

HW3 Fixed Point DWT Filter Design

VLSI DSP HW3

Shun-Liang Yeh, NCHU Lab612

4/07/2023

INDEX

1. [Fixed-Point Discrete Wavelet Filter design](#)
2. [Synthesis Result](#)
3. [Acknowledgements](#)
4. [References](#)

I. Fixed Point Discrete Wavelet Filter Design

Problem

For a 3-level discrete wavelet transform (DWT) described in HW assignment, please conduct fixed point simulations to determine the DWT word length for the following items. Assume a floating-point version IDWT is used to reconstruct the image (same as the one given in Hw2), the PSNR (peak signal to noise ratio) of the reconstructed image should be no less than 50dB. Please use as small word length as possible to achieve this goal.

- The word length of the filter coefficients, all coefficients should have the same word length
 - The word length (integer and fractional) of the filter outputs at each level, i.e., level 1, 2 and 3 of DWT
 - Use synthesis tool and 90nm process technology to obtain the area of each multiplier, adder, and register
-
- Goal is to change the original design of HW2 into a fix point design, searching for the best fix point and fraction length.
 - Later find the area for the synthesized filter's multiplier, adder and registers for each DWT octave outputs.

Derivation steps

Discrete Wavelet transform Structure

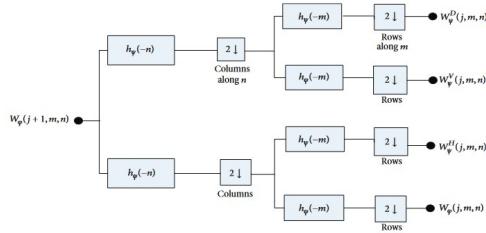


FIGURE 1: The discrete wavelet transform. Here, $h_p(-n)$, $h_p(-m)$, $h_p(-m)$, and $h_p(-m)$ are wavelet vectors. $2 \downarrow$ represents downsampling by 2.

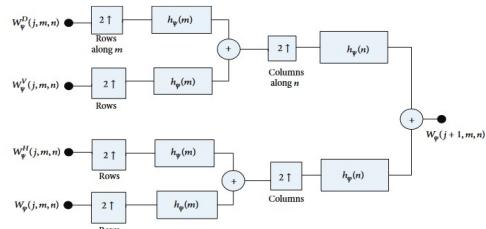
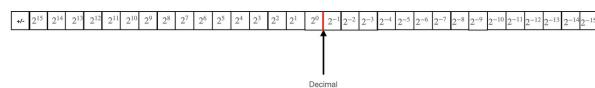


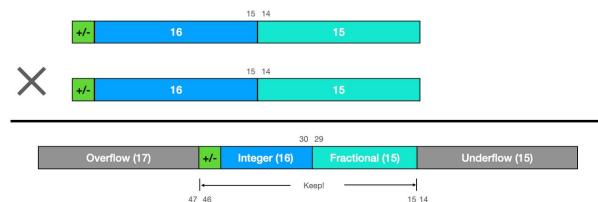
FIGURE 2: The inverse discrete wavelet transform. Here, $h_p(m)$, $h_p(m)$, $h_p(n)$, and $h_p(n)$ are wavelet vectors. $2 \uparrow$ signifies upsampling by 2.

- For the description of the DWT structure, look at VLSI DSP HW2

Fixed Point Representation & Arithmetic



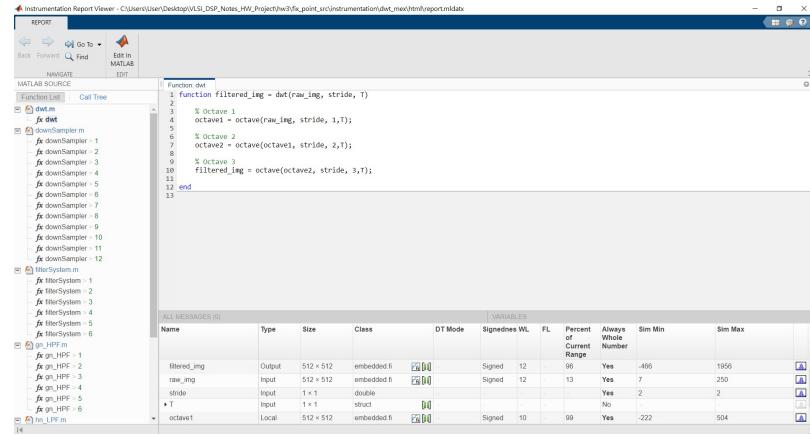
- Since floating point arithmetic units are expensive and difficult to generalised in HW design, fixed point representation for number is adopted.
- To make a fixed point number, all that we do is place this decimal somewhere in the middle. As an example, we'll decide that the decimal point is between bits 14 and 15. Mostly, this is just the way that we now think about this number and the CPU doesn't know the difference. However, there are a few special cases that we need to think about so that the CPU treats this number properly.



