目 录

1 课程设计任务 2 1
1.1 设计目的 1
1.2 设计要求 2
1.3 DWT 概述 2
2 系统总体设计方案（目标 1） 3
2.1 方案比较 3
2.2 方案设计 4
3 主要模块的设计思想与源程序设计（目标 2） 4
3.1 Matlab 模块 4
3.2 FPGA 模块 5
4 软硬件实验系统构建与测试结果（目标 3） 9
4.1 Maltab 模块 9
4.2 FPGA 模块 10

1　课程设计任务 2

1.1 设计目的

1）实时处理

FPGA 的并行计算能力和可编程性使其成为实时信号处理的理想选择。DWT 是一种常用的信号处理方法，广泛应用于图像压缩、信号去噪、特征提取等领域。通过在 FPGA 上实现 DWT，可以实现对实时信号的高效处理和分析。

2）高性能和低延迟

FPGA 具有并行处理的特点，可以同时处理多个数据点，从而实现高性能和低延迟的信号处理。对于大规模的数据集或实时处理要求较高的应用，使用 FPGA 实现 DWT 可以提供更快的计算速度和响应时间。

3）灵活性和可配置性

FPGA 具有可编程性，可以根据具体需求对 DWT 算法进行优化和配置。不同的小波函数、不同的滤波器长度和级数等参数可以根据应用需求进行选择和调整。通过在 FPGA 上实现 DWT，可以灵活地适应各种信号处理任务。

4）节省功耗和资源

FPGA 在实现 DWT 时可以高度优化计算架构，使其能够在相对较低的功耗下完成复杂的信号处理任务。此外，通过并行计算和硬件级别的优化，可以有效地利用 FPGA 资源，提高性能和效率。

5）可嵌入性

FPGA 具有较小的尺寸和低功耗的特点，适合嵌入到各种设备和系统中。通过在 FPGA 上实现 DWT，可以将信号处理功能嵌入到嵌入式系统、嵌入式图像传感器、无线通信设备等各种应用中，提供更高级的信号处理和分析能力。

1.2 设计要求

1）算法选择

选择适合应用需求的小波函数和小波滤波器。不同的小波函数有不同的性质和应用场景，可以根据信号特点和处理目标选择合适的小波函数。同时，确定小波滤波器的长度和级数，以平衡时域和频域的分辨率。

2）并行计算

利用 FPGA 的并行计算能力，将 DWT 算法中的多个计算步骤并行化，以提高计算效率和处理速度。可以利用并行计算单元、数据流架构或流水线技术来实现并行计算。

3）数据流控制

设计合适的数据流控制机制，以确保数据的正确流动和处理顺序。DWT 算法中多个计算步骤之间存在数据依赖关系，需要合理安排数据的传输和计算顺序，避免数据冲突和延迟。

4）性能优化

对 DWT 算法进行性能优化，减少计算和存储开销。可以通过优化滤波器设计、采用快速算法、减少不必要的计算等方法来提高性能和效率。

5）测试和验证

进行针对性的测试和验证，确保设计的正确性和可靠性。包括功能验证、性能测试和边界条件测试等，以保证设计满足预期要求。

1.3 DWT 概述

离散小波变换（Discrete Wavelet Transform，DWT）是一种常用的信号处理技术，用于对信号进行分析、压缩、去噪和特征提取等操作。DWT 基于小波函数的多尺度分析理论，将信号分解为不同频率和尺度的子信号，从而提供了对信号局部特征的多分辨率表示。

DWT 的基本思想是通过一系列的低通滤波和高通滤波操作，将信号分解为低频子信号（近似部分）和高频子信号（细节部分）。这一分解过程可以递归地进

行，得到不同尺度的子信号。在每一级分解中，低频子信号可以进一步分解，而高频子信号则表示信号的高频细节。

DWT 的分解过程可以表示为一个多级滤波器组的级联操作。通常采用的小波函数有 Haar 小波、Daubechies 小波、Symlet 小波等，它们具有不同的频率响应和性质，适用于不同类型的信号处理任务。

除了分解操作，DWT 还包括重构操作，用于将分解得到的子信号合成为原始信号。重构过程是分解过程的逆操作，通过级联的低通和高通滤波器组合，将低频子信号和高频子信号合成为原始信号。

