南京邮电大学

实 验 报 告

实验名称__基于 FPGA 的快速傅立叶变换设计__

指导教师_____孙科学_____

班级学号_____B08020221_____

姓　　名_____孟祥熙_____

开课时间 2010/2011 学年，　　第 二 学期

实验目的

（1）用 matlab 仿真 128 点 FFT
（2）用 Verilog 语言在 FPGA 上实现 FFT 算法

实验内容

1、用 matlab 仿真 128 点 FFT

1， 仿真方法：通过 Matlab 中想用函数完成对输入信号的快速傅里叶变换，并观测方根频谱，功率频谱，对数频谱。并用 IFFT 函数还原并输出波形，观测输入输出是否相同。

2，仿真环境：Matlab 2010a

3，函数介绍：
Y=fft(x,n,dim)：当 X 是一个向量时，返回对 X 的离散傅里叶变换；当 X 是一个矩阵时，返回一个矩阵并送 Y，其列是对 X 的列的离散傅里叶变换。
Y=ifft（X，n，dim)：函数对 X 进行离散傅里叶逆变换，其中 X，N，dim 的意义及用法和离散傅里叶变换函数 fft 完全相同

4，m 程序及图示如下：

```
f=(0:length(y)-1)'*fs/length(y);%进行对应的频率转换
figure(2);
%subplot(232);
plot(f,mag);%做频谱图
axis([0,100,0,80]);
xlabel('频率(Hz)');
ylabel('幅值');
title('正弦信号幅频谱图');
grid;
```