- Suppose that we multiply two 32-bit values. We will get an intermediate value that is 64-bits long. If each of these values were signed 16.15 fixed point (1 sign, 16 integer, 15 fraction), where is the decimal point in this intermediate variable? It's right above bit 29. If you multiply two fixed point values, each of which with 17 bits above the decimal (16 integer and 1 sign) and 15 bits below the decimal, then we end up with an intermediate value that has $2 \times 17 = 34$ bits above the decimal, and $2 \times 15 = 30$ bits below the decimal, for a total length of 64 bits.
- Now, which of these do we actually care about? We want the output of this multiplication to also be a signed 16.15 fixed point, so we care about the bits that immediately surround the decimal point.

Everything below that is underflow, which is ignored. And we better not have anything up here above the region of interest, because that means we've overflowed the representation. So, we care about these bits which hug the decimal. Multiply will not work if the number is too big. Thus we must check for the word length(WL) and Fraction Length(FL) if we want it to work.

Matlab Fix-Point Designer



- Matlab fixed point designer offers great tools for tracing fix-point length, value and precision over the function you specified, it also generate graphs for easy analysis for your design. Enabling one to efficiently convert their algorithm into fixed point design.

Code

FIR Filter(Symmetric extended)

```

● ● ●
1 % Filter
2 function yn = filterSystem(xn, wn, N, T, dwt_mode)
3     N = length(xn);
4     M = size(wn, 2);
5     L = fix(M / 2); % extend size // 2
6
7     if dwt_mode == 1
8         yn = zeros(1, N); % result
9         yn = cast(yn, 'like', xn);
10    else
11        yn = zeros(1, N); % result
12    end
13
14    x = [flip(xn(2:L + 1)), xn, flip(xn(N - L:N - 1))]; % Symmetric extension!
15
16    for i = 1:N
17        yn(i) = wn * x(1, i:i + M - 1)';
18    end
19
20 end

```

g(n) High pass filter and h(n) low pass filter

```

● ● ●
1 function filtered_img = gn_HPF(raw_img, horizontal, T, stage)
2     wn = [-0.064538882629 0.040689417609 0.418092273222 ...
3           -0.788485616406 ...
4           0.418092273222 0.040689417609 -0.064538882629];
5
6     wn = cast(wn, 'like', T.filter_coef);
7     dwt_mode = 1;
8
9     [h, w] = size(raw_img);
10
11    switch stage
12        case 1
13            output_type = T.lv1_output;
14        case 2
15            output_type = T.lv2_output;
16        case 3
17            output_type = T.lv3_output;
18    end
19
20    filtered_img = cast(raw_img, 'like', output_type);
21
22
23    if horizontal == 1
24        %Horizontal filtering
25        for i = 1:h
26            row_img = filtered_img(i,:);
27            filtered_img(i, :) = filterSystem(row_img, wn, w, T, dwt_mode);
28        end
29
30    else
31        % Vertical filtering
32        for i = 1:w
33            col_img = filtered_img(:,i);
34            filtered_img(:, i) = filterSystem(col_img', wn, h, T, dwt_mode);
35        end
36
37    end
38
39 end

```

```

● ● ●
1 function filtered_img = hn_LPF(raw_img, horizontal, T,stage)
2     wn = [0.037828455507 -0.023849465020 -0.110624404418 0.377402855613 ...
3           0.852698679009 ...
4           0.377402855613 -0.110624404418 -0.023849465020 0.037828455507];
5
6     wn = cast(wn, 'like', T.filter_coef);
7
8     dwt_mode = 1;
9
10    [h, w] = size(raw_img);
11
12    switch stage
13        case 1
14            output_type = T.lv1_output;
15        case 2
16            output_type = T.lv2_output;
17        case 3
18            output_type = T.lv3_output;
19    end
20
21    % Output types
22    filtered_img = cast(raw_img, 'like', output_type);
23
24    if horizontal == 1
25        %Horizontal filtering
26        for i = 1:h
27            row_img = filtered_img(i, :);
28
29            % row_img = cast(raw_img(i, :), 'like', T.filter_coef);
30            filtered_img(i, :) = filterSystem(row_img, wn, w, T, dwt_mode);
31        end
32
33    else
34        % Vertical filtering
35        for i = 1:w
36            col_img = filtered_img(:, i);
37
38            % col_img = cast(raw_img(:, i), 'like', T.filter_coef);
39            filtered_img(:, i) = filterSystem(col_img', wn, h, T, dwt_mode);
40        end
41
42    end
43
44 end

