HW3 Fixed Point DWT Filter Design

VLSI DSP HW3

Shun-Liang Yeh, NCHU Lab612

4/07/2023

Github Src Code

INDEX

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

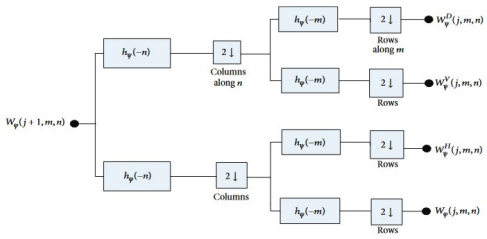


FIGURE 1: The discrete wavelet transform. Here, $h_\varphi(-n)$, $h_\psi(-n)$, $h_\psi(-m)$, and $h_\varphi(-m)$ are wavelet vectors. 2 ↓ represents downsampling by 2.

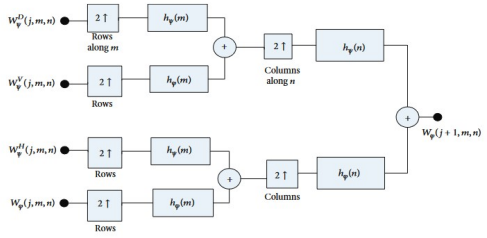
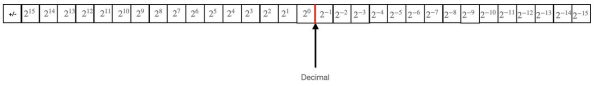


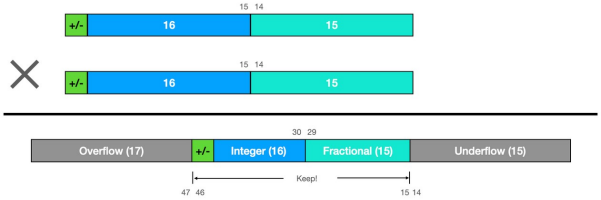
FIGURE 2: The inverse discrete wavelet transform. Here, $h_\psi(m)$, $h_\varphi(m)$, $h_\psi(n)$, and $h_\varphi(n)$ are wavelet vectors. 2 ↑ signifies upsampling by 2.

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

Fixed Point Representation & Arithmetic



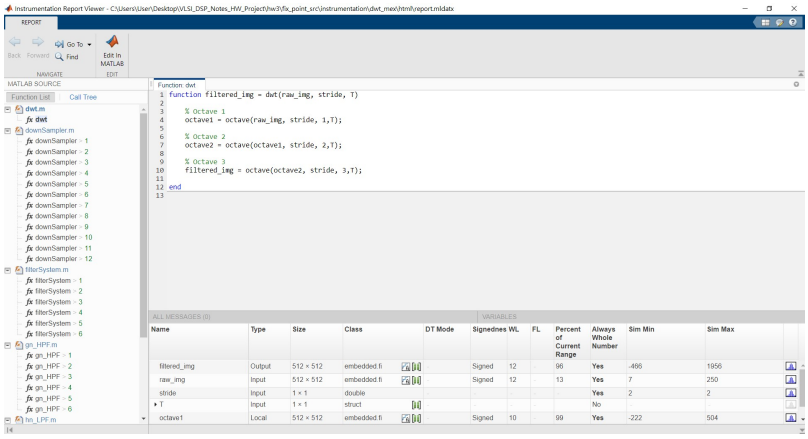
- Since floating point arithmetic units are expensive and difficult to generalised in HW design, fixed point representation for number is adopted.
- To make a fixed point number, all that we do is place this decimal somewhere in the middle. As an example, we'll decide that the decimal point is between 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 2x17 = 34 bits above the decimal, and 2x15 = 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)

```
% Filter
function yn = filterSystem(xn, wn, N, T, dwt_mode)
    N = length(xn);
    M = size(wn, 2);
    L = fix(M / 2); % extend size // 2

    if dwt_mode == 1
        yn = zeros(1, N); % result
        yn = cast(yn, 'like', xn);
    else
        yn = zeros(1, N); % result
    end

    x = [flip(xn(2:L + 1)), xn, flip(xn(N - L:N - 1))]; % Symmetric extension!

    for i = 1:N
        yn(i) = wn * x(1, i:i + M - 1)';
    end

end
```

