5) +(sub1<<3)+(sub0<<1)+sub0) + (sub2<<5)+(sub2<<3) + (sub3<<5)+(sub
assign o2 = (pm07<<5) +(pm07<<3)+(pm07<<2) + (pm16<<5)+(pm16<<2)+(pm16<<1) +  (pm25<<4) +(pm25<<3)+(pm25) + (pm34<<3)+pm34;
assign o3 =  (sub0<<5)+(sub0<<3)+(sub0<<1)+(sub0) + (sub1<<4) +(sub1<<3)+(sub1<<2) +sub1 + (sub2<<2)- ((sub3<<4)+(sub3<<2)+sub3)
assign o4 = (pmp03<<1)+(pmp03<<3)+(pmp03<<5) + (pmp12<<4)+(pmp12)  ;
assign o5 = ((sub0<<5)+(sub0<<3)) + (sub1<<2) - ((sub2<<5)+(sub2<<1)+sub2) - ((sub3<<5)+(sub3<<3)+(sub3<<1)+sub3)-((sub4<<3)+(sub
assign o6 = (pm07<<5) +(pm07<<2)+ (pm07<<1) -(pm16<<3)-(pm16)-( pm25<<5)-(pm25<<3)-(pm25<<2) -(pm34<<4)-(pm34<<3)-pm34 ;
assign o7 = ( (sub0<<5)+(sub0<<1)+sub0) - ((sub1<<4)+(sub1<<2)+sub1) - ((sub2<<5)+(sub2<<3)+(sub2<<1)+sub2)+ (sub3<<2) + ((sub4<<
assign o8 = ((ppp03-ppp12)<<5)  ;
assign o9 = ((sub0<<4)+(sub0<<3)+(sub0<<2)+sub0) - ((sub1<<5)+(sub1<<3)) - ((sub2<<4)+sub2) + ((sub3<<5)+(sub3<<3)+(sub3<<2)+sub
assign o10 = (pm07<<4)+(pm07<<3)+(pm07) - (pm16<<5)-(pm16<<3)-(pm16<<2) + ( pm25<<3)+pm25 + (pm34<<5)+(pm34<<2)+(pm34<<1);
assign o11 = ((sub0<<4)+(sub0<<2)+sub0) - ((sub1<<5)+(sub1<<3)+(sub1<<2)+sub1) + ((sub2<<4)+(sub2<<3)+(sub2<<2)+sub2) + ((sub3<<
assign o12 = (pmp03)+(pmp03<<4)-((pmp12<<1)+(pmp12<<3)+(pmp12<<5));
assign o13 = ((sub0<<4)+(sub0<<3)+sub0) - ((sub1<<5)+(sub1<<1)+sub1) +((sub2<<5)+(sub2<<3)+(sub2<<2)+sub2)- ((sub3<<5)+(sub3<<3))
assign o14 = (pm07<<3)+pm07 -  ((pm16<<4)+(pm16<<3)+pm16) + (pm25<<5)+(pm25<<2)+(pm25<<1) - ((pm34<<5)+(pm34<<3)+(pm34<<2));
assign o15 = (sub0<<2) - ((sub1<<3)+(sub1<<2)+sub1) + ((sub2<<4)+(sub2<<2)+sub2) - ((sub3<<4)+(sub3<<3)+(sub3<<2)+sub3) + ((sub4<
```

와 같이 설계할 수 있다.

또한, o0->o15로 갈수록 high한 frequency가 되는데, 이 때, 자연계 사진에는 low frequency가 많으므로 , col[15]를 계산하지 않고 0으로 처리하였다. (PSNR 30내에서 최소한의 면적을 얻기 위하여) 이를 첨부하면,