```

p(n) High pass filter and q(n) low pass filter

```
● ● ●
1 function filtered_img = pn_HPF(raw_img, horizontal)
2     wn = [-0.037828455507 -0.023849465020 0.110624404418 0.377402855613 ...
3           -0.852698679009 0.377402855613 0.110624404418 -0.023849465020 ...
4           -0.037828455507];
5     T = 0;
6     dwt_mode = 0;
7     [h, w] = size(raw_img);
8     filtered_img = raw_img;
9
10    if horizontal == 1
11        %Horizontal filtering
12        for i = 1:h
13            row_img = raw_img(i, :);
14            filtered_row = filterSystem(row_img, wn, w, T, dwt_mode);
15            filtered_img(i, :) = filtered_row;
16        end
17
18    else
19        % Vertical filtering
20        for i = 1:w
21            col_img = raw_img(:, i);
22            filtered_col = filterSystem(col_img', wn, h, T, dwt_mode);
23            filtered_img(:, i) = filtered_col;
24        end
25
26    end
27
28 end
```

```
● ● ●
1 function filtered_img = qn_LPF(raw_img, horizontal)
2     wn = [-0.064538882629 -0.040689417609 0.418092273222 0.788485616406 ...
3           0.418092273222 -0.040689417609 -0.064538882629];
4
5     T = 0;
6     dwt_mode = 0;
7     [h, w] = size(raw_img);
8     filtered_img = raw_img;
9
10    if horizontal == 1
11        %Horizontal filtering
12        for i = 1:h
13            row_img = raw_img(i, :);
14            filtered_row = filterSystem(row_img, wn, w, T, dwt_mode);
15            filtered_img(i, :) = filtered_row;
16        end
17
18    else
19        % Vertical filtering
20        for i = 1:w
21            col_img = raw_img(:, i);
22            filtered_col = filterSystem(col_img', wn, h, T, dwt_mode);
23            filtered_img(:, i) = filtered_col;
24        end
25
26    end
27
28 end
```

Down Sampler

```

● ● ●

1 function downSampledimg = downSampler(img, stride, odd, n, horizontal,T)
2     [h, w] = size(img);
3     partition = 2 ^ n;
4     downSampledimg = cast(zeros(h),'like',img);
5
6     if horizontal == 1
7
8         if odd == 0
9             %even for HPF
10            downSampledimg(1:h, 1:w / 2) = img(1:h, 2:stride:w);
11        else
12            %odd for LPF
13            downSampledimg(1:h, 1:w / 2) = img(1:h, 1:stride:w);
14        end
15
16    else
17
18        if odd == 0
19            %even for HPF
20            downSampledimg(1:h / 2, 1:w) = img(2:stride:h, 1:w);
21        else
22            %odd for LPF
23            downSampledimg(1:h / 2, 1:w) = img(1:stride:h, 1:w);
24        end
25
26    end
27
28 end

```

UpSampler

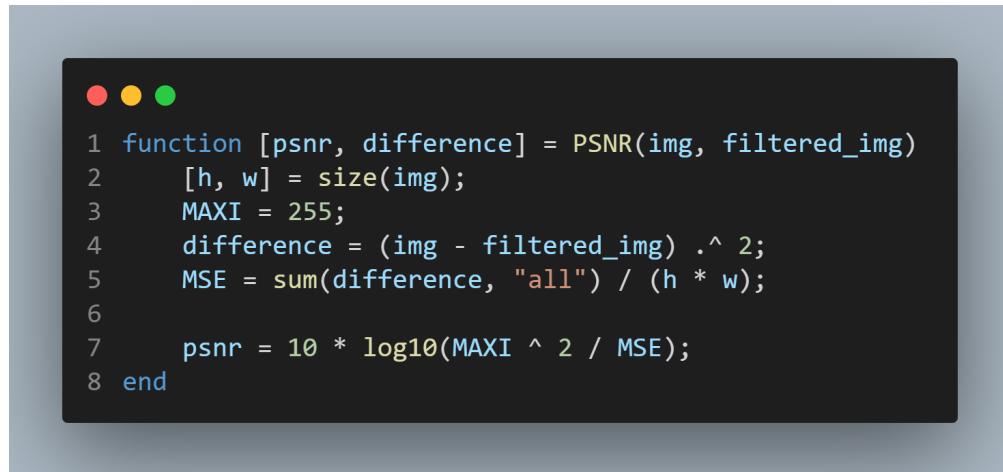
```

● ● ●

1 function upSampledimg = upSampler(img, stride, n, odd, horizontal)
2     % Interpolation algorithm should be adopted to further increase resolution after upSampling
3     % n means nth dwt_octave
4     [h, w] = size(img);
5     partition = 2 ^ n;
6     new_h = 2 * h;
7     new_w = 2 * w;
8
9     if horizontal == 1
10        upSampledimg = zeros(h);
11
12        if odd == 0
13            %even for HPF
14            upSampledimg(1:h, 2:stride:w) = img(1:h, 1:w / 2);
15        else
16            %odd for LPF
17            upSampledimg(1:h, 1:stride:w) = img(1:h, 1:w / 2);
18        end
19
20    else
21        upSampledimg = zeros(2 * h);
22        % Vertical
23        if odd == 0
24            %even for HPF
25            upSampledimg(2:stride:new_h, 1:new_w / 2) = img(1:h, 1:w);
26        else
27            %odd for LPF
28            upSampledimg(1:stride:new_h, 1:new_w / 2) = img(1:h, 1:w);
29        end
30
31    end
32
33 end