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

```
function filtered_img = gn_HPF(raw_img, horizontal, T, stage)
    wn = [-0.064538882629 0.040689417609 0.418092273222 ...
              -0.788485616406 ...
              0.418092273222 0.040689417609 -0.064538882629];

    wn = cast(wn, 'like', T.filter_coef);
    dwt_mode = 1;

    [h, w] = size(raw_img);

    switch stage
        case 1
            output_type = T.lv1_output;
        case 2
            output_type = T.lv2_output;
        case 3
            output_type = T.lv3_output;
    end

    filtered_img = cast(raw_img, 'like', output_type);


    if horizontal == 1
        %Horizontal filtering
        for i = 1:h
            row_img = filtered_img(i,:);
            filtered_img(i, :) = filterSystem(row_img, wn, w, T, dwt_mode);
        end

    else
        % Vertical filtering
        for i = 1:w
            col_img = filtered_img(:,i);
            filtered_img(:, i) = filterSystem(col_img', wn, h, T, dwt_mode);
        end

    end

end
```

```
function filtered_img = hn_LPF(raw_img, horizontal, T,stage)
    wn = [0.037828455507 -0.023849465020 -0.110624404418 0.377402855613 ...
              0.852698679009 ...
              0.377402855613 -0.110624404418 -0.023849465020 0.037828455507];

    wn = cast(wn, 'like', T.filter_coef);

    dwt_mode = 1;

    [h, w] = size(raw_img);

    switch stage
        case 1
            output_type = T.lv1_output;
        case 2
            output_type = T.lv2_output;
        case 3
            output_type = T.lv3_output;
    end

    % Output types
    filtered_img = cast(raw_img, 'like', output_type);

    if horizontal == 1
        %Horizontal filtering
        for i = 1:h
            row_img = filtered_img(i, :);

            % row_img = cast(raw_img(i, :), 'like', T.filter_coef);
            filtered_img(i, :) = filterSystem(row_img, wn, w, T, dwt_mode);
        end

    else
        % Vertical filtering
        for i = 1:w
            col_img = filtered_img(:, i);

            % col_img = cast(raw_img(:, i), 'like', T.filter_coef);
            filtered_img(:, i) = filterSystem(col_img', wn, h, T, dwt_mode);
        end

    end

end
```

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

```
function filtered_img = pn_HPF(raw_img, horizontal)
    wn = [-0.037828455507 -0.023849465020 0.110624404418 0.377402855613 ...
              -0.852698679009 0.377402855613 0.110624404418 -0.023849465020 ...
              -0.037828455507];
    T = 0;
    dwt_mode = 0;
    [h, w] = size(raw_img);
    filtered_img = raw_img;

    if horizontal == 1
        %Horizontal filtering
        for i = 1:h
            row_img = raw_img(i, :);
            filtered_row = filterSystem(row_img, wn, w,T,dwt_mode);
            filtered_img(i, :) = filtered_row;
        end

    else
        % Vertical filtering
        for i = 1:w
            col_img = raw_img(:, i);
            filtered_col = filterSystem(col_img', wn, h,T,dwt_mode);
            filtered_img(:, i) = filtered_col;
        end

    end

end
```

```
function filtered_img = qn_LPF(raw_img, horizontal)
    wn = [-0.064538882629 -0.040689417609 0.418092273222 0.788485616406 ...
              0.418092273222 -0.040689417609 -0.064538882629];

    T = 0;
    dwt_mode = 0;
    [h, w] = size(raw_img);
    filtered_img = raw_img;

    if horizontal == 1
        %Horizontal filtering
        for i = 1:h
            row_img = raw_img(i, :);
            filtered_row = filterSystem(row_img, wn, w, T, dwt_mode);
            filtered_img(i, :) = filtered_row;
        end

    else
        % Vertical filtering
        for i = 1:w
            col_img = raw_img(:, i);
            filtered_col = filterSystem(col_img', wn, h, T, dwt_mode);
            filtered_img(:, i) = filtered_col;
        end

    end

end
```