```
assign out = {(headfinder ? o0[19:8] : temp0), temp1, o2[17:6],o3[17:6], o4[17:6], o5[17:6], o6[17:6], o7[17:6], o8[17:6], o9[17:6], o10[17:6], o11[17:6], o12[17:6], o13[17:6], o14[17:6], 12'b0};
```

위와 같다.

따라서 우리의 module에서는 c value를 사용하지 않는다. C value를 첨부하면 아래와 같다.

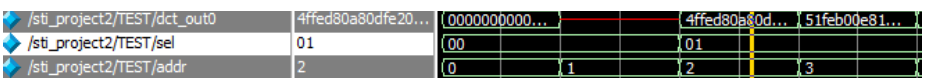
```
/*
assign c1 = 8'h2d; // 0010 1101
assign c2 = 8'h2c; // 0010 1100
assign c3 = 8'h2b; // 0010 1011
assign c4 = 8'h2a; //0010 1010
assign c5 = 8'h28; // 0010 1000
assign c6 = 8'h26; // 0010 0110
assign c7 = 8'h23; // 0010 0011
assign c8 = 8'h20; // 0010 0000
assign c9 = 8'h1d; // 0001 1101
assign c10 = 8'h19;
assign c11 = 8'h15; // 0001 0101
assign c12 = 8'h11; //0001 0001
assign c13 = 8'hd; // 0000 1101
assign c14 = 8'h9;
assign c15 = 8'h4; // 0000 0100
*/
```

다음은 DCT module에서 overflow(glitching block)를 처리하는 방법이다. Overflow를 처리하는 방법은 12bit signed integer 가 -2048 ~2047인데, 이 때 2047이 넘는 value는 2047로, -2048미만의 값은 -2048로 통일하여 적을 수 있었다.

```
module overflow(out, in);

output [11:0] out;
input [12:0] in;

assign out = (in[12:11] == 2'b01) ? 12'b0111_1111_1111 : ((in[12:11] == 2'b10) ? 12'b1000_0000_0000 : in[11:0]);

endmodule
```

```
overflow over0(temp0, o0[18:6]);
overflow over1(temp1, o1[18:6]);

assign out = {(headfinder ? o0[19:8] : temp0), temp1,
```

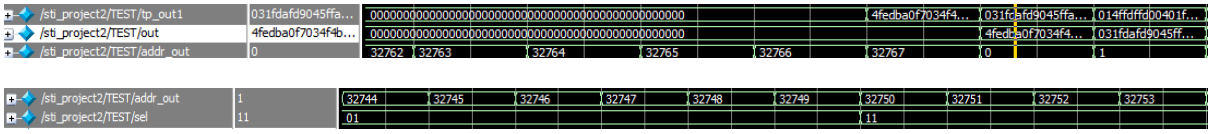
위의 head finder은 추후 Top module에서 head bit의 주소마다 1이되는 형태이다. 그 때 overflow value를 정의한다. DCT module은 끝이 났고, 이제 Top_memory_control unit을 보자. Memory control 부분은 비교적 간단하다.



Input part의 signal control이다. Dct_out0가 row부분 계산인데, 이 계산 결과에 맞추어 sel 즉 TP memory를 reset시켜주었다. 즉 계산 결과가 처음 나올 때, TP mem이 reset되고,

이후 저장과 읽기가 반복된다.

다음은 output part이다.

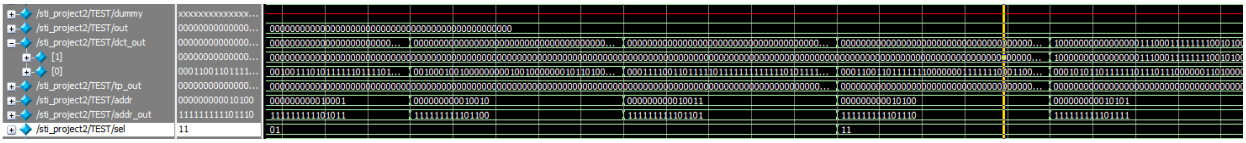


Output part역시, out이 나오는데 맞추어 주소를 0으로 설정하였고, 해당 값-18에서 TPMEM을 초기화 해주었다. 이는 16개의 matrix계산 + d_ff로 인한 2개의 cycle때문이다. (첨부한 모듈 확인)