```

PSNR

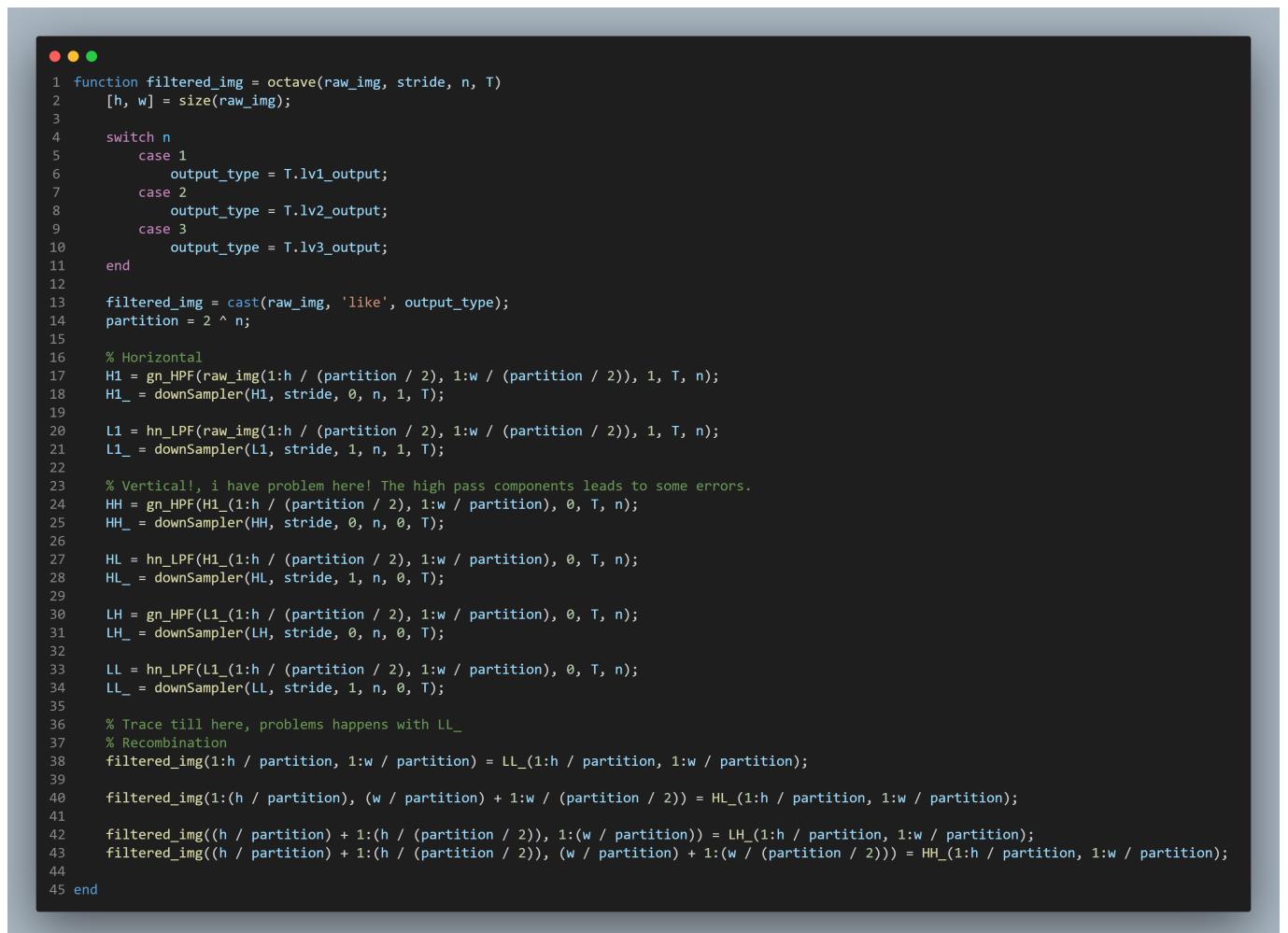


```

1 function [psnr, difference] = PSNR(img, filtered_img)
2     [h, w] = size(img);
3     MAXI = 255;
4     difference = (img - filtered_img) .^ 2;
5     MSE = sum(difference, "all") / (h * w);
6
7     psnr = 10 * log10(MAXI ^ 2 / MSE);
8 end

```

DWT octave



```

1 function filtered_img = octave(raw_img, stride, n, T)
2     [h, w] = size(raw_img);
3
4     switch n
5         case 1
6             output_type = T.lv1_output;
7         case 2
8             output_type = T.lv2_output;
9         case 3
10            output_type = T.lv3_output;
11    end
12
13    filtered_img = cast(raw_img, 'like', output_type);
14    partition = 2 ^ n;
15
16    % Horizontal
17    H1 = gn_HPF(raw_img(1:h / (partition / 2), 1:w / (partition / 2)), 1, T, n);
18    H1_ = downSampler(H1, stride, 0, n, 1, T);
19
20    L1 = hn_LPF(raw_img(1:h / (partition / 2), 1:w / (partition / 2)), 1, T, n);
21    L1_ = downSampler(L1, stride, 1, n, 1, T);
22
23    % Vertical!, i have problem here! The high pass components leads to some errors.
24    HH = gn_HPF(H1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
25    HH_ = downSampler(HH, stride, 0, n, 0, T);
26
27    HL = hn_LPF(H1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
28    HL_ = downSampler(HL, stride, 1, n, 0, T);
29
30    LH = gn_HPF(L1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
31    LH_ = downSampler(LH, stride, 0, n, 0, T);
32
33    LL = hn_LPF(L1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
34    LL_ = downSampler(LL, stride, 1, n, 0, T);
35
36    % Trace till here, problems happens with LL_
37    % Recombination
38    filtered_img(1:h / partition, 1:w / partition) = LL_(1:h / partition, 1:w / partition);
39
40    filtered_img(1:(h / partition), (w / partition) + 1:w / (partition / 2)) = HL_(1:h / partition, 1:w / partition);
41
42    filtered_img((h / partition) + 1:(h / (partition / 2)), 1:(w / partition)) = LH_(1:h / partition, 1:w / partition);
43    filtered_img((h / partition) + 1:(h / (partition / 2)), (w / partition) + 1:(w / (partition / 2))) = HH_(1:h / partition, 1:w / partition);
44
45 end