Down Sampler

```
function downSampledimg = downSampler(img, stride, odd, n, horizontal,T)
    [h, w] = size(img);
    partition = 2 ^ n;
    downSampledimg = cast(zeros(h),'like',img);

    if horizontal == 1

        if odd == 0
            %even for HPF
            downSampledimg(1:h, 1:w / 2) = img(1:h, 2:stride:w);
        else
            %odd for LPF
            downSampledimg(1:h, 1:w / 2) = img(1:h, 1:stride:w);
        end

    else

        if odd == 0
            %even for HPF
            downSampledimg(1:h / 2, 1:w) = img(2:stride:h, 1:w);
        else
            %odd for LPF
            downSampledimg(1:h / 2, 1:w) = img(1:stride:h, 1:w);
        end

    end

end
```

UpSampler

```
function upSampledimg = upSampler(img, stride, n, odd, horizontal)
    % Interpolation algorithm should be adopted to further increase resolution after upSampling
    % n means nth dwt_octave
    [h, w] = size(img);
    partition = 2 ^ n;
    new_h = 2 * h;
    new_w = 2 * w;

    if horizontal == 1
        upSampledimg = zeros(h);

        if odd == 0
            %even for HPF
            upSampledimg(1:h, 2:stride:w) = img(1:h, 1:w / 2);
        else
            %odd for LPF
            upSampledimg(1:h, 1:stride:w) = img(1:h, 1:w / 2);
        end

    else
        upSampledimg = zeros(2 * h);
        % Vertical
        if odd == 0
            %even for HPF
            upSampledimg(2:stride:new_h, 1:new_w / 2) = img(1:h, 1:w);
        else
            %odd for LPF
            upSampledimg(1:stride:new_h, 1:new_w / 2) = img(1:h, 1:w);
        end

    end

end
```

PSNR

```
function [psnr, difference] = PSNR(img, filtered_img)
    [h, w] = size(img);
    MAXI = 255;
    difference = (img - filtered_img) .^ 2;
    MSE = sum(difference, "all") / (h * w);

    psnr = 10 * log10(MAXI ^ 2 / MSE);
end
```

DWT octave

```
function filtered_img = octave(raw_img, stride, n, T)
    [h, w] = size(raw_img);

    switch n
        case 1
            output_type = T.lv1_output;
        case 2
            output_type = T.lv2_output;
        case 3
            output_type = T.lv3_output;
    end

    filtered_img = cast(raw_img, 'like', output_type);
    partition = 2 ^ n;

    % Horizontal
    H1 = gn_HPF(raw_img(1:h / (partition / 2), 1:w / (partition / 2)), 1, T, n);
    H1_ = downSampler(H1, stride, 0, n, 1, T);

    L1 = hn_LPF(raw_img(1:h / (partition / 2), 1:w / (partition / 2)), 1, T, n);
    L1_ = downSampler(L1, stride, 1, n, 1, T);

    % Vertical!, i have problem here! The high pass components leads to some errors.
    HH = gn_HPF(H1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
    HH_ = downSampler(HH, stride, 0, n, 0, T);

    HL = hn_LPF(H1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
    HL_ = downSampler(HL, stride, 1, n, 0, T);

    LH = gn_HPF(L1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
    LH_ = downSampler(LH, stride, 0, n, 0, T);

    LL = hn_LPF(L1_(1:h / (partition / 2), 1:w / partition), 0, T, n);
    LL_ = downSampler(LL, stride, 1, n, 0, T);

    % Trace till here, problems happens with LL_
    % Recombination
    filtered_img(1:h / partition, 1:w / partition) = LL_(1:h / partition, 1:w / partition);

    filtered_img(1:(h / partition), (w / partition) + 1:w / (partition / 2)) = HL_(1:h / partition, 1:w / partition);

    filtered_img((h / partition) + 1:(h / (partition / 2)), 1:(w / partition)) = LH_(1:h / partition, 1:w / partition);
    filtered_img((h / partition) + 1:(h / (partition / 2)), (w / partition) + 1:(w / (partition / 2))) = HH_(1:h / partition, 1:w / partition);

end
```