```
always @(posedge clk) begin
    if(~reset) begin
        addr = 15'b0;
        sel = 2'b00;
    end

    if(reset) begin
        addr <= addr+ 1;

        if(addr[3:0] == 4'b0001) begin
            sel[0] = 1'b1;
        end
        if(addr[4:0] == 5'b10011) begin
            sel[1] = 1'b1;
        end

    end
end
assign addr_out =addr - 15'b0000_0000_0100_110;
```

주소 역시 앞서 말한게 맞추어지도록 설정한 것인데, 이 때 면적을 조금이라도 줄이기 위해 주소 부분의 addr_out을 wire로 할당하였다.

가장 주요했던 부분은 TP_memory 4개를 2개로 줄이는 작업이다. 이는 기존의 output을 보면,



TP_memory 2개가 있을 경우 교대로 reset되며 0이 output으로 나오게 된다. 이는 낭비를 유발하여, 두개를 하나로 합치는 작업을 진행하였다.

먼저, 기존의 counter는 4bit으로 16 cycle로 회로가 구성되게 되었는데, 이를 2x 하여 32cycle로 바꾼 후, first half cycle은 col, second half cycle은 row를 계산하게 하였다. 간단하게 case(counter[3:0])으로, MSB 즉 counter[4] =1 인 경우 col, 0인경우 row 부분이 data_out으로 할당되게 한 것이다. 이를 넣으면,

```
always@(*) begin
    if(counter[4]==1'b1) begin
        data_out <= col[index];
    end
    else    begin
        data_out <= array[index] ;
    end
end
```

위와 같다.

```
else    begin
  if(counter[4] == 1'b0) begin
        array[index] <= i_data;
    end
    else begin
        case(index)
        4'b0000 : begin
            array[0][16*BW-1:15*BW] <= i_data[16*BW-1:15*BW];
            array[1][16*BW-1:15*BW] <= i_data[15*BW-1:14*BW];
            array[2][16*BW-1:15*BW] <= i_data[14*BW-1:13*BW];
            array[3][16*BW-1:15*BW] <= i_data[13*BW-1:12*BW];
            array[4][16*BW-1:15*BW] <= i_data[12*BW-1:11*BW];
            array[5][16*BW-1:15*BW] <= i_data[11*BW-1:10*BW];
            array[6][16*BW-1:15*BW] <= i_data[10*BW-1:9*BW];
            array[7][16*BW-1:15*BW] <= i_data[9*BW-1:8*BW];
            array[8][16*BW-1:15*BW] <= i_data[8*BW-1:7*BW];
            array[9][16*BW-1:15*BW] <= i_data[7*BW-1:6*BW];
            array[10][16*BW-1:15*BW] <= i_data[6*BW-1:5*BW];
            array[11][16*BW-1:15*BW] <= i_data[5*BW-1:4*BW];
            array[12][16*BW-1:15*BW] <= i_data[4*BW-1:3*BW];
            array[13][16*BW-1:15*BW] <= i_data[3*BW-1:2*BW];
            array[14][16*BW-1:15*BW] <= i_data[2*BW-1:1*BW];
            array[15][16*BW-1:15*BW] <= i_data[1*BW-1:0*BW];
        end
```

이를 할당한 과정은 좌측과 같다.

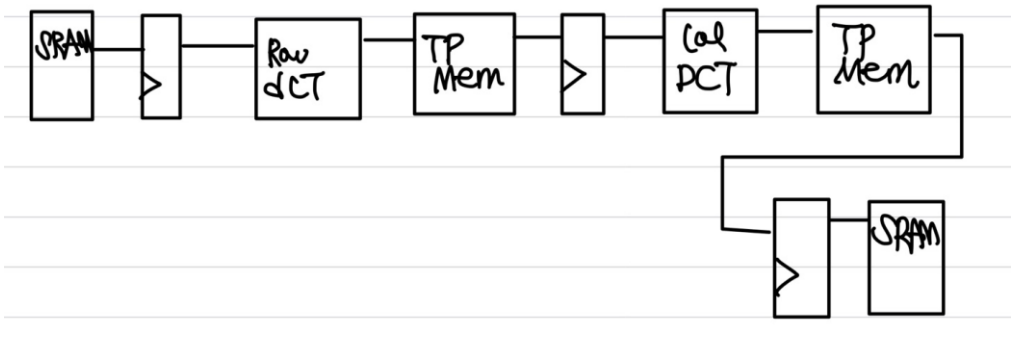
이 때, 8->11->12bit 순이므로 TP1은 11bit , TP2는 12bit의 bitwidth를 갖는다.

따라서 이제 내가 했던 작업을 종합하면, Symmetric ,Adder Tree , Simultaneous TPmem, 그리고 마지막으로 High frequency part를 소거한 것이 있다. 면적에 집중하여 생각하자.