```

IDWT octave

```

1 function reconstructed_img = idwt_octave(compressed_img, stride, n)
2
3 [h, w] = size(compressed_img);
4 partition = 2 ^ n;
5 reconstructed_img = compressed_img;
6
7 % Extraction
8 LL_ = compressed_img(1:h / partition, 1:w / partition);
9 LH_ = compressed_img((h / partition) + 1:(h / (partition / 2)), 1:(w / partition));
10 HL_ = compressed_img(1:(h / partition), (w / partition) + 1:w / (partition / 2));
11 HH_ = compressed_img((h / partition) + 1:(h / (partition / 2)), (w / partition) + 1:(w / (partition / 2)));
12
13 % Vertical!
14 LL = upSampler(LL_, stride, n, 1, 0);
15 LL_test = LL(1:h / (partition / 2), 1:w / partition);
16 LL(1:h / (partition / 2), 1:w / (partition)) = qn_LPF(LL_test, 0);
17
18 % LH is wrong~ for the last 3 columns.
19 LH = upSampler(LH_, stride, n, 0, 0);
20 LH(1:h / (partition / 2), 1:w / (partition)) = pn_HPF(LH(1:h / (partition / 2), 1:w / partition), 0);
21
22 s0 = LL + LH;
23
24 HL = upSampler(HL_, stride, n, 1, 0);
25 HL(1:h / (partition / 2), 1:w / (partition)) = qn_LPF(HL(1:h / (partition / 2), 1:w / partition), 0);
26
27 HH = upSampler(HH_, stride, n, 0, 0);
28 HH(1:h / (partition / 2), 1:w / (partition)) = pn_HPF(HH(1:h / (partition / 2), 1:w / partition), 0);
29
30 % Minor difference in boundary of 160 columns and rows.
31 s1 = HL + HH;
32
33 % Horizontal!
34 s0_ = upSampler(s0, stride, n, 1, 1);
35 disp("Size of s0_");
36 disp(size(s0_));
37 s0_ = qn_LPF(s0_, 1);
38
39 s1_ = upSampler(s1, stride, n, 0, 1);
40 disp("Size of s1_");
41 disp(size(s1_));
42 s1_ = pn_HPF(s1_, 1);
43
44 s_temp = s0_ + s1_;
45
46 reconstructed_img(1:h / (partition / 2), 1:w / (partition / 2)) = s_temp;
47
48 end

```

Main driver

```

1 filter_coef.WL = 11;
2 filter_coef.FL = 10;
3
4 lv1_output.WL = 10;
5 lv1_output.FL = 0;
6
7 lv2_output.WL = 11;
8 lv2_output.FL = 0;
9
10 lv3_output.WL = 12;
11 lv3_output.FL = 0;
12
13 odd = 0;
14 horizontal = 1;
15 zero_padded = 0;
16
17 % Broadcasting data types.
18 T = filterDataType('fixed', filter_coef, lv1_output, lv2_output, lv3_output);
19
20 %=====
21 % RD image
22 %=====
23 img = double(imread('image.bmp'));
24 img_fixed = cast(img, 'like', T.lv3_output);
25
26 [h, w] = size(img_fixed);
27 n = 1;
28 stride = 2;
29
30 %=====
31 % DWT
32 %=====
33 buildInstrumentedMex dwt -o dwt_mex ...
34 --args {img_fixed, stride, T} -histogram
35
36 H1_ = dwt_mex(img_fixed, stride, T);
37
38 H1f = double(H1_);
39 figure(7);
40 imshow(H1f, []);
41 title('H1_');
42
43 %=====
44 % IDWT
45 %=====
46 restored_image = idwt(H1f, stride, zero_padded);
47
48 %=====
49 % PSNR
50 %=====
51 img = double(img);
52 disp("PSNR:");
53 [psnr, difference] = PSNR(img, restored_image);
54 fprintf('.%2f db\n', psnr);
55
56 % Verify results
57 showInstrumentationResults dwt_mex
58 % -proposeFL -defaultDT numerictype(1, 16)
59
60 % Code generation
61 codegen dwt ...
62 --args {img_fixed, stride, T} -config:lib -report