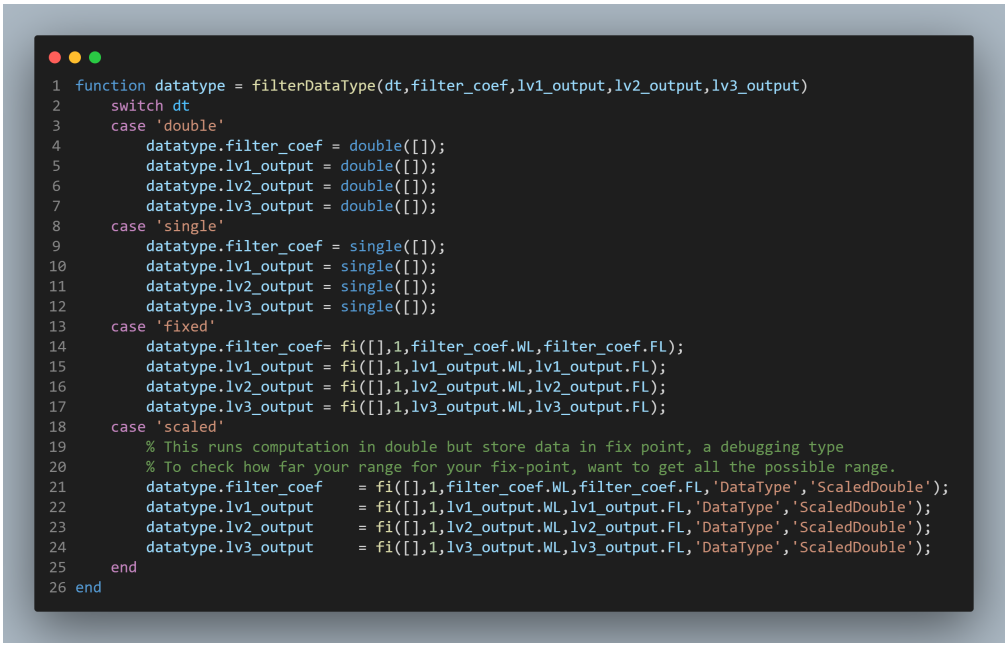
IDWT octave

```
function reconstructed_img = idwt_octave(compressed_img, stride, n)

    [h, w] = size(compressed_img);
    partition = 2 ^ n;
    reconstructed_img = compressed_img;

    % Extraction
    LL_ = compressed_img(1:h / partition, 1:w / partition);
    LH_ = compressed_img((h / partition) + 1:(h / (partition / 2)), 1:(w / partition));

    HL_ = compressed_img(1:(h / partition), (w / partition) + 1:w / (partition / 2));
    HH_ = compressed_img((h / partition) + 1:(h / (partition / 2)), (w / partition) + 1:(w / (partition / 2)));

    % Vertical!
    LL = upSampler(LL_, stride, n, 1, 0);
    LL_test = LL(1:h / (partition / 2), 1:w / partition);
    LL(1:h / (partition / 2), 1:w / (partition)) = qn_LPF(LL_test, 0);

    % LH is wrong~ for the last 3 columns.
    LH = upSampler(LH_, stride, n, 0, 0);
    LH(1:h / (partition / 2), 1:w / (partition)) = pn_HPF(LH(1:h / (partition / 2), 1:w / partition), 0);

    s0 = LL + LH;

    HL = upSampler(HL_, stride, n, 1, 0);
    HL(1:h / (partition / 2), 1:w / (partition)) = qn_LPF(HL(1:h / (partition / 2), 1:w / partition), 0);

    HH = upSampler(HH_, stride, n, 0, 0);
    HH(1:h / (partition / 2), 1:w / (partition)) = pn_HPF(HH(1:h / (partition / 2), 1:w / partition), 0);

    % Minor difference in boundary of 160 columns and rows.
    s1 = HL + HH;

    % Horizontal!
    s0_ = upSampler(s0, stride, n, 1, 1);
    disp("Size of s0_");
    disp(size(s0_));
    s0_ = qn_LPF(s0_, 1);

    s1_ = upSampler(s1, stride, n, 0, 1);
    disp("Size of s1_");
    disp(size(s1_));
    s1_ = pn_HPF(s1_, 1);

    s_temp = s0_ + s1_;

    reconstructed_img(1:h / (partition / 2), 1:w / (partition / 2)) = s_temp;

end
```