DWT 的优势之一是提供了多分辨率表示。通过 DWT，可以将信号在不同的尺度上进行分析，从粗略的近似部分到细节部分，可以捕捉到信号的整体趋势和局部特征。这对于信号的压缩、去噪和特征提取等任务非常有用。

DWT 在实际应用中广泛用于信号和图像处理领域。例如，在图像压缩中，DWT 被用于将图像分解为低频和高频子带，然后对高频子带进行压缩，以减少图像数据的冗余。在信号去噪中，DWT 可以将噪声和信号的高频部分分离，通过去除高频细节实现信号的去噪效果。此外，DWT 还被广泛应用于语音处理、视频编码、生物医学信号分析等领域。

总之，离散小波变换（DWT）是一种基于小波函数的多尺度分析技术，用于信号的分解、重构和分析。它提供了多分辨率表示和对信号局部特征的捕捉，广泛应用于信号处理和图像处理领域。

2　系统总体设计方案（目标 1）

2.1 方案比较

1）方案一

基于二维的 DWT 变换实现图像压缩，优点是压缩效果好，难点是设计门槛高，开发周期长

2）方案二

基于一维的 DWT 实现信号的降噪，优点是去噪效果好，能够反映出信号的频域特性，设计难度较低，仿真容易实现

本课设设计采用方案二。

2.2 方案设计

将一个 300hz 高频信号与一个 10hz 低频信号混合作为输入信号，选择 db2 小波，层数选择两层，将输入信号进行分解，分解之后对得到的细节系数和近似系数进行阈值处理，本次设计直接忽略细节系数，然后进行信号的重构实现信号的降噪

3 主要模块的设计思想与源程序设计（目标 2）

3.1 Matlab 模块

生成原始信号：首先定义了采样率、时间向量和低频、高频信号频率，然后生成了一个包含两个频率成分的原始信号，并进行了量化和归一化处理。

写入 MIF 文件：将生成的原始信号数据写入到 MIF（Memory Initialization File）文件中，用于后续的 DWT 分解。

DWT 分解：选择了 Daubechies-2 小波作为小波基函数，指定了分解的层数。使用 wavedec 函数对原始信号进行 DWT 分解，得到近似系数和细节系数。

重构信号：使用 waverec 函数对 DWT 分解得到的系数进行重构，得到重构后的信号。

计算重构信号与原始信号的差别：将重构信号与原始信号进行比较，计算它们之间的差别。

绘制结果：使用 subplot 函数将原始信号、带噪信号、重构信号和差别分别绘制在四个子图中，以便观察和比较。

代码如下：

```
clc;
% 生成原始信号
fs = 1000; % 采样率
t = 0:1/fs:0.20; % 时间向量
f1 = 10; % 低频信号频率
f2 = 300; % 高频信号频率
org = 0.8*sin(2*pi*f1*t)*128+128;
x = 0.7*sin(2*pi*f1*t) + 0.3*sin(2*pi*f2*t); % 原始信号
x = floor(x *128 + 128);

fid = fopen('signal_data.mif', 'w');
pt_data_mif = x;
% 写入MIF文件的头部信息
depth = length(pt_data_mif);
width = 9;
fprintf(fid, 'WIDTH=%d;\n', width);
fprintf(fid, 'DEPTH=%d;\n', depth);
fprintf(fid, 'ADDRESS_RADIX=DEC;\n');
fprintf(fid, 'DATA_RADIX=BIN;\n');
fprintf(fid, 'CONTENT BEGIN\n');
```

```
% 将一维数组逐行写入文件
for i = 1:depth
    binaryData = dec2bin(pt_data_mif(i), width);
    fprintf(fid, '%d : %s;\n', i-1, binaryData);
end

fprintf(fid, 'END;\n');
fclose(fid);

% DWT分解
wavelet = 'db2'; % 小波基函数，这里使用Daubechies-2小波
level = 2; % 分解层数

[Lo_D,Hi_D,Lo_R,Hi_R] = wfilters(wavelet);
Lo_D = floor(Lo_D * 256);
Hi_D = floor(Hi_D * 256);
Lo_R = floor(Lo_R * 256);
% Hi_R = floor(Hi_R * 256);

[c1, l1] = wavedec(x, level, wavelet);
```

```
% 绘制结果
figure
subplot(4,1,1);
plot(t, org);
title('原始信号');

subplot(4,1,2);
plot(t, x);
% c(l(3):end)=0;
% plot(c, 'r');
title('带噪信号');

subplot(4,1,3);
% plot(t, x_reconstructed);t2
plot(t, t2(1:201));
title('重构信号');

subplot(4,1,4);
plot(t, difference);
title('差别');
```