```

Data Type

```
1 function datatype = filterDataType(dt,filter_coef,lv1_output,lv2_output,lv3_output)
2     switch dt
3         case 'double'
4             datatype.filter_coef = double([]);
5             datatype.lv1_output = double([]);
6             datatype.lv2_output = double([]);
7             datatype.lv3_output = double([]);
8         case 'single'
9             datatype.filter_coef = single([]);
10            datatype.lv1_output = single([]);
11            datatype.lv2_output = single([]);
12            datatype.lv3_output = single([]);
13         case 'fixed'
14             datatype.filter_coef= fi([],1,filter_coef.WL,filter_coef.FL);
15             datatype.lv1_output = fi([],1,lv1_output.WL,lv1_output.FL);
16             datatype.lv2_output = fi([],1,lv2_output.WL,lv2_output.FL);
17             datatype.lv3_output = fi([],1,lv3_output.WL,lv3_output.FL);
18         case 'scaled'
19             % This runs computation in double but store data in fix point, a debugging type
20             % To check how far your range for your fix-point, want to get all the possible range.
21             datatype.filter_coef    = fi([],1,filter_coef.WL,filter_coef.FL,'DataType','ScaledDouble');
22             datatype.lv1_output     = fi([],1,lv1_output.WL,lv1_output.FL,'DataType','ScaledDouble');
23             datatype.lv2_output     = fi([],1,lv2_output.WL,lv2_output.FL,'DataType','ScaledDouble');
24             datatype.lv3_output     = fi([],1,lv3_output.WL,lv3_output.FL,'DataType','ScaledDouble');
25     end
26 end
```

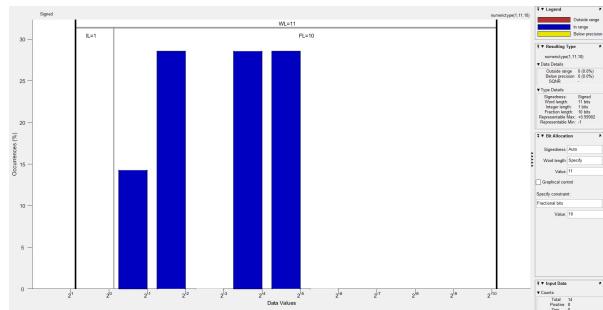
Result

Fixed Point statistical report

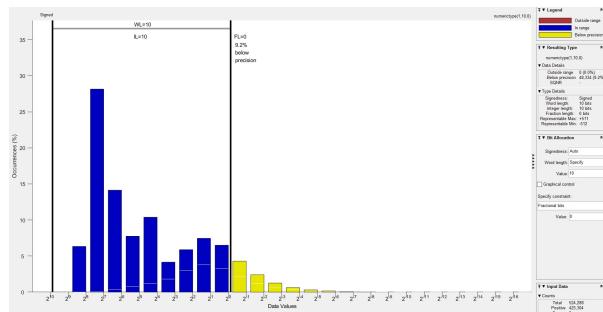
Fixed Point Width length and fraction length

Objects	WL	FL
Filter_Coef	11	10
lv1 output	10	0
lv2 output	11	0
lv3 output	12	0

Filter Coefficients

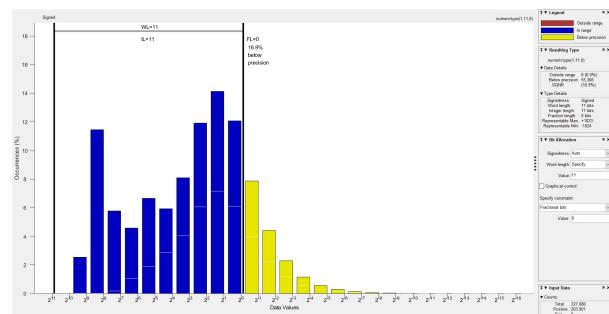


Level 1 output

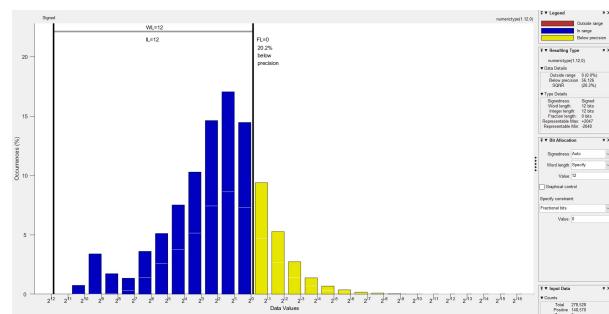


- In the fixed point designer, it tells you the range for your calculation also suggest you the length of the fix-point and fraction length.