Main driver

```
filter_coef.WL = 11;
filter_coef.FL = 10;

lv1_output.WL = 10;
lv1_output.FL = 0;

lv2_output.WL = 11;
lv2_output.FL = 0;

lv3_output.WL = 12;
lv3_output.FL = 0;

odd = 0;
horizontal = 1;
zero_padded = 0;

% Broadcasting data types.
T = filterDataType('fixed', filter_coef, lv1_output, lv2_output, lv3_output);

%==========================================================
%  RD image
%==========================================================
img = double(imread('image.bmp'));
img_fixed = cast(img, 'like', T.lv3_output);

[h, w] = size(img_fixed);
n = 1;
stride = 2;

%==========================================================
%  DWT
%==========================================================
buildInstrumentedMex dwt -o dwt_mex ...
    -args {img_fixed, stride, T} -histogram

H1_ = dwt_mex(img_fixed, stride, T);

H1f = double(H1_);
figure(7);
imshow(H1f, []);
title('H1_');

%==========================================================
%  IDWT
%==========================================================
restored_image = idwt(H1f, stride, zero_padded);

%==========================================================
%  PSNR
%==========================================================
img = double(img);
disp("PSNR:");
[psnr, difference] = PSNR(img, restored_image);
fprintf('%.2f db\n', psnr);

% Verify results
showInstrumentationResults dwt_mex
% -proposeFL -defaultDT numerictype(1, 16)

% Code generation
codegen dwt ...
    -args {img_fixed, stride, T} -config:lib -report
```

Data Type



```
function datatype = filterDataType(dt,filter_coef,lv1_output,lv2_output,lv3_output)
    switch dt
    case 'double'
        datatype.filter_coef = double([]);
        datatype.lv1_output = double([]);
        datatype.lv2_output = double([]);
        datatype.lv3_output = double([]);
    case 'single'
        datatype.filter_coef = single([]);
        datatype.lv1_output = single([]);
        datatype.lv2_output = single([]);
        datatype.lv3_output = single([]);
    case 'fixed'
        datatype.filter_coef= fi([],1,filter_coef.WL,filter_coef.FL);
        datatype.lv1_output = fi([],1,lv1_output.WL,lv1_output.FL);
        datatype.lv2_output = fi([],1,lv2_output.WL,lv2_output.FL);
        datatype.lv3_output = fi([],1,lv3_output.WL,lv3_output.FL);
    case 'scaled'
        % This runs computation in double but store data in fix point, a debugging type
        % To check how far your range for your fix-point, want to get all the possible range.
        datatype.filter_coef    = fi([],1,filter_coef.WL,filter_coef.FL,'DataType','ScaledDouble');
        datatype.lv1_output     = fi([],1,lv1_output.WL,lv1_output.FL,'DataType','ScaledDouble');
        datatype.lv2_output     = fi([],1,lv2_output.WL,lv2_output.FL,'DataType','ScaledDouble');
        datatype.lv3_output     = fi([],1,lv3_output.WL,lv3_output.FL,'DataType','ScaledDouble');
    end
end
```

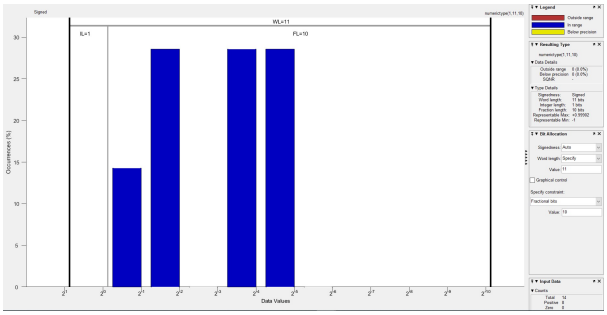
Result

Fixed Point statistical report

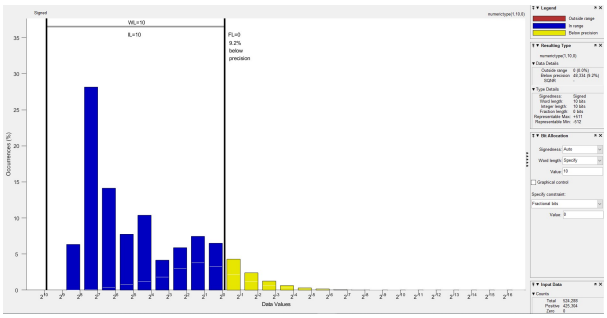
Fixed Point Width length and fraction length

