# 课程名称：EDA 技术综合设计

实验名称: 实验八 创新设计

基于串口的数字滤波器设计和硬件验证平台

团队成员

| 姓名 | 班级 | 学号 |
| --- | --- | --- |
| 王峤宇 | 通信 214 | 214022 |
| 朱书硕 | 通信 214 | 214026 |
| 王占同 | 通信 214 | 214010 |

# 一、设计内容及原理

数字滤波器从实现结构上划分，有 FIR 和 IIR 两种。FIR (Finite Impulse Response) 滤波器的特点是它的冲击响应是有限的，它跟过去的信号无关，所以在使用时容易实现，速度快。本次任务为设计一个 FIR 滤波器。

## FIR 滤波器原理

FIR 滤波器简单来说, 其本质上就是加权有限数量输入信号的过去值, 来计算当前的输出, 比如针对按键的消抖, 采样多次值, 由过去判断当前输出, 本质上就是一种滤波器, 但是按键消抖是一种中值滤波器。

FIR 的基本原理由如下公式给出:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k] \tag{1}$$

其中,

- $y[n]$ 是输出信号。
- $x[n]$ 是输入信号。
- $h[k]$ 是滤波器的脉冲响应系数。
- $N$ 是滤波器的长度（系数的数量）。

线性相位 FIR 滤波器的关键在于其脉冲响应 $h[k]$ 的对称性。这种对称性可以保证滤波器具有线性相位特性。根据对称性的不同，可以将线性相位 FIR 滤波器分为以下两种类型：

1. 偶对称 FIR 滤波器, 第一类线性相位

$$h[k] = h[N-1-k] \tag{2}$$

2. 奇对称 FIR 滤波器, 第二类线性相位

$$h[k] = -h[N-1-k] \tag{3}$$

实际应用中, 通常采用第一类线性相位, 便于运算的同时, 其相位特性也是严格不失真的。

## 线性相位 FIR 滤波器结构

FIR 的设计通常借助其对称的特性简化设计, 由于对称性的存在，可以减少乘法运算的数量。例如，对于一个长度为 $N$ 的对称 FIR 滤波器，只需要将输入数据相加, 采用 $\frac{N}{2}$ 的乘法器, 将其与 FIR 系数相乘再相加就可以。

## 串行和并行设计思路

### 并行设计

在 FPGA 设计中，为了提高计算效率，通常会采用并行和流水线结构。通过并行处理多个乘法和加法操作，可以加快滤波器的处理速度。同时，通过流水线技术，可以将计算过程分解为多个阶段，每个阶段独立完成部分计算，从而提高整体计算效率。

### 串行设计

在 FPGA 中实现 FIR 滤波器时，可以采用串行设计方法。这种方法的主要特点是每个时钟周期只处理一个输入数据，通过多个时钟周期完成整个 FIR 滤波器的计算。串行设计的主要优点是硬件资源需求较少，但处理速度较慢。串行 FIR 滤波器的基本结构如下：

- **移位寄存器**：用于存储输入信号的过去值。每个时钟周期，新的输入样本被移入寄存器，而最旧的样本被移出。
- **乘法器**：用于将输入样本与滤波器系数相乘。由于是串行结构，通常只有一个乘法器，每个时钟周期进行一次乘法运算。
- **累加器**：用于累加乘法器的输出。累加器的结果在每个时钟周期更新，直到所有系数都计算完毕，得到最终的滤波器输出。

## 基础任务

**设计任务**: 利用板卡资源，设计一个 8 阶的 FIR 低通滤波器, 采样频率是 5KHz, 阻带截止频率是 1KHz. 具体频率可以板卡实际资源为参照进行调整。基于并行结构设计的 FIR 的 8 阶数字滤波器

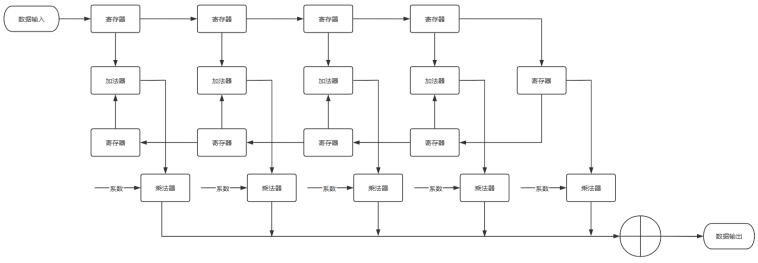


图 1: 8 阶并行 FIR 滤波器的模块框图

模块框图如图 1所示。

## 提高内容

**设计任务**: 在基础任务的基础上，设置任意阶 FIR 滤波器，要求采用模块化设计思想设计。15 阶并行 FIR 滤波器的模块框图如图 2所示。

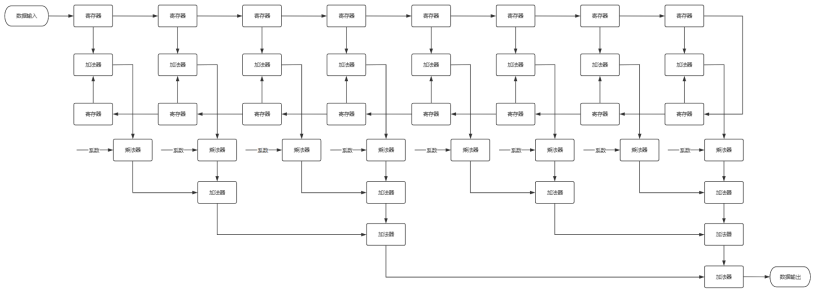

图 2: 15 阶并行 FIR 滤波器模块框图

## 拓展任务

**任务要求**: 滤波器的架构方法可采用串行、并行等不同方法来实现，采用不同方法设计实现 FIR 滤波器。基于串行结构设计的 15 阶 FIR 滤波器的模块框图如图 3所示。

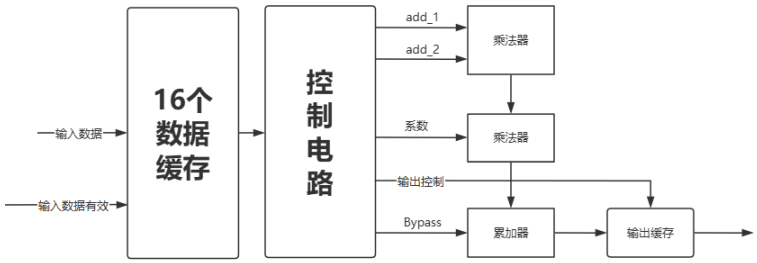


图 3: 15 阶串行 FIR 滤波器模块框图

## 创新设计

**设计思路**: 不满足仅仅在仿真中实现的 FIR 滤波器, 将搭建的 FIR 滤波器综合并部署到 FPGA 开发板上, 利用串口与 FPGA 通信, PC 机向 FPGA 传输待滤波数据, FPGA 将滤波输出结果回传到 FPGA 上,PC 通过 MATLAB 或其他语言对数据进行验证。

**十六倍过采样原理**: 标准 UART 的 RXD 前端有一个”1 到 0 跳变检测器”, 当其连续接受到 8 个 RXD 上的地电平时, 该检测器就认为 RXD 线出现了起始位, 进入接受数据状态. 在接受状态, 接受控制器对数据位 7,8,9 三个脉冲采样, 并遵从三中取二的原则确定最终值. 采用这一方法的根本目的还是为了增强抗干扰, 提高数据传送的可靠性, 采样信号总是在每个接受位的中间位置, 可以避开数据位两端的边沿失真, 也可以防止接受时钟频率和发送时钟频率不完全同步引起的误差.

十六倍过采样本身也是对 UART 这种异步通信的一个延迟补偿, 如果没有过采样的话, 发送端和接收端的时钟将会 0 1 个波特率时钟周期, 引入 16 倍过采样后, 检测到发送的的起始位后, 实现时钟同步, 由于同步的过程由 16 倍过采样的时钟控制, 同步之后的时钟相差仅为 0 1/16 个波特率时钟周期。

十六倍过采样的示意图如图 4所示。

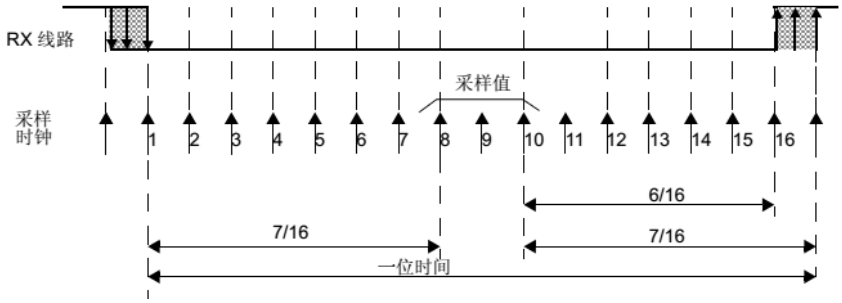


图 4: 十六倍过采样示意图

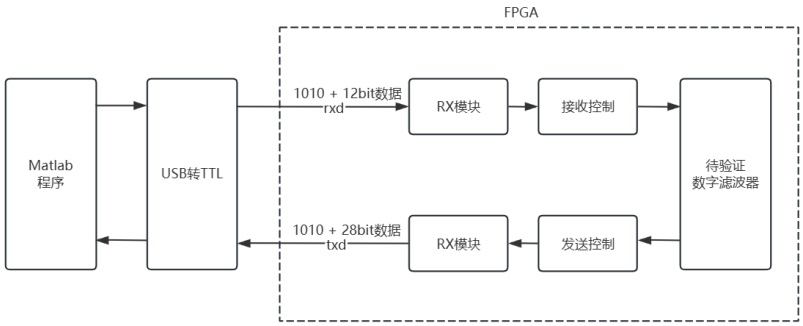


图 5: 创新设计的模块框图

该部分的模块框图如图 5所示, 主要由待验证 FIR 和串口模块以及相应的控制电路构成, 上位机部分通过 Matlab 程序完成串口通信, 直接利用 FPGA 进行滤波然后检验数据正确性。

## 二、设计过程

### 基础任务

并行设计, 使用 Xilinx 提供的 IP 核进行设计, 源文件如下:

Listing 1: 并行八阶 FIR 滤波器

```
/* 8阶并行FIR滤波器 */
module fir_parallel (
        input rst,            // 复位，低有效
        input clk,            // 工作频率，即采样频率
        input en,             // 输入数据开始信号
        input [11:0]  xin,    // 输入混合频率的信号数据
        output out_rdy,       /* 输出数据有效 */
        output [28:0]  yout   // 输出数据
    );
    /* 滤波器设计 */
    parameter ORDER = 8;                /* 默认为N为偶数 */
    parameter MULT_NUM = ORDER/2 + 1;   /* 系数为奇数 */
```

```
13      parameter DATA_NUM = (ORDER+1);      /* 系数为奇数 */
14
15      wire signed [11:0] coe[MULT_NUM-1:0];
16      assign coe[0] = 12'd60;
17      assign coe[1] = 12'd165;
18      assign coe[2] = 12'd307;
19      assign coe[3] = 12'd432;
20      assign coe[4] = 12'd482;
21
22      //(0) 输入数据有效缓存，保证输入数据延迟5个时钟时输出
23      reg [4:0] en_r;
24      always @(posedge clk or posedge rst) begin
25          if (rst) begin
26              en_r[4:0] <= 'b0;
27          end
28          else begin
29              en_r[4:0] <= {en_r[3:0], en};
30          end
31      end
32
33      //(1) 读取数据
34      reg signed [11:0] xin_reg[DATA_NUM-1:0];
35      integer i, j;
36      always @(posedge clk or posedge rst) begin
37          if (rst) begin
38              for (i = 0; i < DATA_NUM; i = i+1) begin
39                  xin_reg[i]  <= 12'b0;
40              end
41          end
42          else if (en) begin
43              xin_reg[0] <= xin;
44              for (j = 0; j < DATA_NUM - 1; j = j+1) begin
45                  xin_reg[j+1] <= xin_reg[j]; //周期性移位操作
46              end
47          end
48      end
49
50      //(2) 首尾相加，实现线性相位结构，延迟两个时钟
51      wire signed [12:0] add_reg[MULT_NUM-1:0];
52      genvar l;
53      generate
54          for (l = 0; l < DATA_NUM/2; l = l+1) begin
55              c_addsub_0 your_instance_name (
56                  .A(xin_reg[l]),              // 首部数据
57                  .B(xin_reg[DATA_NUM-1-l]),   // 尾部数据
58                  .CLK(clk),                   // 输入时钟
59                  .S(add_reg[l])               // 输出数据
60              );
61          end
62      endgenerate
63
64      /* 人为延迟两个周期 */
```

```
    reg signed [12:0] add_reg_r[1:0];
    assign add_reg[MULT_NUM-1] = add_reg_r[1];
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            add_reg_r[0] <= 'b0;
            add_reg_r[1] <= 'b0;
        end
        else if(en_r[3]) begin
            add_reg_r[0] <= {xin_reg[MULT_NUM-1][11], xin_reg[MULT_NUM-1]};
            add_reg_r[1] <= add_reg_r[0];
        end
    end

    // (3) 乘法器，延迟3个时钟
    wire signed [24:0] mout[MULT_NUM-1:0];
    genvar k;
    generate
        for (k=0; k < MULT_NUM; k=k+1) begin
            mult_gen_0 mult_parallel (
                .CLK(clk),          // 时钟
                .A(add_reg[k]),     // 13位的有符号数据
                .B(coe[k]),         // 12位的有符号系数
                .P(mout[k])         // 25位的输出数据
            );
        end
    endgenerate

    //(4) 积分累加，MULT_NUM组25bit数据
    reg signed [28:0] sum;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            sum  <= 29'd0;
        end
        else begin
            sum  <= mout[0] + mout[1] + mout[2] + mout[3] + mout[4];
        end
    end

    reg signed [28:0] yout_r;
    reg out_rdy_r;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            yout_r <= 29'd0;
        end
        else if (en_r[4]) begin
            yout_r <= sum;
            out_rdy_r <= 1'b1;
        end
        else begin
            out_rdy_r <= 1'b0;
        end
    end
```

```
    assign out_rdy = out_rdy_r;
    assign yout  = yout_r;
endmodule
```

通过 Matlab 程序生成用于仿真的数据结果如下:

Listing 2: 生成 12 位的波形数据

```
clear all;close all;clc;

fs = 5e3;
T = 1 / fs;
n = 0:2047;
t = n * T;
x = cos(2*pi*50*t) + 0.5*cos(2*pi*2000*t);
x = mapminmax(x);
x_dis = floor((2 ^ 11 - 1) * x);
fid = fopen('signal_data.txt', 'wt'); %写数据文件
for k=1:2048
    B_si = dec2bin(x_dis(k) + (x_dis(k)<0)*2^12, 12);
    fprintf(fid,'%s\n',B_si);
end
fprintf(fid,';');
fclose(fid);
stem(x(1:256));
title('原信号');
```

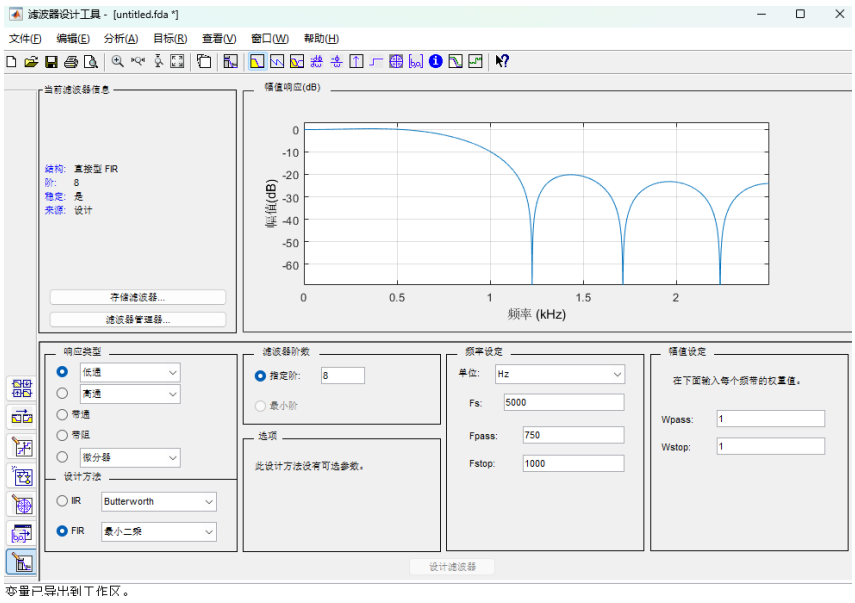


图 6: 滤波器设计

设计的滤波器如图 6所示, 采用最小二乘法设计, 阻带权值为 40, 通带为 1, 阻带截止频率为 1000Hz, 对输入为 200Hz+2000Hz 的信号进行滤波。

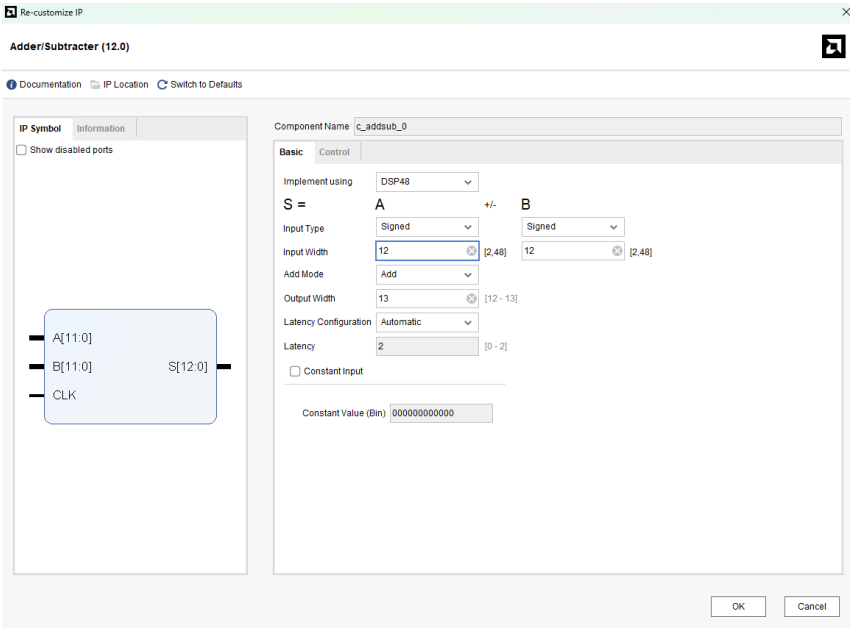


图 7: 加法器 IP

加法器 IP 核的设置如图 7所示, 使用 Xilinx 提供的 DSP Slice 资源上提供的乘法器和加法器实现。

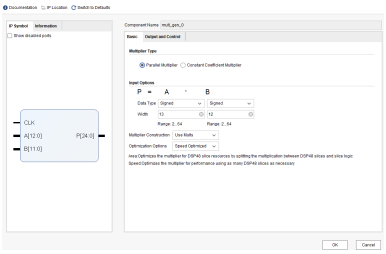

图 8: 乘法器 IP 核配置 a

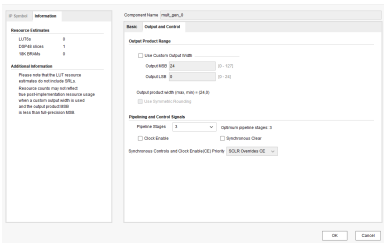

图 9: 乘法器 IP 核配置 b

乘法器的 IP 核配置结果如图 8和图 9所示, DSP Slice 资源提供的乘法器为流水线工作方式, 配置为 13 位和 12 位的符号数输入, 完成两个数据相加后和 FIR 系数的乘法运算, 在设置流水线阶段时, 设置为系统提示的最佳阶段, 三段流水线, 输出结果延迟三个时钟。仿真文件如下文件如下:

Listing 3: 8 阶 FIR 滤波器仿真文件

```
module fir_parallel_tb;
    reg  rst;
    reg  clk;
    reg  en;
    reg [11:0] xin;
    wire signed [28:0] yout;
    wire out_rdy;
    parameter SIN_DATA_NUM = 2048;
    parameter FILE_PATH_A = "E:/Projects/matlabProject/fpgaTools/signal_data.txt";
    parameter FILE_PATH_B = "E:/Projects/matlabProject/fpgaTools/signal_res.txt";
    initial begin
        rst = 0;
        clk = 0;
```

```
14          #10;
15          rst = 1;
16          #10;
17          rst = 0;
18      end
19
20      reg [11:0] stimulus[0: SIN_DATA_NUM-1];
21      integer i, j;
22      integer file;
23      initial begin
24          $readmemb(FILE_PATH_A, stimulus);
25          file = $fopen(FILE_PATH_B, "w");
26          en = 1;
27          i = 0;
28          j = 0;
29          xin = 0;
30          #200;
31          forever begin
32              xin = stimulus[i];
33              @(negedge clk) begin
34                  if (i == SIN_DATA_NUM-1) begin
35                      i = 0;
36                  end
37                  else begin
38                      i = i + 1;
39                  end
40              end
41              @(posedge clk) begin
42                  if (out_rdy) begin
43                      $fwrite(file, "%d\n", yout);
44                      j = j + 1;
45                      if (j == SIN_DATA_NUM) begin
46                          $finish;
47                          $fclose(file);
48                      end
49                  end
50              end
51          end
52      end
53
54      fir_parallel  fir_parallel_inst (
55          .rst(rst),
56          .clk(clk),
57          .en(en),
58          .xin(xin),
59          .yout(yout),
60          .out_rdy(out_rdy)
61      );
62
63      always #5  clk = ! clk ;
64
65  endmodule
```

## 提高任务

设计 15 阶的 FIR 滤波器, 采样频率 16kHz, 输入信号为语音信号, 使用低通滤波器保留 3.4kHz 以下的信号, 滤除 5000hz 单频噪声, 并且改善最终计算的求和过程。改善后的 FIR 滤波器源文件如下:

Listing 4: 15 阶并行 FIR 滤波器

```
/* 16阶并行FIR滤波器 */
module fir_parallel_15 (
        input rst,            // 复位，低有效
        input clk,            // 工作频率，即采样频率
        input en,             // 输入数据开始信号
        input [11:0] xin,     // 输入混合频率的信号数据
        output out_rdy,       // 输出数据有效
        output [27:0] yout    // 输出数据
    );
    /* 滤波器设计 */
    parameter ORDER = 15;                /* 默认N为奇数 */
    parameter MULT_NUM = (ORDER+1)/2;    /* 乘法器个数为偶数 */
    parameter DATA_NUM = (ORDER+1);      /* 数据个数为偶数数 */

    wire signed [11:0] coe[MULT_NUM-1:0];
    assign coe[0] = -12'd94;
    assign coe[1] =  12'd101;
    assign coe[2] =  12'd190;
    assign coe[3] =  12'd75;
    assign coe[4] = -12'd153;
    assign coe[5] = -12'd110;
    assign coe[6] =  12'd362;
    assign coe[7] =  12'd865;

    //(0) 输入数据有效缓存，保证输入数据延迟5+6个时钟时输出
    reg [10:0] en_r;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            en_r <= 'b0;
        end
        else begin
            en_r <= {en_r[9:0], en};
        end
    end

    //(1) 读取数据
    reg signed [11:0] xin_reg[DATA_NUM-1:0];
    integer i, j;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            for (i = 0; i < DATA_NUM; i = i+1) begin
                xin_reg[i]  <= 12'b0;
```

```
            end
        end
        else if (en) begin
            xin_reg[0] <= xin;
            for (j = 0; j < DATA_NUM - 1; j = j+1) begin
                xin_reg[j+1] <= xin_reg[j]; //周期性移位操作
            end
        end
    end

    //(2) 首尾相加，实现线性相位结构，延迟两个时钟
    wire signed [12:0] add_reg[MULT_NUM-1:0];
    genvar l;
    generate
        for (l = 0; l < DATA_NUM/2; l = l+1) begin
            c_addsub_0 adder_12bit (
                .A(xin_reg[l]),             // 首部数据
                .B(xin_reg[DATA_NUM-1-l]),  // 尾部数据
                .CLK(clk),                  // 输入时钟
                .S(add_reg[l])              // 输出数据
            );
        end
    endgenerate

    // (3) 乘法器，延迟3个时钟
    wire signed [24:0] mout[MULT_NUM-1:0];
    genvar k;
    generate
        for (k=0; k < MULT_NUM; k=k+1) begin
            mult_gen_0 mult_25bit (
                .CLK(clk),          // 时钟
                .A(add_reg[k]),     // 13位的有符号数据
                .B(coe[k]),         // 12位的有符号系数
                .P(mout[k])         // 25位的输出数据
            );
        end
    endgenerate

    //(4) 积分累加，8组25bit数据，输出28bit数据，使用三级加法器，延迟2*3个周期
    wire signed [27:0] sum[6:0];
    genvar m;
    generate
        for (m=0; m < 7; m=m+1) begin
            if (m < 4) begin
                c_addsub_1 adder_28bit (
                    .A({{3{mout[m*2][24]}}, mout[m*2]}),
                    .B({{3{mout[m*2+1][24]}}, mout[m*2+1]}),
                    .CLK(clk),
                    .S(sum[m])
                );
            end
            else begin
```

```
                c_addsub_1 adder_28bit (
                    .A(sum[(m-4)*2]),
                    .B(sum[(m-4)*2+1]),
                    .CLK(clk),
                    .S(sum[m])
                );
            end
        end
    endgenerate
    reg signed [27:0] yout_r;
    reg out_rdy_r;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            yout_r <= 28'd0;
        end
        else if (en_r[10]) begin
            yout_r <= sum[6];
            out_rdy_r <= 1'b1;
        end
        else begin
            out_rdy_r <= 1'b0;
        end
    end
    assign out_rdy = out_rdy_r;
    assign yout = yout_r;
endmodule
```

针对 15 阶的 FIR 滤波器的仿真文件如下:

Listing 5: 15 阶并行 FIR 滤波器的仿真文件

```
module fir_parallel_15_tb;
    reg  rst;
    reg  clk;
    reg  en;
    reg [11:0] xin;
    wire signed [28:0] yout;
    wire out_rdy;
    parameter SIN_DATA_NUM = 2048;
    parameter FILE_PATH_A = "E:/Projects/matlabProject/fpgaTools/signal_data.txt";
    parameter FILE_PATH_B = "E:/Projects/matlabProject/fpgaTools/signal_res.txt";
    initial begin
        rst = 0;
        clk = 0;
        #10;
        rst = 1;
        #10;
        rst = 0;
    end

    reg [11:0] stimulus[0: SIN_DATA_NUM-1];
    integer i, j;
    integer file;
```

```
    initial begin
        $readmemb(FILE_PATH_A, stimulus);
        file = $fopen(FILE_PATH_B, "w");
        en = 1;
        i = 0;
        j = 0;
        xin = 0;
        #200;
        forever begin
            xin = stimulus[i];
            @(negedge clk) begin
                if (i == SIN_DATA_NUM-1) begin
                    i = 0;
                end
                else begin
                    i = i + 1;
                end
            end
            @(posedge clk) begin
                if (out_rdy) begin
                    $fwrite(file, "%d\n", yout);
                    j = j + 1;
                    if (j == SIN_DATA_NUM) begin
                        $finish;
                        $fclose(file);
                    end
                end
            end
        end
    end

    fir_parallel_15 fir_parallel_inst (
        .rst(rst),
        .clk(clk),
        .en(en),
        .xin(xin),
        .yout(yout),
        .out_rdy(out_rdy)
    );

    always #5  clk = ! clk ;

endmodule
```

语音信号的 matlab 文件如下:

Listing 6: 语音信号文件处理

```
clear all; clc; close all;

[file, path]  = uigetfile('*.wav');     % 读取音频
[xx, fs] = audioread([path, file]);     % 读取音频

```

```
xx=xx-mean(xx);                          % 消除直流分量
x=xx/max(abs(xx));                       % 幅值归一化
N=length(x);                             % 计算音频长度

time=(0:N-1)/fs;                         % 计算时间
noiseFreq=5000;                          % 单频噪声频率
noiseAmp=0.125;                          % 单频噪声幅度
signal=x + noiseAmp*cos(2*pi*noiseFreq*time)'; % 添加噪声
res = mapminmax(signal);
x_dis = floor((2 ^ 11 - 1) * res);
fid = fopen('signal_data.txt', 'wt'); %写数据文件
for k=1:N
    B_si = dec2bin(x_dis(k) + (x_dis(k)<0)*2^12, 12);
    fprintf(fid, '%s\n', B_si);
end
fprintf(fid,';');
fclose(fid);

% 作图
subplot 411; plot(time,x,'k'); grid;
title('原始语音'); ylabel('幅值'); xlabel('时间/s');

subplot 412; audioFFT = fft(x);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('原始语音频谱'); ylabel('幅值'); xlabel('freq(Hz)');

subplot 413; plot(time,signal,'k');grid;
title('带噪语音'); ylabel('幅值'); xlabel('时间/s');

subplot 414; audioFFT = fft(signal);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('带噪语音频谱'); ylabel('幅值'); xlabel('freq(Hz)');
```

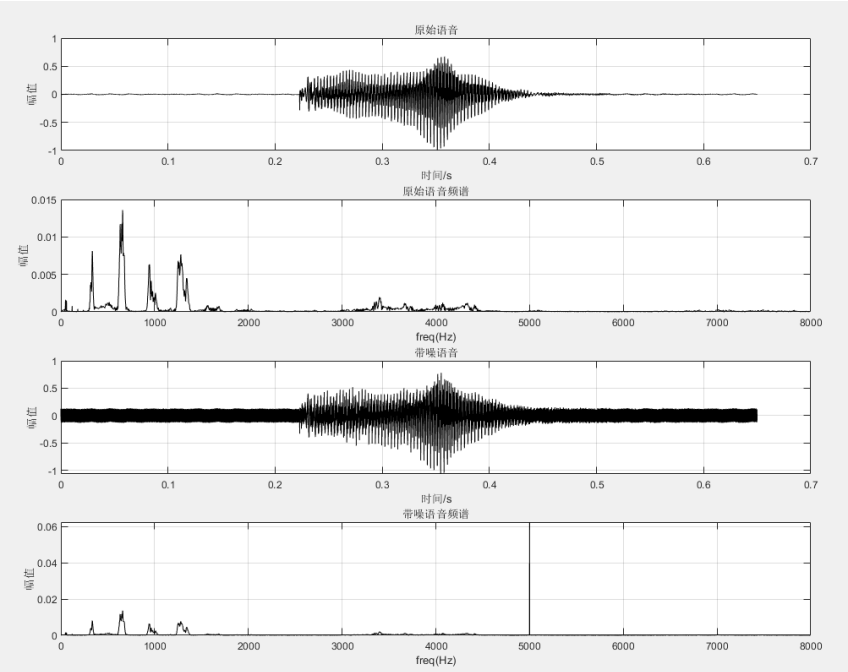


图 10: 语音信号时域和频域图

语音信号频谱和时域图如图 10所示, 在 16000 采样速率的语音信号中添加了幅值为 0.125 单频噪音信号, 并将其转变为 12bit 的补码输出到 txt 文件中, 供 Verilog 仿真的系统级函数 $readmemb 读取, 并输入到 FIR 滤波器之中。

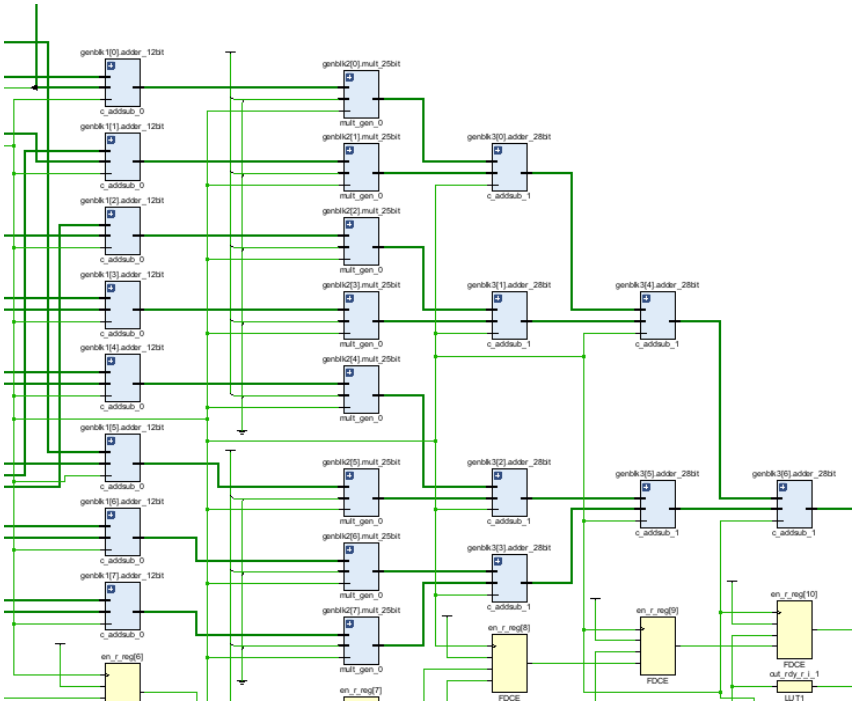

图 11: 15 阶并行 FIR 滤波器综合结果

15 阶并行 FIR 滤波器的综合结果如图 11所示, 使用了 8 个乘法器和 8+7 个加法器, 极大的消耗资源, 但整个系统仅使用到一个 LUT。

## 拓展任务

15 阶串行 FIR 滤波器源文件如下:

Listing 7: 15 阶串行 FIR 滤波器源文件

```
`timescale 1ns / 1ns
/* 串行设计FIR滤波器 */
module fir_serial (
    input rst,          // 低电平复位
    input clk,          // 系统工作时钟，100MHz
    input en,           // 输入数据有效信号
    input [11:0] xin,   // 输入信号数据
    output out_rdy,       // 输出数据有效信号
    output [27:0] yout  // 输出数据
    );
    // 数据有效缓冲
    reg [15:0] en_r;
    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            en_r[15:0] <= 'b0;
        end
        else begin
            en_r[15:0] <= {en_r[14:0], en};
        end
    end

    // 滤波器系数
    wire signed [11:0] coe[7:0];
    assign coe[0] = -12'd94;
    assign coe[1] =  12'd101;
    assign coe[2] =  12'd190;
    assign coe[3] =  12'd75;
    assign coe[4] = -12'd153;
    assign coe[5] = -12'd110;
    assign coe[6] =  12'd362;
    assign coe[7] =  12'd865;

    // (1) 输入数据移位部分
    reg [2:0] cnt;
    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            cnt <= 3'b0;
        end
        else if (en || cnt != 0) begin
            cnt <= cnt + 1'b1;    // 8个周期计数
        end
    end

    // (1) 读入数据，每8个周期读入一次数据，延迟一个周期
    integer i, j;
    reg signed [11:0] xin_reg[15:0];
    always @ (posedge clk or posedge rst) begin
```

```
48          if (rst) begin
49              for (i=0; i<16; i=i+1) begin
50                  xin_reg[i] <= 12'b0;
51              end
52          end
53          else if (cnt == 3'd0 && en) begin       // 每8个周期读入一次有效数据
54              xin_reg[0] <= xin;
55              for (j=0; j<15; j=j+1) begin
56                  xin_reg[j+1] <= xin_reg[j];     // 数据移位
57              end
58          end
59      end
60
61      /* 3个周期 */
62      reg signed [11:0] add_a, add_b;
63      wire signed [12:0] add_s;
64      reg signed [11:0] coe_s[2:0];
65      wire [2:0] xin_index = cnt[2:0] >= 1 ? cnt[2:0] - 1 : 3'd7;
66      always @ (posedge clk or posedge rst) begin
67          if (rst) begin
68              add_a <= 12'b0;
69              add_b <= 12'b0;
70              coe_s[0] <= 12'b0;
71          end
72          else if (en_r[xin_index]) begin //from en_r[1]
73              add_a <= xin_reg[xin_index];
74              add_b <= xin_reg[15-xin_index];
75              coe_s[0] <= coe[xin_index];
76          end
77      end
78      always @(posedge clk or posedge rst) begin
79          if (rst) begin
80              coe_s[1] <= 12'b0;
81              coe_s[2] <= 12'b0;
82          end
83          else begin
84              coe_s[1] <= coe_s[0];
85              coe_s[2] <= coe_s[1];
86          end
87      end
88      c_addsub_0 adder (
89          .A(add_a),
90          .B(add_b),
91          .CLK(clk),
92          .S(add_s)
93      );
94
95      // (3) 乘法运算，使用一个流水线乘法器，3个周期
96      wire signed [24:0] mout;
97      /* 延迟3个时钟的流水线乘法器 */
98      mult_gen_0 mult (
99          .CLK(clk),  // input wire CLK
```

```
        .A(add_s),  // input wire [12 : 0] A
        .B(coe_s[2]),  // input wire [11 : 0] B
        .P(mout)    // output wire [24 : 0] P
    );

    // (4) 积分累加，延迟2个周期
    wire signed [27:0] sum;
    c_accum_0 your_instance_name (
        .B(mout),          // input wire [24 : 0] B
        .CLK(clk),         // input wire CLK
        .BYPASS(cnt==3'd7),// input wire BYPASS
        .Q(sum)            // output wire [27 : 0] Q
    );
    reg out_rdy_r;
    reg signed [27:0] yout_r;     // 输出数据
    always @(posedge clk) begin
        if (en_r[15]) begin
            out_rdy_r <= 1'b1;
            yout_r <= sum;
        end
        else begin
            out_rdy_r <= 1'b0;
        end
    end

    assign yout = yout_r;
    assign out_rdy = out_rdy_r;
endmodule
```

15 阶串行 FIR 滤波器的仿真文件如下:

Listing 8: 15 阶串行 FIR 滤波器的仿真文件

```
`timescale 1ns / 1ps

module fir_serial_tb;
    //input
    reg clk;
    reg rst;
    reg en ;
    reg [11:0] xin ;
    //output
    wire signed [27:0] yout ;
    wire out_rdy ;
    parameter SIN_DATA_NUM = 10400;

    initial begin
        clk = 1'b0;
        forever begin
            #5 clk = ~clk;
        end
    end

```

```
    parameter FILE_PATH_A = "E:/Projects/matlabProject/fpgaTools/extend/signal_data.txt";
    parameter FILE_PATH_B = "E:/Projects/matlabProject/fpgaTools/extend/signal_res.txt";

    initial begin
        rst = 1'b1;
        #30 rst = 1'b0;
    end
    reg [11:0] stimulus[0: SIN_DATA_NUM-1];
    integer i, j, file;
    initial begin
        $readmemb(FILE_PATH_A, stimulus);
        file = $fopen(FILE_PATH_B, "w");
        en = 0;
        i = 0;
        j = 0;
        xin = 0;
        #200;
        forever begin
            repeat(7) @(negedge clk); //空置7个周期，每过16个周期给数据
            en = 1;
            xin = stimulus[i];
            @(negedge clk);
            en = 0;          //输入数据有效信号只持续一个周期即可
            if (i == SIN_DATA_NUM-1)
                i = 0;
            else
                i = i + 1;
        end
    end
    always @(posedge out_rdy) begin
        $fwrite(file, "%d\n", yout);
        j = j + 1;
        if (j == SIN_DATA_NUM) begin
            $finish;
            $fclose(file);
        end
    end
    fir_serial fir_filter_inst (
        .rst(rst),
        .clk(clk),
        .en(en),
        .xin(xin),
        .out_rdy(out_rdy),
        .yout(yout)
    );
endmodule
```

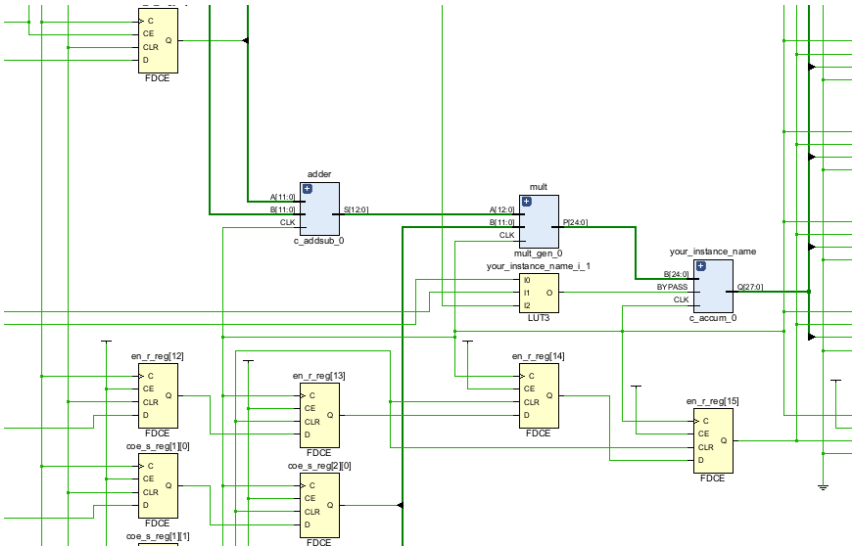


图 12: 串行 FIR 综合结果

15 阶串行 FIR 滤波器的总结结果如图所示 12所示, 仅使用了 1 个加法器、1 个乘法器以及一个累加器, 其数据输入要求间隔至少 8 个周期, 因为流水线以 8 个周期完成一次计算, 此外其数据相比于并行延迟较多, 同样的 DSP 单元, 延迟达到了 16 个周期。相比于串行设计用到了大量的查找表和触发器资源, 但也不到片上总资源的%0.5。

## 创新设计

用到的 16 倍过采样串口通信模块的源文件如下:

Listing 9: 16 倍过采样 UART 模块

```
/* 时钟模块 */
module uart_clk_div (
    input clk,
    input rstn,
    output clk_uart,
    output clk_uart_16
    );
    // localparam step_freq = 32'd6597070;     /* 9600 * 16 */
    localparam step_freq = 32'd79164837;    /* 115200 * 16 */

    reg [31:0] cnt = 32'b0;
    reg cnt_equal;
    always@(posedge clk or negedge rstn) begin
        if(!rstn)
            cnt <= 32'b0;
        else
            cnt <= cnt + step_freq;
    end
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            cnt_equal <= 1'b0;
        end else if(cnt < 32'h8000_0000) begin
```

```
            cnt_equal <= 1'b0;
        end else begin
            cnt_equal <= 1'b1;
        end
    end
    assign clk_uart_16 = cnt_equal;

    /* 16倍分频 */
    reg [3:0] cnt2 = 4'b0;
    always @(posedge clk_uart_16 or negedge rstn) begin
        if (!rstn) begin
            cnt2 <= 4'b0;
        end else begin
            cnt2 <= cnt2 + 1'b1;
        end
    end
    assign clk_uart = cnt2[3];
endmodule
/* 16倍过采样rx模块 */
module uart_rx(
    input rxd,
    input clk_uart_16,
    input rst,
    output rx_ne,
    output reg [7:0] data_i
    );
    reg [7:0] data_i_buf;

    // 接收器三个状态，格雷码
    localparam st_IDLE = 3'b000;      /* 空闲状态 */
    localparam st_confirm = 3'b001;   /* 确认接收 */
    localparam st_RECEIVE = 3'b011;   /* 接收数据 */
    localparam st_STOP = 3'b010;      /* 等待停止位 */
    localparam st_RXNE = 3'b110;      /* 接收非空 */
    reg [2:0] rx_cur_st, rx_nxt_st;
    reg [3:0] count1;
    reg [3:0] count2;
    /* 状态切换 */
    always @(posedge clk_uart_16 or posedge rst) begin
        if (rst) begin
            rx_cur_st <= st_IDLE;
        end
        else begin
            rx_cur_st <= rx_nxt_st;
        end
    end

    /* 过采样时钟计数 */
    always @(posedge clk_uart_16) begin
        if (rx_cur_st == st_confirm || rx_cur_st == st_RECEIVE) begin
            count2 <= count2 + 1'b1;
        end else begin
```

```
75              count2 <= 4'b0000;
76          end
77      end
78
79      reg sampleBuf1;     /* 采样缓存单元 */
80      reg sampleBuf2;     /* 采样缓存单元 */
81      reg sampleBuf3;     /* 采样缓存单元 */
82      always @(posedge clk_uart_16 or posedge rst) begin
83          if (rst) begin
84              {sampleBuf1, sampleBuf2, sampleBuf3} <= 3'b0;
85          end
86          else if (rx_cur_st == st_confirm) begin
87              if (rxd) begin
88                  sampleBuf1 <= sampleBuf1 | 1'b1;
89              end else begin
90                  sampleBuf1 <= sampleBuf1;
91              end
92          end else if (rx_cur_st == st_RECEIVE) begin
93              case (count2)
94                  4'd7: sampleBuf1 <= rxd;
95                  4'd8: sampleBuf2 <= rxd;
96                  4'd9: begin
97                      casez ({sampleBuf1, sampleBuf2, rxd})
98                          3'b11?, 3'b1?1, 3'b?11: begin
99                              sampleBuf3 <= 1'b1;
100                         end
101                         default: sampleBuf3 <= 1'b0;
102                     endcase
103                 end
104                 default: sampleBuf1 <= sampleBuf1;
105             endcase
106         end else begin
107             sampleBuf1 <= 1'b0;
108         end
109     end
110
111     always @(posedge clk_uart_16) begin
112         if (rx_cur_st == st_RECEIVE) begin
113             if (count2 == 4'd10) begin
114                 count1 <= count1 + 1'b1;
115             end else begin
116                 count1 <= count1;
117             end
118         end else begin
119             count1 <= 4'b0000;
120         end
121     end
122
123     always @(posedge clk_uart_16 or posedge rst) begin
124         if (rst) begin
125             data_i_buf <= 8'b0;
126         end
```

```
127         else if (rx_cur_st == st_RECEIVE) begin
128             if (count2 == 4'd10) begin
129                 data_i_buf <= {sampleBuf3, data_i_buf[7:1]};
130             end else begin
131                 data_i_buf <= data_i_buf;
132             end
133         end
134     end
135 
136     /* 状态切换 */
137     always @(*) begin
138         case (rx_cur_st)
139             st_IDLE: begin
140                 if (rxd == 1'b0) begin           /* 接收到开始信号 */
141                     rx_nxt_st <= st_confirm;
142                 end else begin
143                     rx_nxt_st <= st_IDLE;
144                 end
145             end
146             st_confirm: begin                    /* 确认起始信号 */
147                 if (count2 == 4'b1000) begin
148                     if (!sampleBuf1) begin
149                         rx_nxt_st <= st_RECEIVE;
150                     end else begin
151                         rx_nxt_st <= st_IDLE;
152                     end
153                 end else begin
154                     rx_nxt_st <= st_confirm;
155                 end
156             end
157             st_RECEIVE: begin                     /* 接收到数据 */
158                 if (count1 == 4'd9) begin
159                     rx_nxt_st <= st_STOP;
160                 end else begin
161                     rx_nxt_st <= st_RECEIVE;
162                 end
163             end
164             st_STOP: begin
165                 if (rxd) begin
166                     rx_nxt_st <= st_RXNE;
167                 end else begin
168                     rx_nxt_st <= st_STOP;
169                 end
170             end
171             st_RXNE: begin
172                 if (rxd == 1'b0) begin
173                     rx_nxt_st <= st_confirm;
174                 end else begin
175                     rx_nxt_st <= st_IDLE;
176                 end
177             end
178             default: rx_nxt_st <= st_IDLE;
```

```
179         endcase
180     end
181     always @(posedge rx_ne or posedge rst) begin
182         if (rst) begin
183             data_i <= 8'b0;
184         end
185         else begin
186             data_i <= data_i_buf;
187         end
188     end
189     /* 接收完成标志位 */
190     assign rx_ne = (rx_cur_st == st_RXNE)? 1'b1:1'b0;
191 endmodule
192 /* tx模块 */
193 module uart_tx(
194     input [7:0] data_o,
195     output reg txd,
196     input clk,
197     input rst,
198     input tx_en
199     );
200     // 发送机四个状态
201     localparam st_IDLE = 2'b00;      /* 空闲状态 */
202     localparam st_start = 2'b01;     /* 发送起始位 */
203     localparam st_data = 2'b10;      /* 发送数据 */
204     localparam st_end = 2'b11;       /* 发送结束 */
205 
206     reg [2:0] count;
207     reg [1:0] tx_cur_st, tx_nxt_st;
208     reg [7:0] data_o_tmp;
209 
210     reg tx_en_buf;
211     always @(posedge tx_en or posedge clk) begin
212         if (tx_en) begin
213             tx_en_buf <= 1'b1;
214         end else if (tx_cur_st == st_start) begin
215             tx_en_buf <= 1'b0;
216         end
217     end
218 
219     always @(*) begin                /* 组合逻辑 */
220         case (tx_cur_st)
221             st_IDLE: begin
222                 if (tx_en_buf) begin
223                     tx_nxt_st <= st_start;
224                 end else begin
225                     tx_nxt_st <= st_IDLE;
226                 end
227             end
228             st_start: begin
229                 tx_nxt_st <= st_data;
230             end
```

```
231             st_data: begin
232                 if (count == 7) begin
233                     tx_nxt_st <= st_end;
234                 end else begin
235                     tx_nxt_st <= st_data;
236                 end
237             end
238             st_end: begin
239                 if (tx_en_buf) begin
240                     tx_nxt_st <= st_start;
241                 end else begin
242                     tx_nxt_st <= st_end;
243                 end
244             end
245             default: tx_nxt_st <= st_IDLE;
246         endcase
247     end
248 
249     always @(posedge clk) begin     /* 时序逻辑 */
250         if (rst) begin
251             tx_cur_st <= st_IDLE;
252         end else begin
253             tx_cur_st <= tx_nxt_st;
254         end
255     end
256 
257     always @(posedge clk) begin
258         if (tx_cur_st == st_data) begin
259             count <= count + 1'b1;
260         end else begin
261             count <= 0;
262         end
263     end
264 
265     always @(posedge clk) begin
266         if (tx_cur_st == st_start) begin
267             data_o_tmp <= data_o;
268         end else if (tx_cur_st == st_data) begin
269             data_o_tmp <= {1'b0, data_o_tmp[7:1]};
270         end
271     end
272 
273     always @(posedge clk) begin
274         if (tx_cur_st == st_start) begin
275             txd <= 0;
276         end else if (tx_cur_st == st_data) begin
277             txd <= data_o_tmp[0];
278         end else begin
279             txd <= 1;
280         end
281     end
282 endmodule
```

基于串口通信的 FIR 硬件验证设计源文件如下:

Listing 10: 基于串口 FIR 硬件验证

```
module fir_top (
    input clk,      /* 100MHz时钟输入 */
    input rxd,      /* rxd接口 */
    output txd,     /* txd接口 */
    input rst       /* 复位信号 */
    );
    // 时钟模块
    wire clk_uart;
    wire clk_uart_16;
    uart_clk_div uart_clk_div (
        .clk(clk),
        .rstn(!rst),
        .clk_uart(clk_uart),
        .clk_uart_16(clk_uart_16)
    );
    // 串口接收模块
    wire [7:0] data_rx;
    wire rx_ne;
    uart_rx uart_rx1 (
        .clk_uart_16(clk_uart_16),
        .rxd(rxd),
        .rst(rst),
        .data_i(data_rx),
        .rx_ne(rx_ne)
    );
    // 串口发送模块
    reg [7:0] data_tx;
    reg tx_en;
    uart_tx uart_tx1 (
        .clk(clk_uart),
        .txd(txd),
        .rst(rst),
        .data_o(data_tx),
        .tx_en(tx_en)
    );
    wire out_rdy;
    wire [27:0] yout;
    reg [11:0] xin;
    reg data_en;
    fir_parallel_15  fir_parallel_15_inst (
        .rst(rst),
        .clk(clk_uart_16),
        .en(data_en),
        .xin(xin),
        .out_rdy(out_rdy),
        .yout(yout)
    );
```

```
48
49      /* 一段式状态机 */
50      reg cnt_rx;
51      always @(posedge clk_uart_16 or posedge rst) begin
52          if (rst) begin
53              xin <= 12'b0;
54              cnt_rx <= 2'b0;
55              data_en <= 1'b0;
56          end
57          else if (cnt_rx == 1'b1) begin
58              if (rx_ne) begin
59                  xin <= {xin[11-:4], data_rx};
60                  cnt_rx <= 1'b0;
61                  data_en <= 1'b1;
62              end
63          end
64          else if(rx_ne && data_rx[7:4] == 4'b1010) begin
65              xin <= {data_rx[3:0], 8'b0};
66              cnt_rx <= 1'b1;
67              data_en <= 1'b0;
68          end
69          else begin
70              cnt_rx <= 1'b0;
71              data_en <= 1'b0;
72          end
73      end
74
75      reg [9:0] cnt_tx; /* 2^5 + 2^4 */
76      reg cnt_tx_en;
77      reg [31:0] data_out_buf;
78      always @(posedge clk_uart_16 or posedge rst) begin
79          if (rst) begin
80              cnt_tx_en <= 1'b0;
81              cnt_tx <= 5'b0;
82              data_out_buf <= 32'b0;
83          end
84          else if (out_rdy) begin
85              data_out_buf <= {4'b1010, yout};
86              cnt_tx_en <= 1'b1;
87              cnt_tx <= 10'b1;
88          end
89          else if (cnt_tx == 10'b0) begin
90              cnt_tx_en <= 1'b0;
91          end
92          else if (cnt_tx_en == 1'b1) begin
93              if (cnt_tx[7:0] == 8'b1110_0000) begin
94                  tx_en <= 1'b1;
95                  data_tx <= data_out_buf[31-:8];
96                  data_out_buf <= {data_out_buf[24:0], 8'b0};
97              end
98              else begin
99                  tx_en <= 1'b0;
```

```
            end
            cnt_tx <= cnt_tx + 5'd1;
        end
    end
endmodule
```

基于串口通信的 FIR 硬件验证设计仿真文件如下:

Listing 11: 基于串口 FIR 硬件验证的仿真文件

```
module fir_top_tb;
    reg  clk;
    wire rxd;
    wire txd;
    reg  rst;
    reg  clk_uart;
    reg  clk_uart_16;
    parameter SIN_DATA_NUM = 10400;
    parameter FILE_PATH_A = "E:/Projects/matlabProject/fpgaTools/innovation/signal_data.txt";
    parameter FILE_PATH_B = "E:/Projects/matlabProject/fpgaTools/innovation/signal_res.txt";
    /* 复位 */
    initial begin
        rst = 1;
        #30 rst = 0;
    end
    /* 时钟生成 */
    initial begin
        clk = 1'b0;
        forever begin
            #5 clk = ~clk;
        end
    end
    initial begin
        clk_uart_16 = 0;
        forever begin
            #271 clk_uart_16 = ~clk_uart_16;
        end
    end
    initial begin
        clk_uart = 0;
        forever begin
            #4336 clk_uart = ~clk_uart;
        end
    end

    fir_top fir_top_inst (
        .clk(clk),
        .rxd(rxd),
        .txd(txd),
        .rst(rst)
    );

    reg  tx_en;
```

```
    reg [7:0] data_o;
    uart_tx uart_tx (
        .clk(clk_uart),
        .txd(rxd),
        .rst(rst),
        .data_o(data_o),
        .tx_en(tx_en)
    );
    wire rx_ne;
    wire [7:0] data_i;
    uart_rx uart_rx (
        .rst(rst),
        .clk_uart_16(clk_uart_16),
        .rxd(txd),
        .data_i(data_i),
        .rx_ne(rx_ne)
    );

    reg [11:0] stimulus[0: SIN_DATA_NUM-1];
    integer i;
    initial begin
        $readmemb(FILE_PATH_A, stimulus);
        #100;
        for (i = 0; i<SIN_DATA_NUM; i = i+1) begin
            data_o <= {4'b1010, stimulus[i][11-:4]};
            @(negedge clk_uart) tx_en <= 1'b1;
            @(posedge clk_uart) tx_en <= 1'b0;
            repeat(8)@(posedge clk_uart);
            data_o <= {stimulus[i][7:0]};
            @(negedge clk_uart) tx_en <= 1'b1;
            @(posedge clk_uart) tx_en <= 1'b0;
            repeat(4)@(posedge rx_ne);
        end
    end
    reg signed [27:0] data_rx;
    integer j;
    integer k;
    integer file;
    initial begin
        #100;
        file = $fopen(FILE_PATH_B, "w");
        for (j = 0; j<SIN_DATA_NUM; j = j + 1) begin
            @(posedge rx_ne) data_rx[27:24] = data_i[3:0];
            while (data_i[7:4] != 4'b1010) begin
                @(posedge rx_ne) data_rx[27:24] = data_i[3:0];
            end
            @(posedge rx_ne) data_rx[23:16] = data_i;
            @(posedge rx_ne) data_rx[15:8]  = data_i;
            @(posedge rx_ne) data_rx[7:0]   = data_i;
            $fwrite(file, "%d\n", data_rx);
        end
        $finish;
```

```
96          $fclose(file);
97      end
98  endmodule
```

创新设计部分的约束文件如下:

Listing 12: 创新设计部分的约束文件

```
1  # 100MHz时钟信号
2  set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports clk]
3  # 复位信号
4  set_property -dict {PACKAGE_PIN R11 IOSTANDARD LVCMOS33} [get_ports rst]
5  # 串口
6  set_property -dict {PACKAGE_PIN N5 IOSTANDARD LVCMOS33} [get_ports rxd]
7  set_property -dict {PACKAGE_PIN T4 IOSTANDARD LVCMOS33} [get_ports txd]
```

# 三、仿真结果

## 基础任务

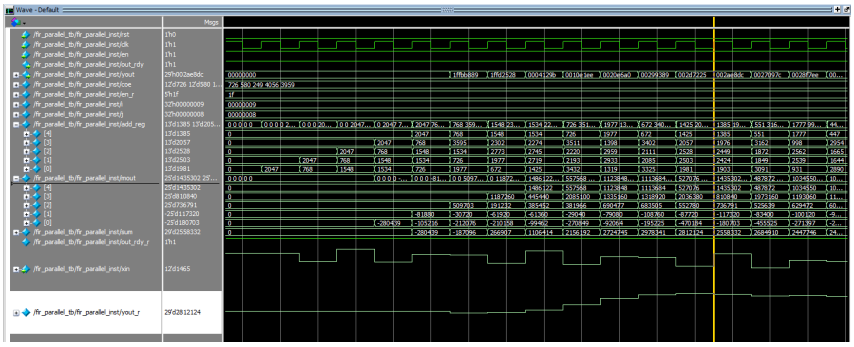

图 13: 数据流过程仿真

针对输入数据计算流程的仿真结果如图 13所示, 当数据 en 有效时, 开始读入 xin 数据, 数据读入后看以看到, 其在延迟两个时钟后由加法器输出到对应的 add_reg 中, 然后数据开始不为 0, 乘法器开始工作, 从 add_reg 不为 0 开始, 延迟 3 时钟后乘法器开始输出, 再延迟一个时钟, 最后得到累加结果。

此后, 流水线进入填满状态, 每一个时钟输出一个有效数据。

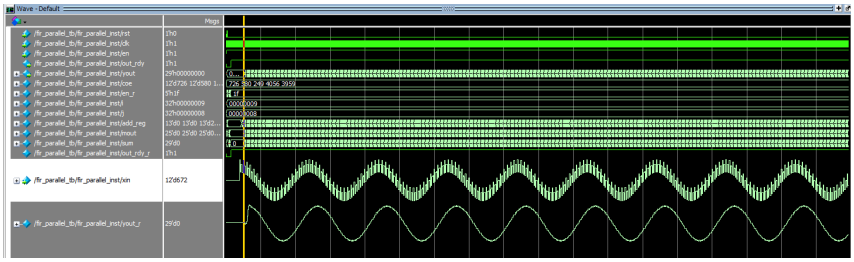

图 14: 滤波结果仿真

滤波结果如图 14所示, 输入信号如下:

```
fs = 5e3;
T = 1 / fs;
n = 0:2047;
t = n * T;
x = cos(2*pi*50*t) + 0.5*cos(2*pi*2000*t);
```

将 50Hz 的信号视为目标信号, 2000Hz 的信号视为单频噪声, 可以看到通过 1k 截止频率的低通 FIR 滤波器后其信噪比明显得到改善, 2000Hz 信号被削弱。

## 提高任务

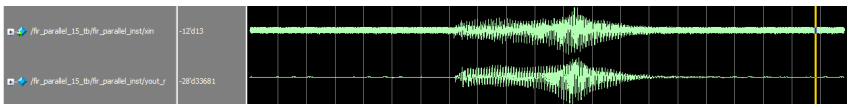

图 15: 滤波结果仿真

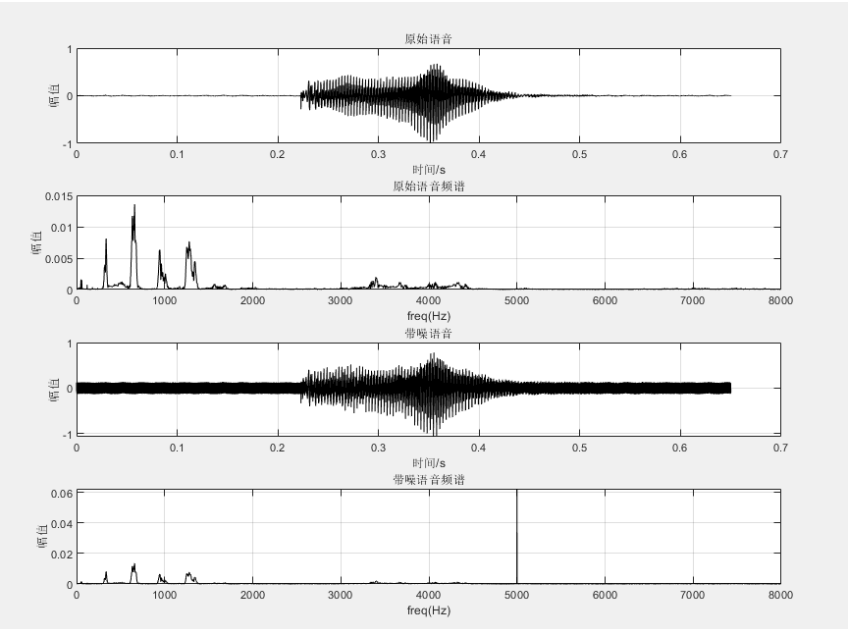


图 16: 音频信号理论情况

对输入信号, 即添加单频噪声的音频信号, 其时域图和频域图如图 16所示, 滤波结果仿真如图 15所示, 可以看到滤波结果的时域非常接近原始音频信号, 成功将音频信号中添加的单频噪声滤除。

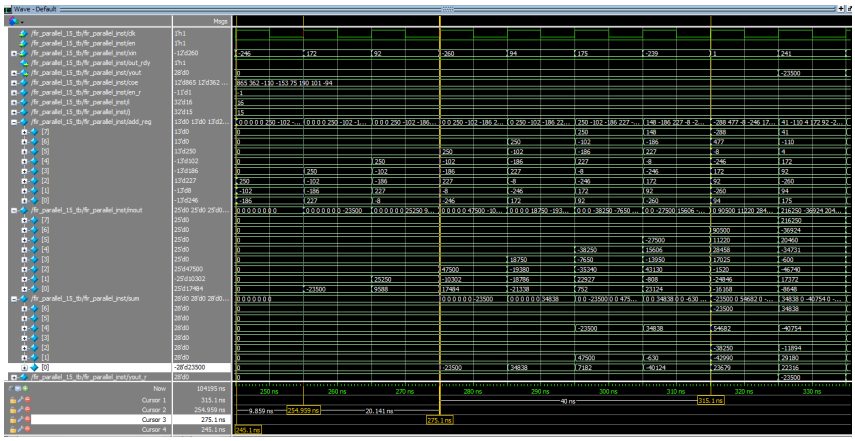

图 17: 数据流仿真

针对数据流的仿真如图 17所示, 从输入数据开始, 延迟两个时钟后, 得到 add_reg 中的结果, 之后延迟三个时钟后乘法器完成输出, 最后的累加积分的加法器也进行了优化, 因为使用了三级的延迟两个时钟的 DSP Slice 中的加法器单元, 累加求和延迟 6 个时钟, 整套计算系统为流水线模式, 经过 11 个时钟后, 流水线被填满之后, 每个时钟输出一个有效数据。

## 拓展任务

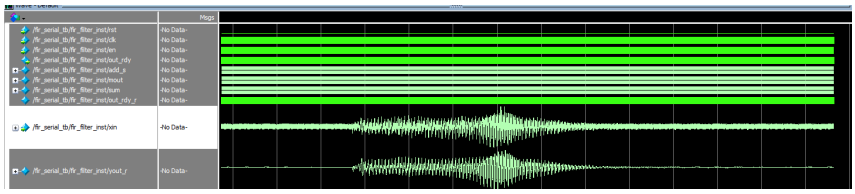

图 18: 仿真效果图

滤波效果仿真结果如图 18所示, 滤波效果理想, 可以看到加噪信号中的单频噪声被滤除。

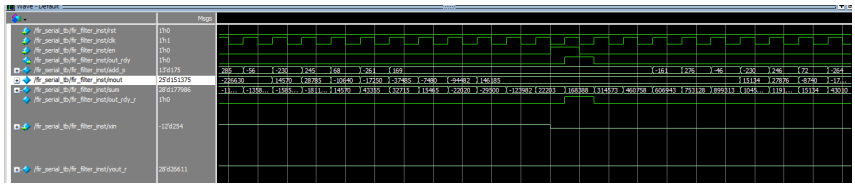

图 19: 时序仿真图

串行设计的 FIR 滤波器时序仿真如图 19所示, 可以看到其与并行设计的最大差异也直接体现在仿真上, 不同的计算结果在并行 FIR 滤波器中体现为不同的数据, 不同的连线, 而在串行设计中只需要用到一个加法器, 一个乘法器和一个累加器, 计算过程中的 8 次加法通过时分复用同一个模块实现, 因此导致了较长的计算延迟, 使用 8 个时钟计算一次数据输出。图 19中可以看到, 不同的时钟周期, 模块的输出都不同, 而这些不同的数据属于一次计算, 最终比对计算结果, 可以看到其计算结果与并行设计的相同系数的 FIR 数值一样。