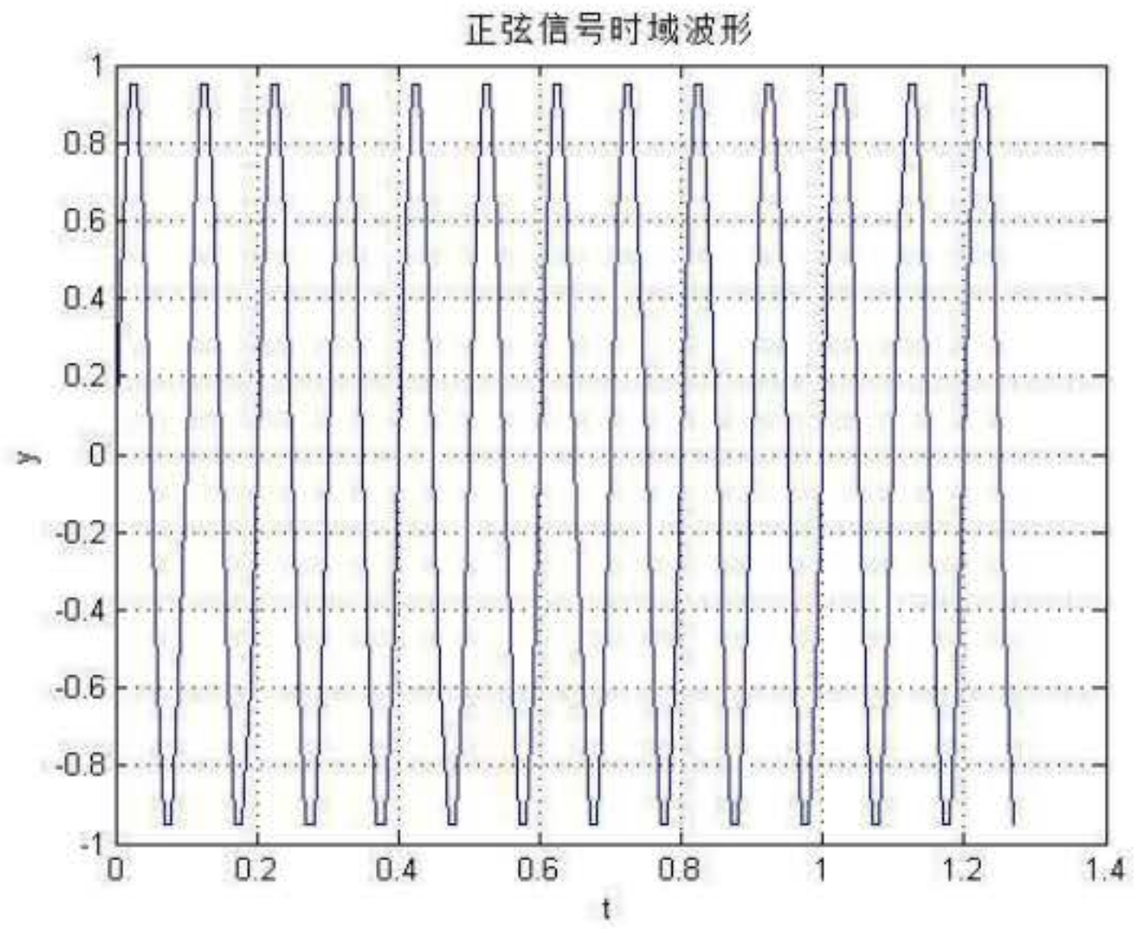


图 1 正弦信号

```
%进行 FFT 变换并做频谱图
y=fft(x,N);%进行 fft 变换
mag=abs(y);%求幅值
f=(0:length(y)-1)'*fs/length(y);%进行对应的频率转换
figure(2);
%subplot(232);
plot(f,mag);%做频谱图
axis([0,100,0,80]);
xlabel('频率(Hz)');
ylabel('幅值');
title('正弦信号幅频谱图');
grid;
```

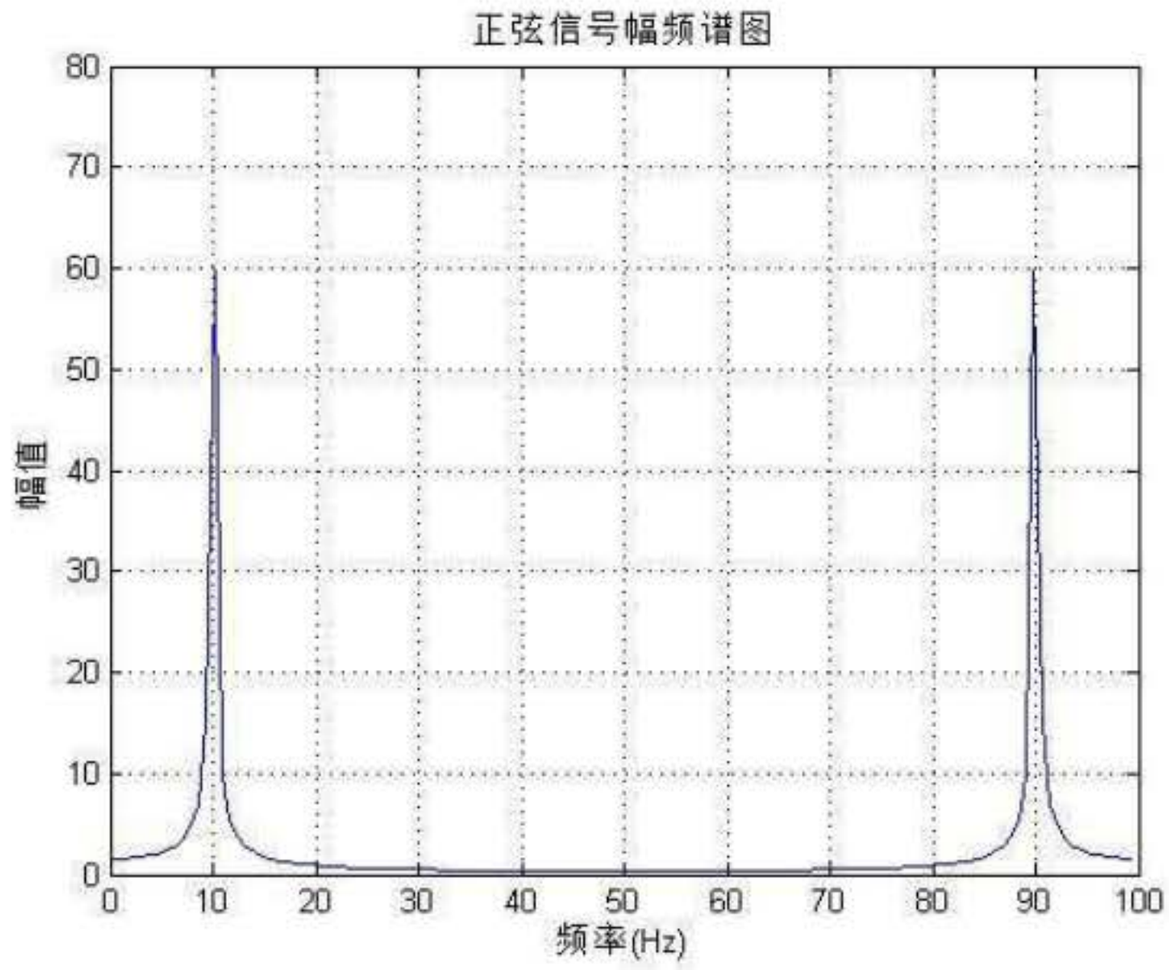


图 2 幅频谱图

```
%求均方根谱
sq=abs(y);
figure(3);
plot(f,sq);
xlabel('频率(Hz)');
ylabel('均方根谱');
title('正弦信号均方根谱');
grid;
```