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

Filter Coefficients

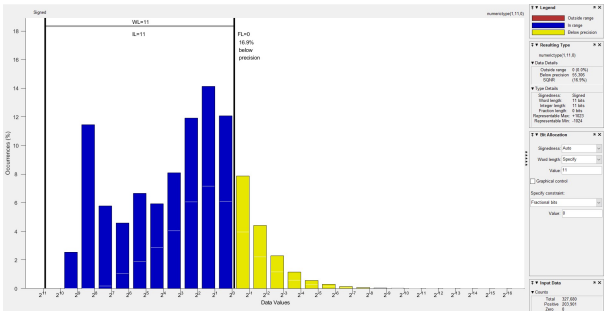


Level 1 output

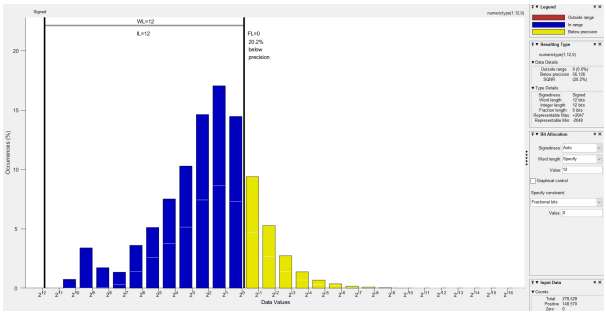


- In the fixed point designer, it tells you the range for your calculation also suggest you the length of the fix-point and fraction length.