3.2 FPGA 模块

1）分解模块

首先需要将信号的数据保存至 ROM 里面，然后读取出来处理。分解模块是采用 db2 小波的分解系数进行 FIR 滤波，本设计采用 6 级四阶并行 FIR 滤波，每次得到 6 个输入结果，其中 db2 小波的分解系数分为 Lo_D，Hi_D，每个系数共有 4 个参数。可以通过 matlab 查看分解系数的具体大小值，分为高通模块和低通小模块，这两个小模块内部的系数参数不同，信号的延迟数也不同。

```
module DWT_1
#(
    parameter w_in   = 9 ,//输入的位宽
    parameter y_out  = 25,//输出的位宽
    parameter c_in   = 9 //系数的位宽
)
(
    input clk,
    input rstn,
    // FIR Outputs
    output  signed [y_out-1:0]  Hi_D_c_y_6k  ,
    output  signed [y_out-1:0]  Hi_D_c_y_6k_1,
    output  signed [y_out-1:0]  Hi_D_c_y_6k_2,
    output  signed [y_out-1:0]  Hi_D_c_y_6k_3,
    output  signed [y_out-1:0]  Hi_D_c_y_6k_4,
    output  signed [y_out-1:0]  Hi_D_c_y_6k_5,
    output                      Hi_D_valid,
    // FIR Outputs
    output  signed [y_out-1:0]  Lo_D_c_y_6k  ,
    output  signed [y_out-1:0]  Lo_D_c_y_6k_1,
    output  signed [y_out-1:0]  Lo_D_c_y_6k_2,
    output  signed [y_out-1:0]  Lo_D_c_y_6k_3,
    output  signed [y_out-1:0]  Lo_D_c_y_6k_4,
    output  signed [y_out-1:0]  Lo_D_c_y_6k_5,
    output                      Lo_D_valid

);
// data_gen Outputs
wire  valid;
wire  signed [8:0]  data;
```

```
// data_gen Outputs
wire  valid;
wire  signed [8:0]  data;

data_gen  u_data_gen (
    .clk                    ( clk     ),
    .rstn                   ( rstn    ),

    .valid                  ( valid   ),
    .data                   ( data    )
);

reg [44:0] data_reg;
always @(posedge clk or negedge rstn) begin
    if(rstn == 1'b0)
        data_reg<= 0;
    else data_reg <= {data_reg[35:0],data};
end

wire signed [8:0] Hi_D_x_in;
assign Hi_D_x_in = data_reg[44:36];

// reg   signed [w_in-1:0]  x_in;
reg   signed [c_in-1:0]  Hi_D_c_0=-124;
reg   signed [c_in-1:0]  Hi_D_c_1=214;
reg   signed [c_in-1:0]  Hi_D_c_2=-58;
reg   signed [c_in-1:0]  Hi_D_c_3=-34;
```

2）下采样模块

将分解得到的 6 个并行数据进行下采样，下采样本质是对信号进行抽取。在 DWT 中是为了降低计算的复杂度。得到的 6 个数据，间隔抽取 3 个数据，所以在 6 个并行数据有效的时间内，需要连续的输出 3 个数据，因为时钟需要变为原来的一半。代码如下：

```
41      reg down_clk_reg;
42      always @(posedge clk or negedge rstn) begin
43          if(rstn == 1'b0)
44              down_clk_reg<= 0;
45          else if(!cnt[0]&&Hi_D_valid)
46              down_clk_reg<= 1;
47          else down_clk_reg<= 0;
48      end
49
50      reg  signed [y_out-1:0]  Hi_D_y_down_reg;
51      always @(posedge clk or negedge rstn) begin
52          if(rstn == 1'b0)
53              Hi_D_y_down_reg<= 0;
54          else if((cnt == 0)&&Hi_D_valid)
55              Hi_D_y_down_reg<= Hi_D_c_y_6k;
56          else if((cnt == 2)&&Hi_D_valid)
57              Hi_D_y_down_reg<= Hi_D_c_y_6k_2;
58          else if((cnt == 4)&&Hi_D_valid)
59              Hi_D_y_down_reg<= Hi_D_c_y_6k_4;
60          else  Hi_D_y_down_reg <= Hi_D_y_down_reg;
61      end
62
63      reg  signed [y_out-1:0]  Lo_D_y_down_reg;
64      always @(posedge clk or negedge rstn) begin
65          if(rstn == 1'b0)
66              Lo_D_y_down_reg<= 0;
67          else if((cnt == 0)&&Hi_D_valid)
68              Lo_D_y_down_reg<= Lo_D_c_y_6k;
69          else if((cnt == 2)&&Hi_D_valid)
70              Lo_D_y_down_reg<= Lo_D_c_y_6k_2;
```