```
Library(s) Used:

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

Number of ports:              352
Number of nets:              1863
Number of cells:               77
Number of references:          23

Combinational area:       2367573.214867
Noncombinational area:    2748106.803986
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          5115680.000000
Total area:                 undefined
```

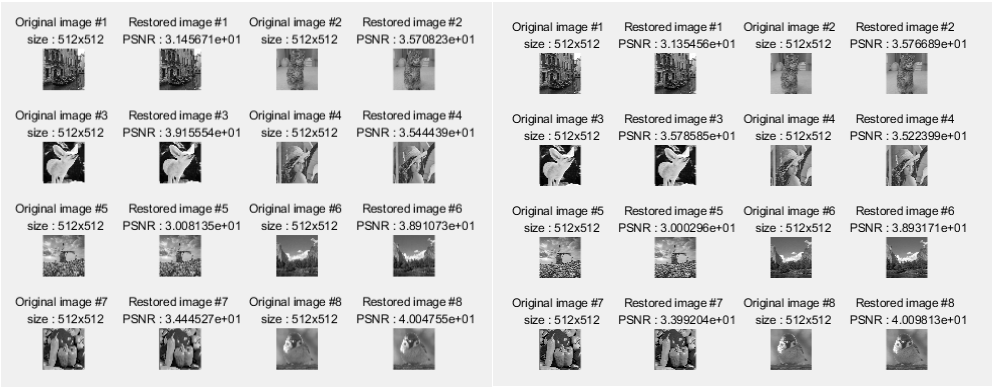
```
Library(s) Used:

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

Number of ports:              352
Number of nets:              1462
Number of cells:               80
Number of references:          22

Combinational area:       2256093.235569
Noncombinational area:    1318820.389664
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          3574913.750000
Total area:                 undefined
1
```

위를 통해 약 155만마이크로미터의 면적을 줄일 수 있었다.



이때의 Architecture는 위와 같다.

이제 PSNR result를 보자.



위의 좌측 사진은 full codec.mat이고 우측 파일은 DCT를 apply한 결과를 쓴 것이다. 보면 전체적으로 유의미하게 PSNR이 감소한 부분과, PSNR이 거의 차이가 없는 부분이 있는데 이를 보기 위하여,가장 크게 감소한 여우의 사진을 참고하자.

위의 다른 사진에 비하여 우리가 apply했던 사각형의 구조가 눈에 띄는 것을 알 수 있고, 이에 화질이 하락했음을 알 수 있었다. 또한 PSNR의 감소 요인으로는 glitching block (overflow) 때문이라고 판단하였다. 12bit이 over하는 값들을 12bit으로 값을 강제하니, 자연스럽게 화질이 하락 했다는 사실이다.

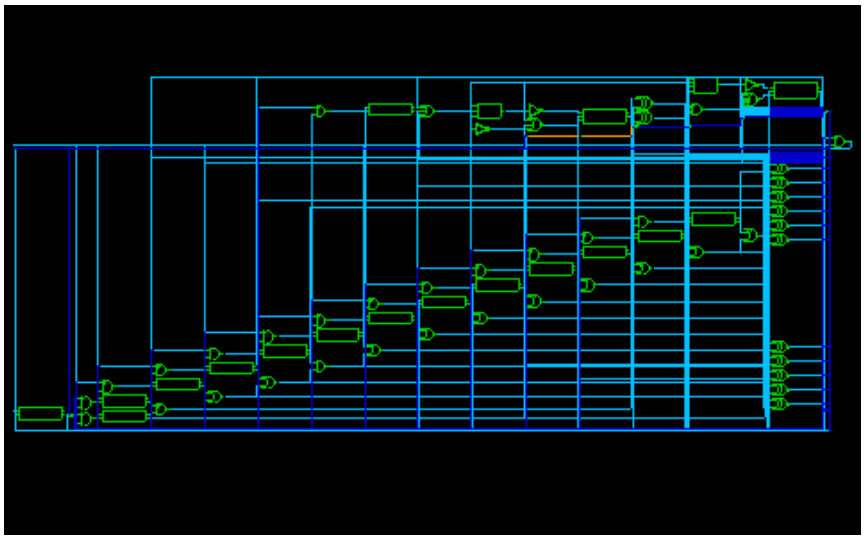
더 개선하는 방법을 생각했을 때, 사실 내가 구현하는 adder tree보다 high efforted in area상태의 design vision이 multiplying 을 더 작은 면적으로 구현할 수 있지 않을까 라는 생각도 들었다.

마지막으로 schematic과, timing report, power report를 첨부한다.