### 创新设计

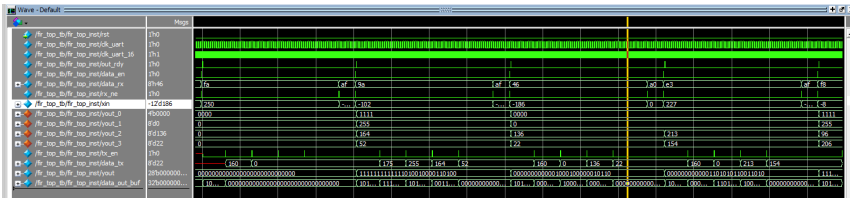

图 20: 创新设计的仿真结果

创新设计部分的仿真结果如图 20所示, 可以看到数据通过串口发送到该模块后, 控制电路部分会等待有起始信号 1010 的数据, 同步输入数据, 每接收到两个数据有效数据后, 去除 1010 标识符后送入 FIR 滤波器, 当 FIR 滤波器输出时, 将数据缓存, 分 8 个 bit 通过串口的 tx 模块发送, 当上位机接收后, 再次传入数据, 之后计算完毕。

## 四、Matlab 验证结果

### 基础任务

使用 Matlab 进行验证的程序如下:

Listing 13: 基础任务 Matlab 验证结果

```
clear all; clc; close all;
fs = 5e3;
T = 1 / fs;
n = 0:2047;
t = n * T;
x = cos(2*pi*50*t) + 0.5*cos(2*pi*2000*t);
x = mapminmax(x);
x_dis = floor((2 ^ 11 - 1) * x);

b = [-137   -40   249   580   726   580   249   -40  -137];

x = x_dis;
y = round(filter(b, 1, x_dis));
time = t;
N = 2048;
% 读取FPGA滤波结果数据z
load res.mat
z = VarName1;
% 作图
subplot 611; plot(time,x,'k'); grid;
title('原始信号'); ylabel('幅值'); xlabel('时间/s');

subplot 612; audioFFT = fft(x);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('原始信号频谱'); ylabel('幅值'); xlabel('freq(Hz)');
```

```
26
27 subplot 613; plot(time,y,'k');grid;
28 title('Matlab滤波时域'); ylabel('幅值'); xlabel('时间/s');
29
30 subplot 614; audioFFT = fft(y);
31 plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
32 title('Maltab滤波频域'); ylabel('幅值'); xlabel('freq(Hz)');
33
34 subplot 615; plot(time,z,'k'); grid;
35 title('FPGA滤波后时域'); ylabel('幅值'); xlabel('时间/s');
36
37 subplot 616; audioFFT = fft(z);
38 plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
39 title('FPGA滤波后频域'); ylabel('幅值'); xlabel('freq(Hz)');
```

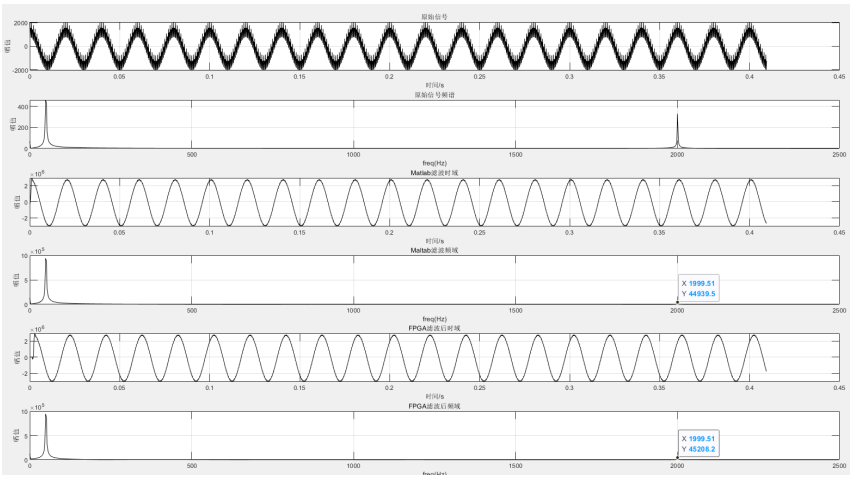


图 21: 8 阶并行 FIR 滤波器滤波结果

通过 Matlab 计算比对滤波结果, 结果如图 21所示, FIR 低通滤波器的效果基本实现, 其中数据略微不同是由于 FPGA 滤波器将开始滤波时的无效数据输出, 导致了二者微小的差距。

## 提高任务

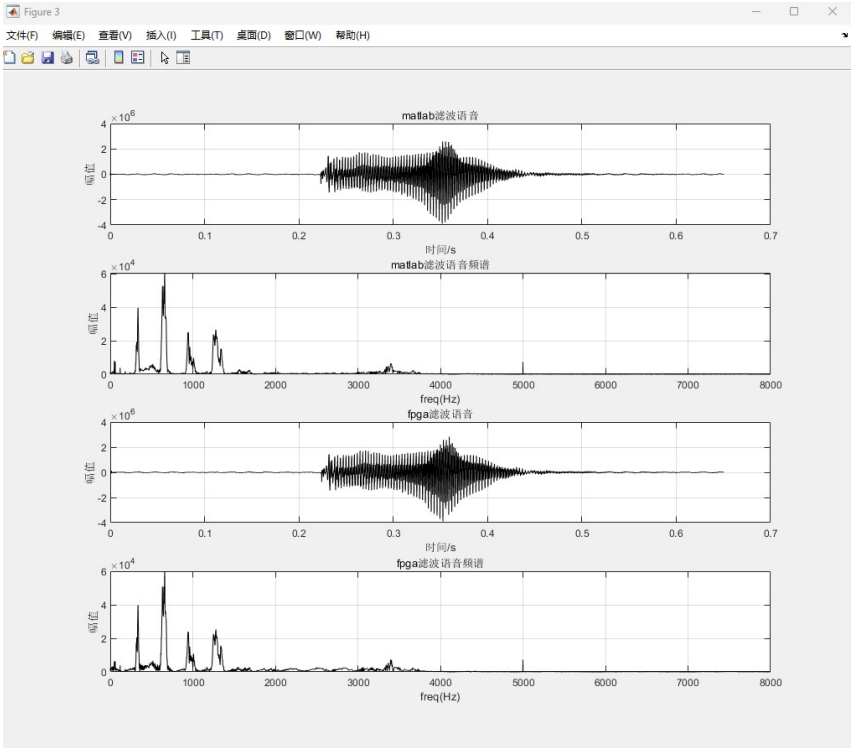


图 22: Matlab 验证结果

通过 Matlab 将计算结果比对, 结果如图 22所示, 效果很好的实现, 可以从途中观察到数值有略微差距, 通过分析 FPGA 仿真输出的数据可以发现问题, 在进行数据输入前, 数据输入使能便已经置位, 导致输出数据有一部分前导零, 采样固定点数截止便会导致会丢失末尾几个滤波结果, 导致了滤波结果的细微差异。

## 拓展任务

15 阶滤波器对语音信号的 Matlab 滤波结果验证程序如下:

Listing 14: 语音信号滤波验证程序

```
[file, path]  = uigetfile('*.wav');      % 读取音频
[xx, fs] = audioread([path, file]);      % 读取音频

xx=xx-mean(xx);                          % 消除直流分量
x=xx/max(abs(xx));                       % 幅值归一化
N=length(x);                             % 计算音频长度

time=(0:N-1)/fs;                         % 计算时间
noiseFreq=5000;                          % 单频噪声频率
noiseAmp=0.125;                          % 单频噪声幅度
signal=x + noiseAmp*cos(2*pi*noiseFreq*time)';      % 添加噪声
x_dis = floor((2 ^ 11 - 1) * mapminmax(signal));    % 引入量化噪声
b = [-94   101   190    75  -153  -110   362   865   865   362  -110  -153    75   190   101   -94];
x = mapminmax(x_dis);
```

```
x = filter(b, 1, x);

load('res.mat');
y = mapminmax(res);
% 作图
subplot 411; plot(time,x,'k'); grid;
title('Matlab滤波信号时域'); ylabel('幅值'); xlabel('时间/s');

subplot 412; audioFFT = fft(x);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('Matlab滤波信号频域'); ylabel('幅值'); xlabel('freq(Hz)');

subplot 413; plot(time,y,'k');grid;
title('FPGA滤波时域'); ylabel('幅值'); xlabel('时间/s');

subplot 414; audioFFT = fft(y);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('FPGA滤波频域'); ylabel('幅值'); xlabel('freq(Hz)');
```

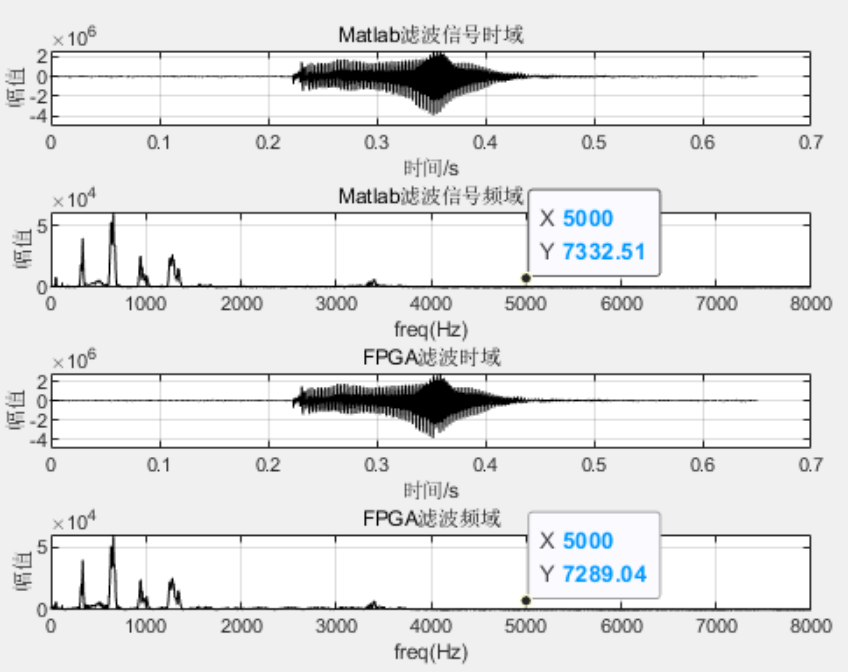


图 23: 15 阶串行 FIR 滤波器验证结果

15 阶串行 FIR 的滤波结果与 15 阶并行的 FIR 滤波器滤波结果一致, 相比之下没有并行 FIR 滤波器输出的无效数据, 其值更加接近 Matlab 的滤波结果, 认为二者效果相同, 从图中也可以明显的观察到信号为发生相位失真, 具备 FIR 滤波器的特性。