3）上采样模块

上采样模块是为了在重构过程中恢复信号的信号精度和细节，本质是插值。在输入的数据流中，需要间隔的插入值，本设计插零值。所以时钟会变为原来的两倍，也即是恢复到原本的时钟

代码如下：

```
module upsample
#(
    parameter w_in_1   = 25 ,//输出的位宽
    parameter y_out_2  = 40//输入的位宽
)
(
    input input rstn,
    input rstn,
    input  Lo_D2_down_valid,
    input  signed [y_out_2-1:0]  Lo_D2_y_down,
    output  signed [w_in_1-1:0]  Lo_D2_y_up  ,
    output Lo_D2_up_valid,
    output up_clk

);
    assign up_clk =clk;

    reg flag;
    always @(posedge clk or negedge rstn) begin
        if(rstn == 1'b0)
            flag<= 0;
        else if(Lo_D2_down_valid)
            flag<= !flag;
        else flag<= flag;
    end

    wire  signed [w_in_1-1:0]  Lo_D2_data;
    assign Lo_D2_data = {Lo_D2_y_down[y_out_2-1],Lo_D2_y_down[w_in_1-2:0]};

    assign Lo_D2_up_valid = Lo_D2_down_valid;
    assign Lo_D2_y_up = (flag==1) ? 0 :Lo_D2_data;
```

4）重构模块

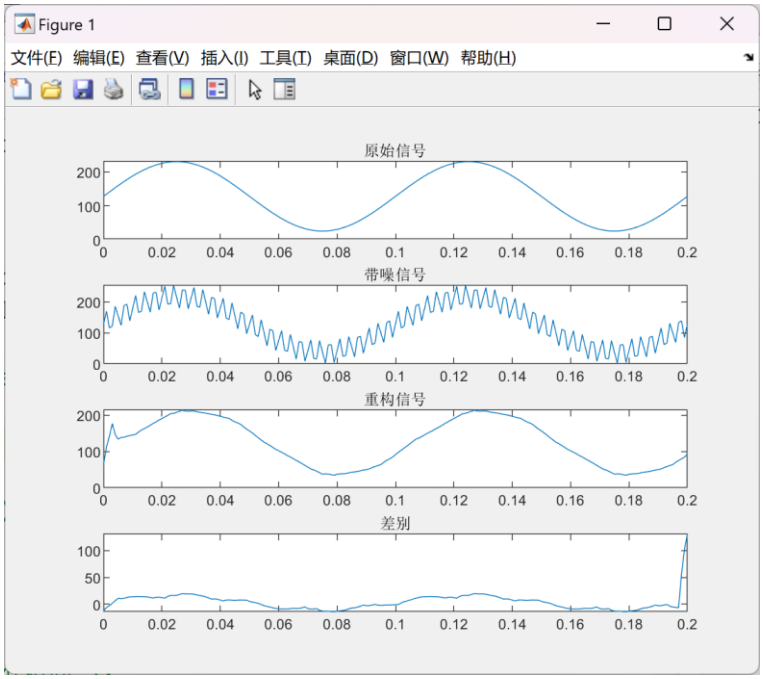
将输入的数据流进行 FIR 滤波，分为高通和低通重构，系数分别为 Hi_R，Lo_R。分解两层之后会得到一个近似系数，与三个细节系数，采样率为 1000hz，根据采样定理，分解过程中 500hz 首先分为 0-250,250-500，再次分解，将 250 分解为 0-125,125-250，所以混合信号的 10hz 主要集中在 0-125,300hz 主要集中在 250-500。因此忽略分解得到的细节系数，只对近似系数进行重构理论上应该可以较好的还原出 10hz 信号。在 matlab 仿真可以发现，的确可以较好的还原出 10hz 信号。因此对近似系数进行两次重构之后，便可以还原出 10hz 的信号。代码如下：

```
    input  signed [w_in-1:0]  Lo_R_up_y_k    ,
    output  signed [y_out-1:0]  Lo_R1_c_y_k  ,
    output                      Lo_R1_valid
);

reg [4:0] Lo_R1_valid_reg;
always @(posedge clk or negedge rstn) begin
    if(rstn == 1'b0)
        Lo_R1_valid_reg<= 0;
    else Lo_R1_valid_reg <= {Lo_R1_valid_reg[3:0],Lo_D_up_valid};
end

assign Lo_R1_valid = Lo_R1_valid_reg[4];

reg   signed [c_in-1:0]  Lo_R1_c_0=123;
reg   signed [c_in-1:0]  Lo_R1_c_1=214;
reg   signed [c_in-1:0]  Lo_R1_c_2=57;
reg   signed [c_in-1:0]  Lo_R1_c_3=-34;
```