Level 2 output



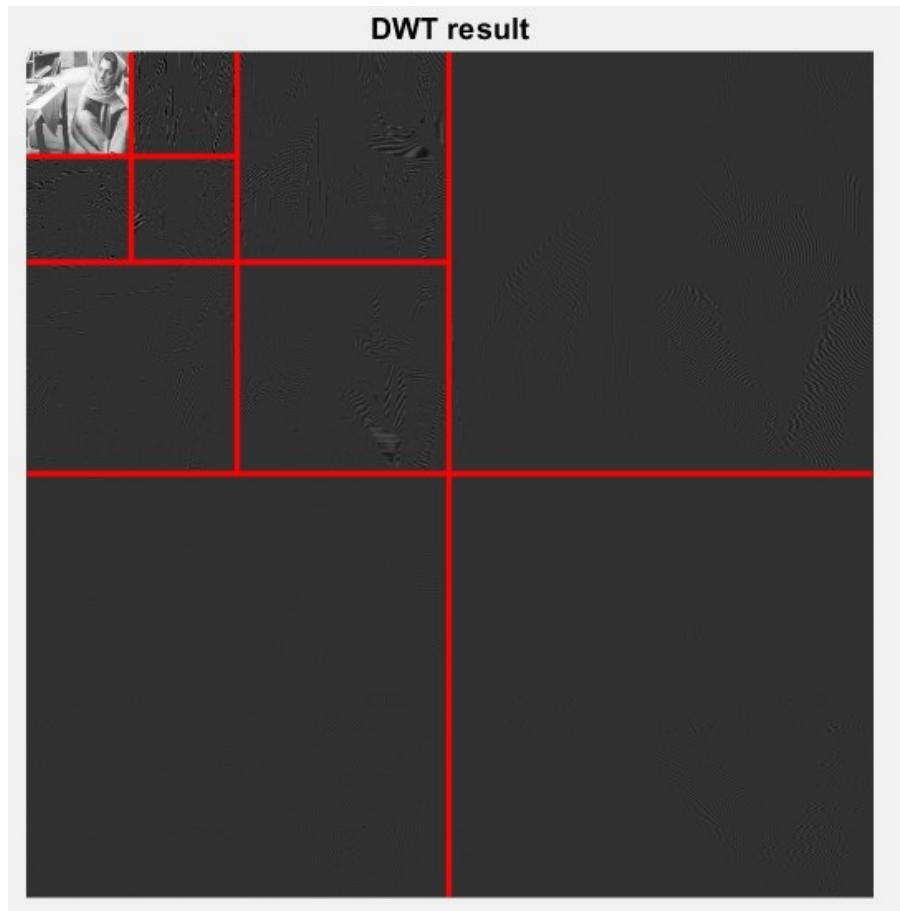
Level 3 output



Original image



DWT result



Restored image



PSNR

PSNR:
51.04 db

- The result is close to 50db, tuning any further result in a distortion or making the restoration lower than 50db.

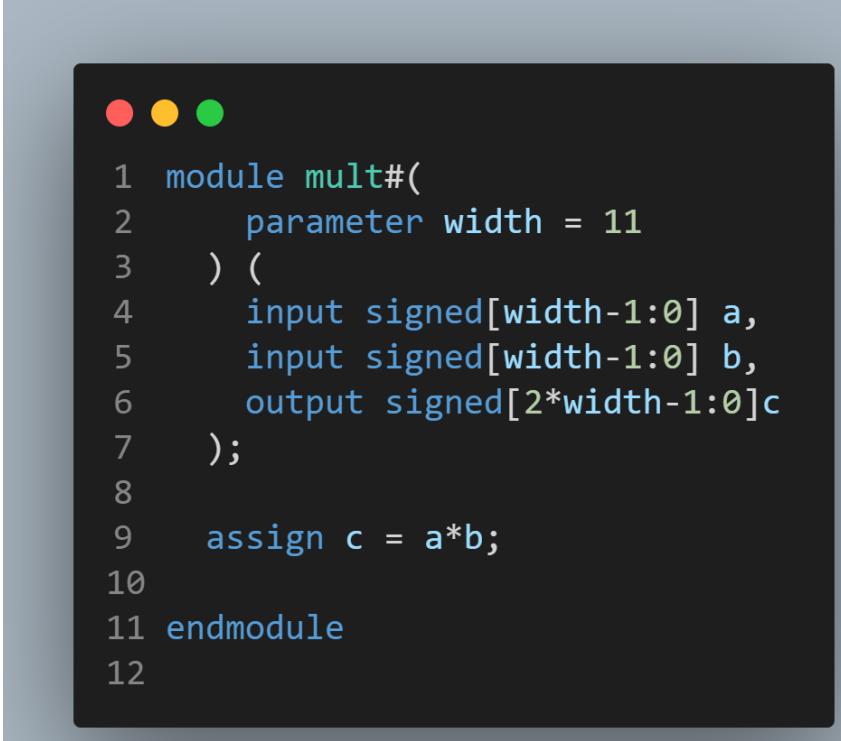
Note

1. Fix point designer is a powerful tool for analysis.
2. High frequency components are important for image reconstruction, thus the length for the filter coefficients are important for high-frequency restoration.
3. The low frequency components are the most important for main image reconstruction.

II. Synthesis Result

- Synthesize the word length for adders and filter's registers results according to the matlab simulation into gate-level circuit.