```
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 un.
    Leakage Power Units = 1pW


  Cell Internal Power  =  95.1374 mW   (76%)
  Net Switching Power  =  30.2459 mW   (24%)
                         ---------
Total Dynamic Power    = 125.3833 mW  (100%)

Cell Leakage Power     =   2.4967 mW
```

```
Startpoint: FF_BUT/out_reg[99]
          (rising edge-triggered flip-flop clocked by 321)
Endpoint: TP2/array_reg[15][130]
          (rising edge-triggered flip-flop clocked by 321)
Path Group: 321
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------
clock 321 (rise edge)                                    1.00       1.00
clock network delay (ideal)                              0.00       1.00
FF_BUT/out_reg[99]/CLK (dffcs2)                          0.00 #     1.00 r
FF_BUT/out_reg[99]/QN (dffcs2)                           0.25       1.25 f
FF_BUT/out_reg[99]/Q (dffcs2)                            0.17       1.42 r
FF_BUT/out[99] (D_FF176)                                 0.00       1.42 r
DCT2/in[99] (col_1D_DCT)                                 0.00       1.42 r
DCT2/sub_222/A[0] (col_1D_DCT_DW01_sub_13)               0.00       1.42 r
DCT2/sub_222/U155/Q (nor2s3)                             0.19       1.61 f
DCT2/sub_222/U111/Q (oai21s3)                            0.25       1.87 r
DCT2/sub_222/U151/Q (i1s4)                               0.12       1.98 f
DCT2/sub_222/U152/Q (i1s3)                               0.09       2.08 r
DCT2/sub_222/U156/Q (nnd2s1)                             0.14       2.22 f
DCT2/sub_222/U157/Q (nnd2s2)                             0.15       2.36 r
DCT2/sub_222/U170/Q (xnr2s3)                             0.45       2.82 f
DCT2/sub_222/DIFF[3] (col_1D_DCT_DW01_sub_13)            0.00       2.82 f
DCT2/U1854/Q (oai21s3)                                   0.26       3.08 r
DCT2/U1853/Q (nnd2s3)                                    0.21       3.28 f
DCT2/U1478/Q (xnr3s2)                                    0.56       3.84 r
DCT2/add_0_root_add_267_16/A[7] (col_1D_DCT_DW01_add_18)
                                                         0.00       3.84 r
DCT2/add_0_root_add_267_16/U224/Q (nor2s1)               0.25       4.09 f
```

```
                                                         0.18       8.90 r
DCT2/add_0_root_add_0_root_add_267_20/U1_9/OUTC (fadd1s3)
                                                         0.40       9.30 f
DCT2/add_0_root_add_0_root_add_267_20/U1_10/OUTC (fadd1s3)
                                                         0.38       9.68 f
DCT2/add_0_root_add_0_root_add_267_20/U1_11/OUTC (fadd1s3)
                                                         0.38      10.06 f
DCT2/add_0_root_add_0_root_add_267_20/U1_12/OUTC (fadd1s3)
                                                         0.38      10.44 f
DCT2/add_0_root_add_0_root_add_267_20/U1_13/OUTC (fadd1s3)
                                                         0.38      10.82 f
DCT2/add_0_root_add_0_root_add_267_20/U1_14/OUTC (fadd1s3)
                                                         0.38      11.20 f
DCT2/add_0_root_add_0_root_add_267_20/U1_15/OUTC (fadd1s3)
                                                         0.38      11.59 f
DCT2/add_0_root_add_0_root_add_267_20/U1_16/OUTS (fadd1s3)
                                                         0.78      12.36 r
DCT2/add_0_root_add_0_root_add_267_20/SUM[16] (col_1D_DCT_DW01_add_141)
                                                         0.00      12.36 r
DCT2/out[130] (col_1D_DCT)                               0.00      12.36 r
TP2/i_data[130] (TPmem_16x16v2)                          0.00      12.36 r
TP2/U692/Q (and2s2)                                      0.26      12.62 r
TP2/U261/Q (i1s5)                                        0.18      12.80 f
TP2/U259/Q (nnd2s2)                                      0.20      13.00 r
TP2/array_reg[15][130]/DIN (dffles1)                     0.00      13.00 r
data arrival time                                                  13.00