## 五、创新内容的硬件验证结果

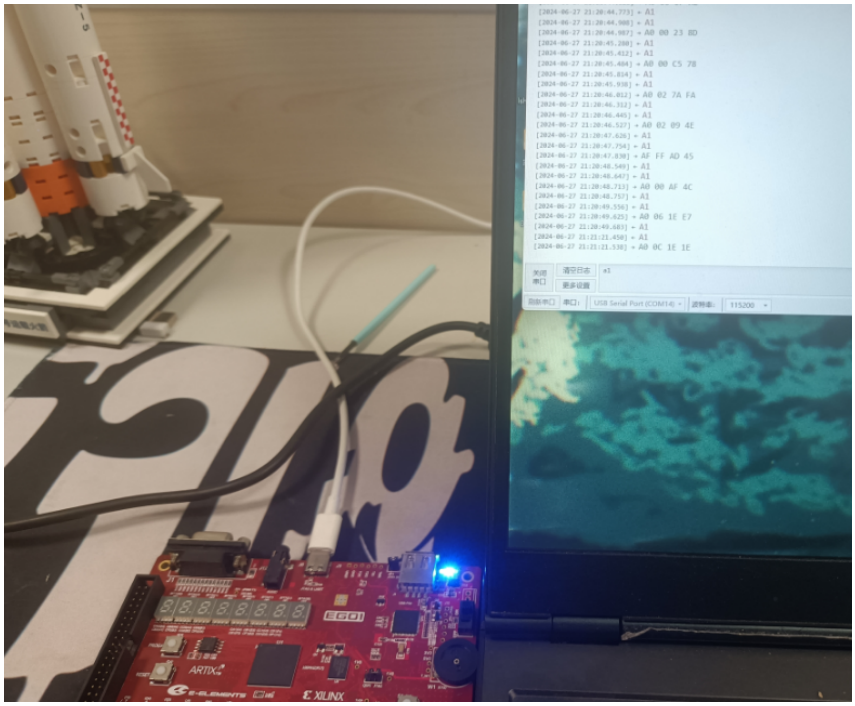

图 24: 基于串口的 FIR 滤波器设计验证实验结果

串口 FIR 硬件效果如图 24所示, 输入数据为 12bit, 输出数据为 28bit, 串口使用 8bit 进行通信, 在输入数据和输出数据前添加 4bit 的前同步码, ”1010”, 即图 24中串口调试助手显示的十六进制数据 0xA,PC 上位机向 FPGA 传输两个数据后, 输入数据达到 12bit, 进行一次计算, 输出 4 个 8bit 结果, 其中包含 28bit 数据。

针对创新部分的串口硬件验证过程, 设计 MATLAB 脚本程序, 通过串口实现 FIR 滤波, 其程序如下:

Listing 15: Matlab 通过串口与 FPGA 的 FIR 传输数据并验证

```
clear all;
hserial = serialport("COM14", 115200, "ByteOrder", "big-endian");
flush(hserial);

[file, path]  = uigetfile('*.wav');              % 读取音频
[xx, fs] = audioread([path, file]);              % 读取音频

xx = xx-mean(xx);                                % 消除直流分量
x = xx/max(abs(xx));                             % 幅值归一化
N = length(x);                                   % 计算音频长度
time=(0:N-1)/fs;                                 % 计算时间
noiseFreq=5000;                                  % 单频噪声频率
noiseAmp=0.125;                                  % 单频噪声幅度
signal=x + noiseAmp*cos(2*pi*noiseFreq*time)';   % 添加噪声

```

```
q_12 = quantizer('Format', [12, 0]);
res = mapminmax(signal);                          % 归一化数据
x_dis_1 = floor((2 ^ 11 - 1) * res);              % 数据量化
x_dis_2 = num2bin(q_12, x_dis_1);
x_dis_bin = strcat(repmat("1010", N, 1), x_dis_2);   % 添加1010标识

q_8 = quantizer('Format', [8, 0], 'DataMode', 'ufixed');
q_32 = quantizer('Format', [32, 0]);
signal_filtered = zeros(1, N);
for index = 1:N
    str = x_dis_bin(index);
    data_out = bin2num(q_8, extractBefore(str, 9));
    write(hserial, data_out, "uint8");
    data_out = bin2num(q_8, extractAfter(str, 8));
    write(hserial, data_out, "uint8");
    data = read(hserial, 1, "uint32");
    % 符号位扩展
    for k = 0:3
        data = bitset(data, 32-k, bitget(data, 28));
    end
    % 识别补码，转化为符号数输出
    signal_filtered(index) = bin2num(q_32, dec2bin(data));
end
save result
% load result.mat
% 作图
xx = signal - mean(signal);
signal = xx/max(abs(xx));
xx = signal_filtered - mean(signal_filtered);
signal_filtered = xx/max(abs(xx));
subplot 611; plot(time,x,'k'); grid;
title('原始语音'); ylabel('幅值'); xlabel('时间/s');

subplot 612; audioFFT = fft(x);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('原始语音频谱'); ylabel('幅值'); xlabel('freq(Hz)');

subplot 613; plot(time,signal,'k');grid;
title('带噪语音'); ylabel('幅值'); xlabel('时间/s');

subplot 614; audioFFT = fft(signal);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('带噪语音频谱'); ylabel('幅值'); xlabel('freq(Hz)');

subplot 615; plot(time,signal_filtered,'k'); grid;
title('FPGA滤波后时域'); ylabel('幅值'); xlabel('时间/s');

subplot 616; audioFFT = fft(signal_filtered);
plot((0:round(N/2)-1) / N * fs, abs(audioFFT(1:round(N/2)) / N), 'k');grid;
title('FPGA滤波后频域'); ylabel('幅值'); xlabel('freq(Hz)');
```

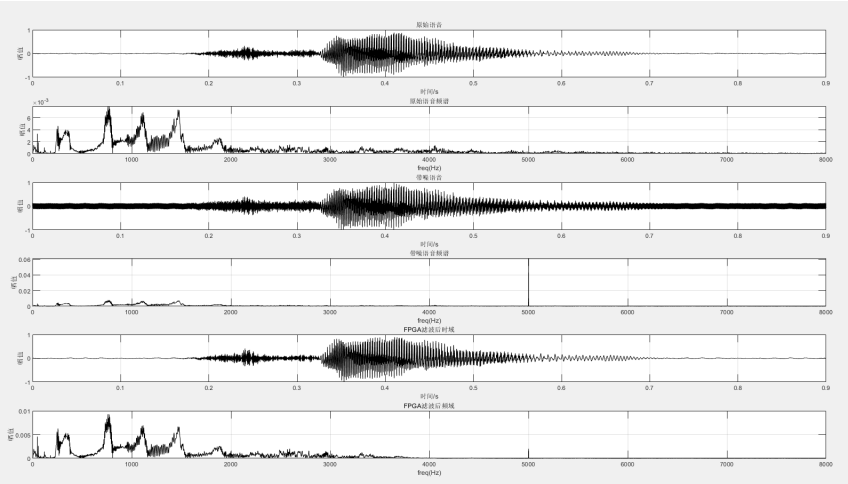

图 25: FPGA 硬件验证结果

通过串口将待滤波数据传输到 FPGA 开发板上, 并通过串口读取其回传的滤波结果, 之后绘制其时域和频域图, 得到如图 25所示的结果, 改滤波器的效果显著, 且无线性相位失真发生, 效果明显, 证明了在提高任务和拓展任务中使用 xilinx 提供的 DSP48E 的 IP 核设计的 FIR 滤波器可以在硬件上有效的工作, 不仅仅局限于仿真结果。

## 六、问题解决

### Vivado 的仿真过程中经常出现 UI 问题

解决: Vivado 的仿真界面虽然美观但是经常出现波形上数字消失, 而且模拟波形可以随意放缩比例尺, 不利于固定图像比例比对不同波形, 不利于观察波形, 最后简单的仿真过程, 如本次设计中的简单时序问题通过 Vivado 仿真, 但是复杂的如模拟波形数据, 流水线数据等通过 Modelsim 进行仿真。

### Modelsim 和 Vivado 的联合仿真, 在修改文件后需要重启

解决: 通过查阅资料查看原理, 了解到 Vivado 在调用 Modelsim 仿真时是将文件交付 Modelsim 后, 其会对仿真文件和源文件进行编译, 然后再进行仿真, 并且编译结果是静态的, 仅仅重新编译还不够, 还需要手动更新。Vivado 交付给 Modelsim 的源文件保存在 Modelsim 中的 xil_defaultlib 库中, 更改文件后找到对应文件 update 以后再进行 recompile, 之后 reset 仿真, 再次仿真就是更改后的仿真结果。

### 串行 FIR 滤波器的设计过程无论如何改动, 结果都不正确

解决: 花费很长的时间观察仿真时序图, 起初未考虑到加法器的 2 周期时延与滤波器系数直接赋值无时延导致的不同步, 一直认为是累加器的累加过程中 BYPASS 信号控制不正确, 最后实在无法定位问题, 信号输入开始, 通过笔算的形式手动计算 FIR 结果和流程, 最终将问题定位到滤波器系数提

前加法器输出两个周期的问题, 通过缓存移位寄存器人为将滤波器系数的赋值延迟 2 个周期, 解决问题, 最终得到与并行 FIR 滤波器完全一致的计算结果。

## 七、心得体会

作为最后的创新设计实验, 追求创新的前提下是比综合设计更加进一步复杂的设计过程。在本次创新设计中由于设计的是 FIR 滤波器, 相比于乘法器或者求解最大公倍数等等简单的算法而言, FIR 滤波器的算法原理上不复杂, 但随着阶数的上升其结构会极度的复杂。

如果直接在算法中使用"*" 或者"+" 会使得设计过程十分简单, 但如果通过仿真的方式验证的话, 仿真工具不会考虑乘法和加法得综合结果, 直接以计算机语言得形式进行计算仿真, 只要结构正确, 就可以得到正确的结果, 此外, 直接使用 * 和 + 的综合结果是未知的, 在未指定的情况下, 甚至无法正确处理符号数

在实际的设计中, 会使用并行或者串行加上流水线形式得乘法器和加法器进行设计。比如, 对滤波器输出得最后累加积分过程中, 并行设计的输出如果直接使用 7 个"+" 将 8 个数据相加, 综合工具甚至会将该累加过程综合成" 一群"LUT 构成的组合逻辑, 在实际应用中如此庞大得组合逻辑电路肯定无法完全避免竞争冒险等毛刺问题, 以至于最终设计得到得 FIR 滤波器只能在很低的工作频率下运行, 甚至有可能出错。由于创新设计得时间有限, 采用 Xilinx FPGA 上集成的 DSP 模块中的乘法器和累加器实现 FIR 滤波器, 实现设计安全和高效的均衡。

在设计阶段, 花费了将近 3/4 的时间用于仿真, 仿真能力得到了极大的提升, 从最开始的找不到仿真方向, 到现在可以熟练的联合 Matlab 和 Modelsim 等仿真工具, 熟悉了 Verilog 仿真文件中的若干系统级函数, 也对 Modelsim 仿真工具更加熟练。

最后不满足于仅仅通过仿真实现 FIR 滤波器, 尝试在硬件上验证所设计 FIR 滤波器的有效性, 但局限于信号发生器、示波器等的设备问题, 放弃了 XADC 采样信号的并通过 DAC 再生的验证思路, 着眼于使用串口将 FPGA 与上位机连接, FPGA 作为一个独立的计算单元供上位机调用, 上位机通过 Matalb 程序验证 FPGA 输出的计算结果是否正确。在该创新内容的设计过程中, 掌握了很多新的能力, 如 Matlab 对系统串口资源的使用, Matlab 中补码的整数的转换等, 以及更加复杂的仿真文件的编写等。

通过对数字滤波器和 Matlab 的设计, 不仅加深了自己对于 FPGA 技术的理解, 同时巩固了自己所学过的理论知识。