Multiplexer



```

1 module mult#(
2     parameter width = 11
3 ) (
4     input signed[width-1:0] a,
5     input signed[width-1:0] b,
6     output signed[2*width-1:0]c
7 );
8
9     assign c = a*b;
10
11 endmodule
12

```

```

*****
Report : area
Design : mult
Version: T-2022.03
Date   : Fri Apr  7 16:17:56 2023
*****


Library(s) Used:

    slow (File: /cad/CBDK/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)

Number of ports:                      88
Number of nets:                       410
Number of cells:                      281
Number of combinational cells:        280
Number of sequential cells:           0
Number of macros/black boxes:         0
Number of buf/inv:                   64
Number of references:                 1

Combinational area:                  4078.852236
Buf/Inv area:                      478.666793
Noncombinational area:              0.000000
Macro/Black Box area:               0.000000
Net Interconnect area:             33040.164856

Total cell area:                   4078.852236
Total area:                        37119.017092

```

Adder

```
 1 module adder#(
 2     parameter width = 10
 3 ) (
 4     input signed[width-1:0] a,
 5     input signed[width-1:0] b,
 6     output signed[width:0]c
 7 );
 8
 9     assign c = a*b;
10
11 endmodule
```

```
*****
Report : area
Design : adder
Version: T-2022.03
Date  : Fri Apr  7 16:08:10 2023
*****  
  
Library(s) Used:  
  
    slow (File: /cad/CBDK/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)  
  
Number of ports:                      71  
Number of nets:                       231  
Number of cells:                      156  
Number of combinational cells:        155  
Number of sequential cells:           0  
Number of macros/black boxes:         0  
Number of buf/inv:                   33  
Number of references:                 1  
  
Combinational area:                  2002.932017  
Buf/Inv area:                      244.425596  
Noncombinational area:               0.000000  
Macro/Black Box area:                0.000000  
Net Interconnect area:              17826.755920  
  
Total cell area:                    2002.932017  
Total area:                         19829.687937
```

Registers

```

1 module register#(
2     parameter width = 10
3 ) (
4     input clk,
5     input rst,
6     input signed [width-1:0] register_wr,
7     output reg signed [width:0] register_ff
8 );
9
10    always@(posedge clk)
11    begin
12        if(rst)
13        begin
14            register_ff <= 0;
15        end
16        else
17        begin
18            register_ff <= register_wr;
19        end
20    end
21 endmodule
22

```

|v1

```

*****
Report : area
Design : register
Version: T-2022.03
Date   : Fri Apr 7 16:10:01 2023
*****


Library(s) Used:

    slow (File: /cad/CBDK/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)

Number of ports:                      23
Number of nets:                       44
Number of cells:                      32
Number of combinational cells:        21
Number of sequential cells:           11
Number of macros/black boxes:         0
Number of buf/inv:                    11
Number of references:                 3

Combinational area:                  254.610009
Buf/Inv area:                      186.714010
Noncombinational area:              280.071005
Macro/Black Box area:                0.000000
Net Interconnect area:              2986.681671

Total cell area:                    534.681014
Total area:                         3521.362685

```

lv2

```
*****
Report : area
Design : register
Version: T-2022.03
Date   : Fri Apr  7 16:15:17 2023
*****
```

Library(s) Used:

```
slow (File: /cad/CBDK/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)
```

Number of ports:	25
Number of nets:	48
Number of cells:	35
Number of combinational cells:	23
Number of sequential cells:	12
Number of macros/black boxes:	0
Number of buf/inv:	12
Number of references:	3
Combinational area:	278.373610
Buf/Inv area:	203.688011
Noncombinational area:	305.532005
Macro/Black Box area:	0.000000
Net Interconnect area:	3266.683075
Total cell area:	583.905615
Total area:	3850.588690

lv3

```
*****
Report : area
Design : register
Version: T-2022.03
Date   : Fri Apr  7 16:16:31 2023
*****
```

Library(s) Used:

```
slow (File: /cad/CBDK/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)
```

Number of ports:	27
Number of nets:	52
Number of cells:	38
Number of combinational cells:	25
Number of sequential cells:	13
Number of macros/black boxes:	0
Number of buf/inv:	13
Number of references:	3
Combinational area:	302.137211
Buf/Inv area:	220.662012
Noncombinational area:	330.993006
Macro/Black Box area:	0.000000
Net Interconnect area:	3546.684418
Total cell area:	633.130217
Total area:	4179.814634

III. Acknowledgement

- I would like to acknowledge EECS undergraduate Chun-Wei Su and EECS undergraduate Guan-Ting Du for their assistance throughout the research, design and implementation of this homework. Without them, this HW cannot become a reality.

IV. References

- [1] [ECE 4760, Adams/Land, Fixed Point arithmetic , Spring 2021](#)
- [2] [Best Practices for Converting MATLAB Code to Fixed Point Using Fixed-Point Designer](#)
- [3] [EE123 Digital Signal Processing, SP'16 L12 - Discrete Wavelet Transform](#)
- [4] [Easy Introduction to Wavelets, by Simon Xu](#)
- [5] [Image Denoising Based on Improved Wavelet Threshold Function for Wireless Camera Networks and Transmissions,Sep 2015, Reserach Gate,Xiaoyu Wang Xiaoxu Ou Bo-Wei Chen Mucheol Kim](#)