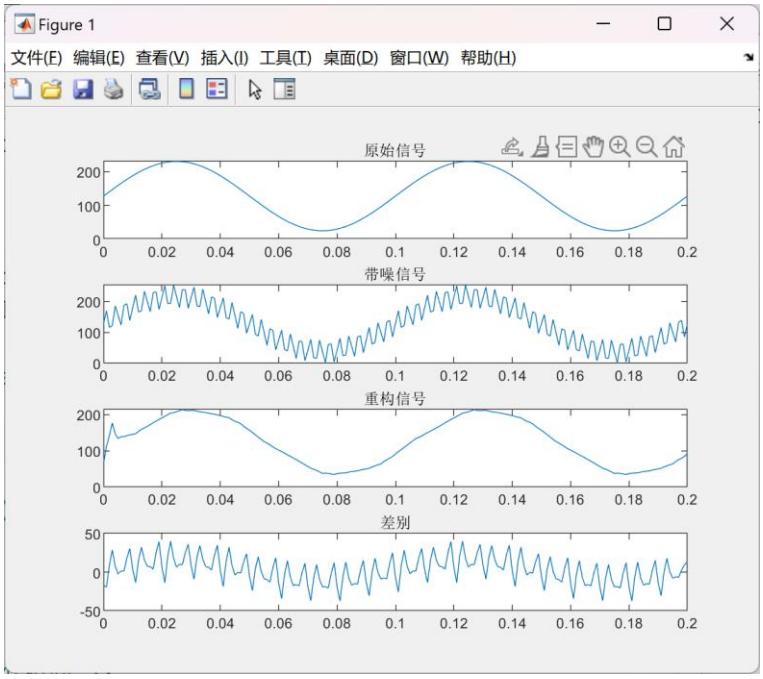
4　软硬件实验系统构建与测试结果（目标 3）

4.1 Maltab 模块

忽略所有细节系数进行还原如下：



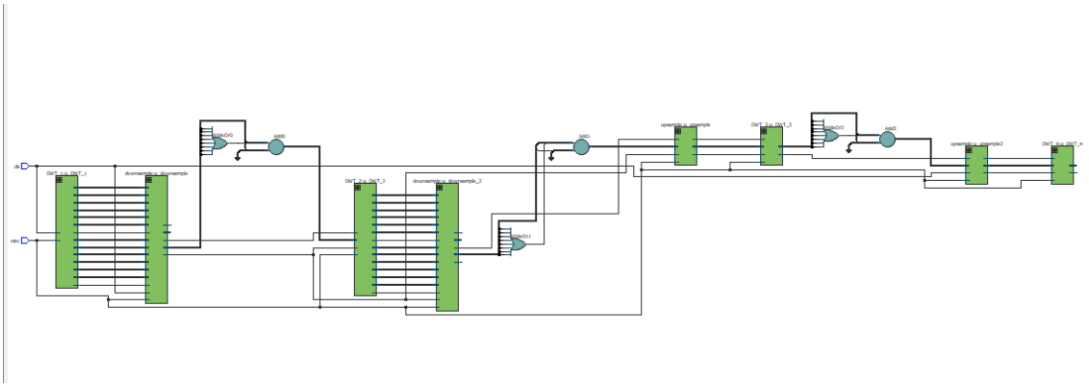
只忽略第一次分解得到的细节数据得到的结果如下：



可以看到，只忽略第一次分解得到的细节数据所得到的结果误差反而变大了许多。

4.2 FPGA 模块

1）RTL



2）仿真波形