clock 321 (rise edge)                                   13.50      13.50
clock network delay (ideal)                              0.00      13.50
TP2/array_reg[15][130]/CLK (dffles1)                     0.00      13.50 r
library setup time                                      -0.50      13.00
data required time                                                 13.00
-----------------------------------------------------------------------
data required time                                                 13.00
data arrival time                                                 -13.00
-----------------------------------------------------------------------
slack (MET)                                                        0.00
```



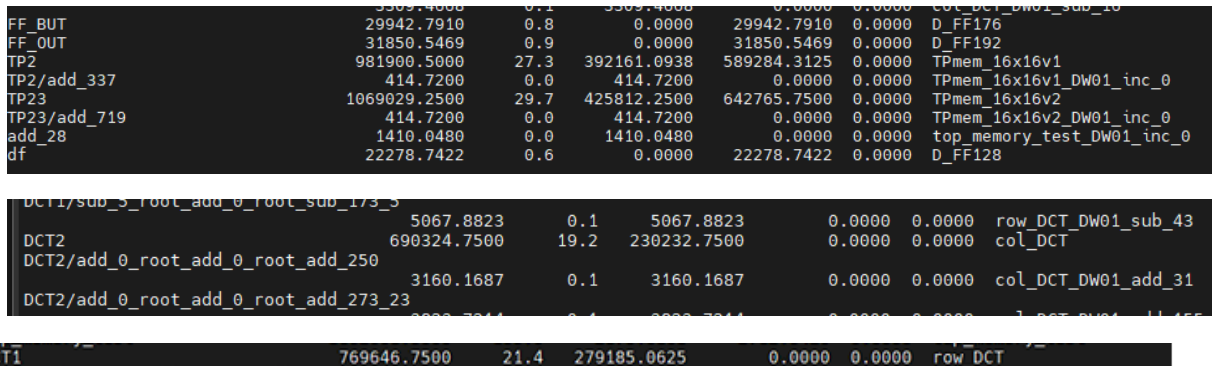
<Top module schematic>

위와 같고, 이를 모두 표로 정리하면,

	Top_DCT_control
Min period [ns]	12.5
Clock speed [MHz]	80

Data arrival time [ns]	13.0
Area $[\mu m^2]$	3574913
Total power [mW]	125.4

과 같다.



Area hierarchy를 적용하여 각각의 면적을 보이면,

	Area $[\mu m^2]$
Row_DCT	799645
Col_DCT	690324
TP1	981900
TP2	1069029

와 같다. 사실 아쉬운 것이, PSNR이 29만 됐었더라도 더 공격적으로 TPMEM이나 computation을 줄일 수 있었을 것 같다. 즉 c의 bit을 9bit까지 올리고 high한 frequency들을 다 vanishing하는 것이다. 지금의 30psnr에서는 그것보다, c가 7~8bit일 때 더 작은 면적 값을 가지리라 판단하였다. Critical path의 경우, 분석이 크게 유의미하지는 않다. 왜냐하면, 시작 bit과 끝나는 bit 사이에 많은 계산들이 우리가 사용하던 naïve adder 나 naïve multiplier처럼 1bit씩 올라가는 것이 아닌 adder tree형태로 진행되기 때문이다. 하지만 이 전체적 경향성을 확인하면, D_FF -> DCT1 -> DFF-> TPmem -> DCT2->DFF가 되는 것을 알 수 있었다.

마지막으로 Verification이다. 사실 위의 module은 memory controling이나 혹은 계산 값이 맞지 않으면 사진이 마치 아예 다른 사진인 것처럼 나오게된다. 즉, Vector verification 자체가 PSNR을

구하는 과정인 것이다. 이를 통해 우리는 JPEG codec processing 을 구현할 수 있었다.

(3) **Reference**

Verilog HDL -joseph Cavanagh

Verilog HDL 이론 -한양대학교 전자전기공학부

Quora- CPU clock speed vs power vs area

ResearchGate – Circuit Area vs frequency

고려대학교 전자전기공학부 – VLSI design project2