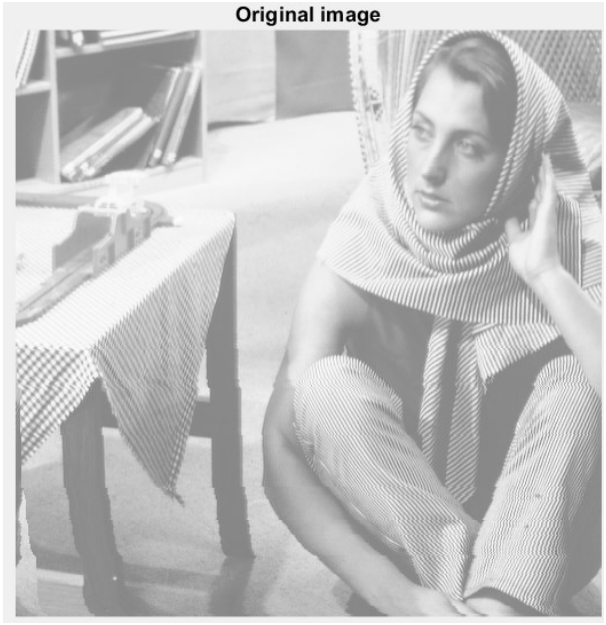
Level 2 output



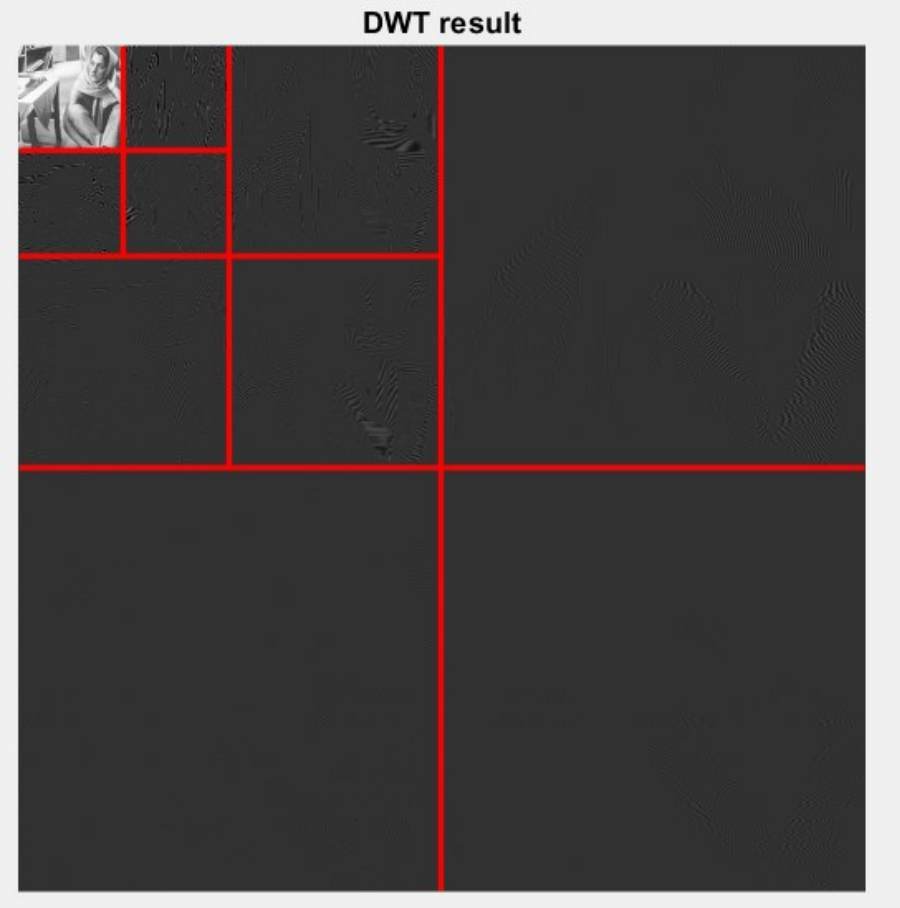
Level 3 output



Original image



DWT result



Restored image

PSNR

- 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

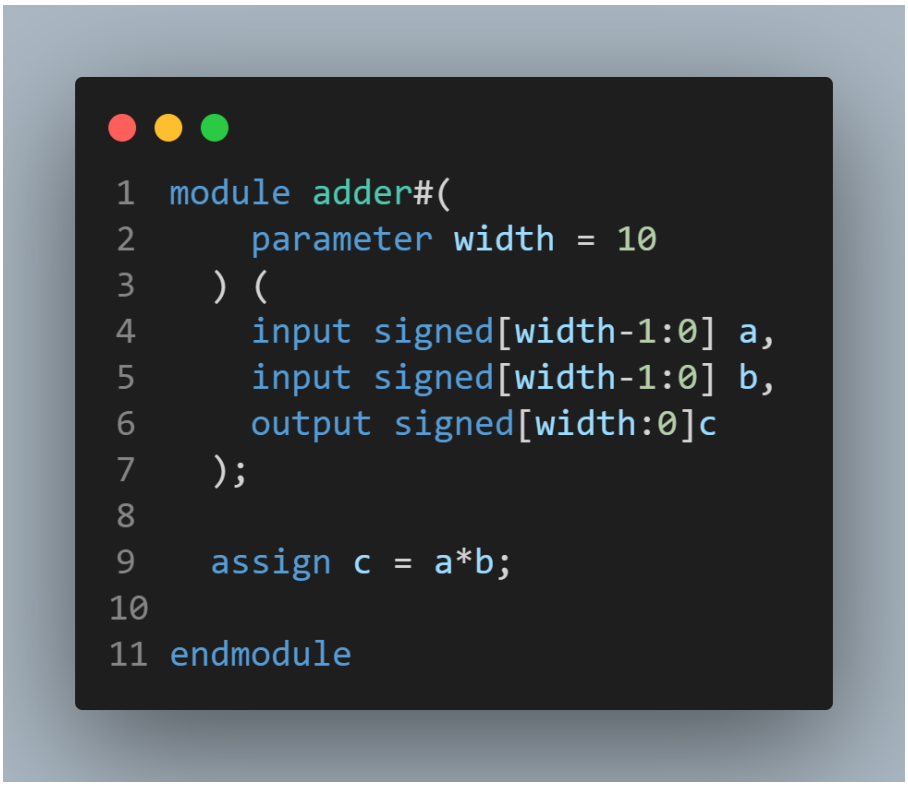
- Synthesize the word length for adders and filter's registers results according to the matlab simulation into gate-level circuit.

Multipler

```
module mult#(
    parameter width = 11
  ) (
    input signed[width-1:0] a,
    input signed[width-1:0] b,
    output signed[2*width-1:0]c
  );

  assign c = a*b;

endmodule
```

```
****************************************
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

```systemverilog
module register#(
    parameter width = 10
  ) (
    input clk,
    input rst,
    input signed [width-1:0] register_wr,
    output reg signed [width:0]  register_ff
  );

  always@(posedge clk)
  begin
    if(rst)
    begin
      register_ff <= 0;
    end
    else
    begin
      register_ff <= register_wr;
    end
  end
endmodule

```

lv1

```
****************************************
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 ackowledge EECS undergraduate Chun-Wei Su and EECS undergraduate Kuan-Ting Du for their assistent 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