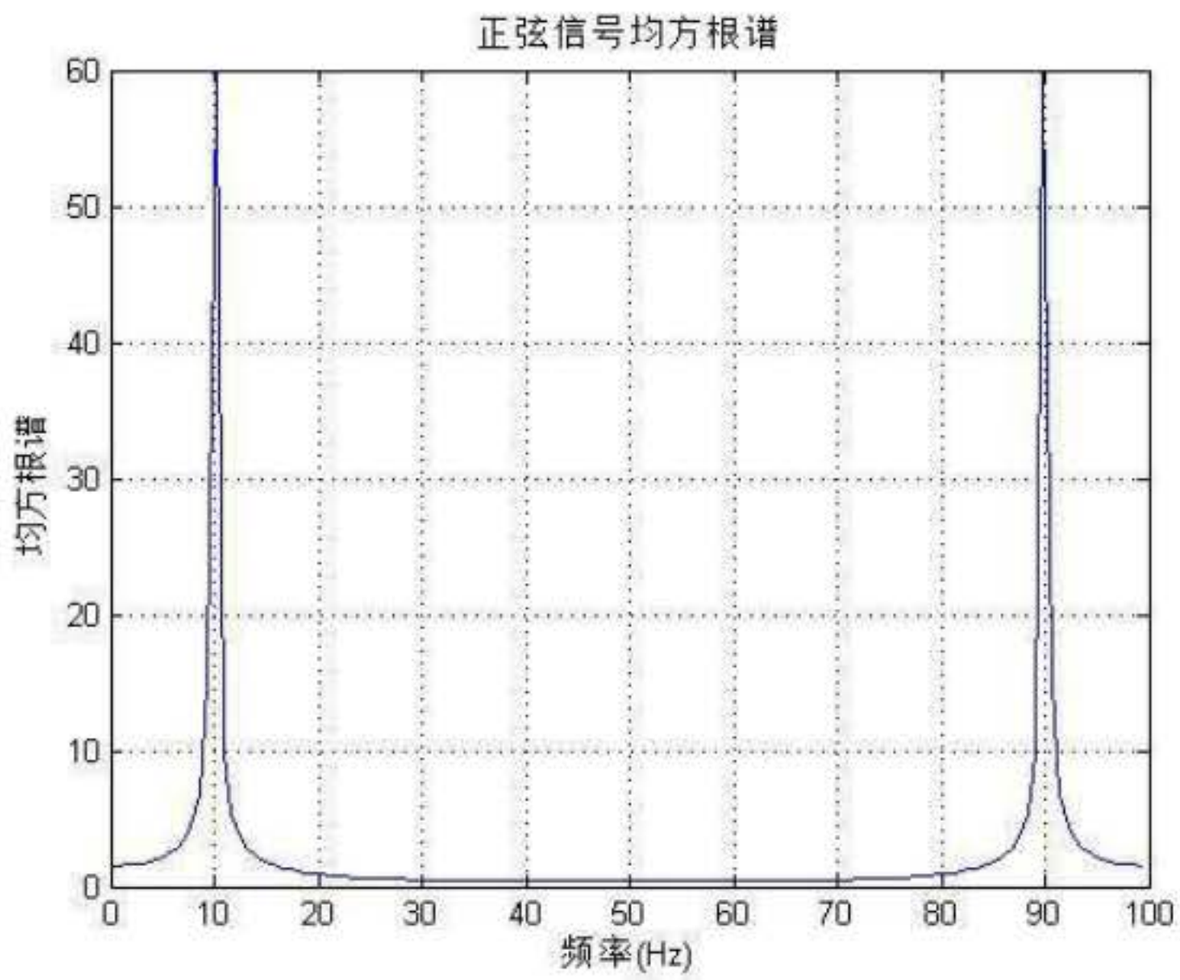


图 3 均方根谱

```
%求功率谱
power=sq.^2;
figure(4);
plot(f,power);
xlabel('频率(Hz)');
ylabel('功率谱');
title('正弦信号功率谱');
grid;
```

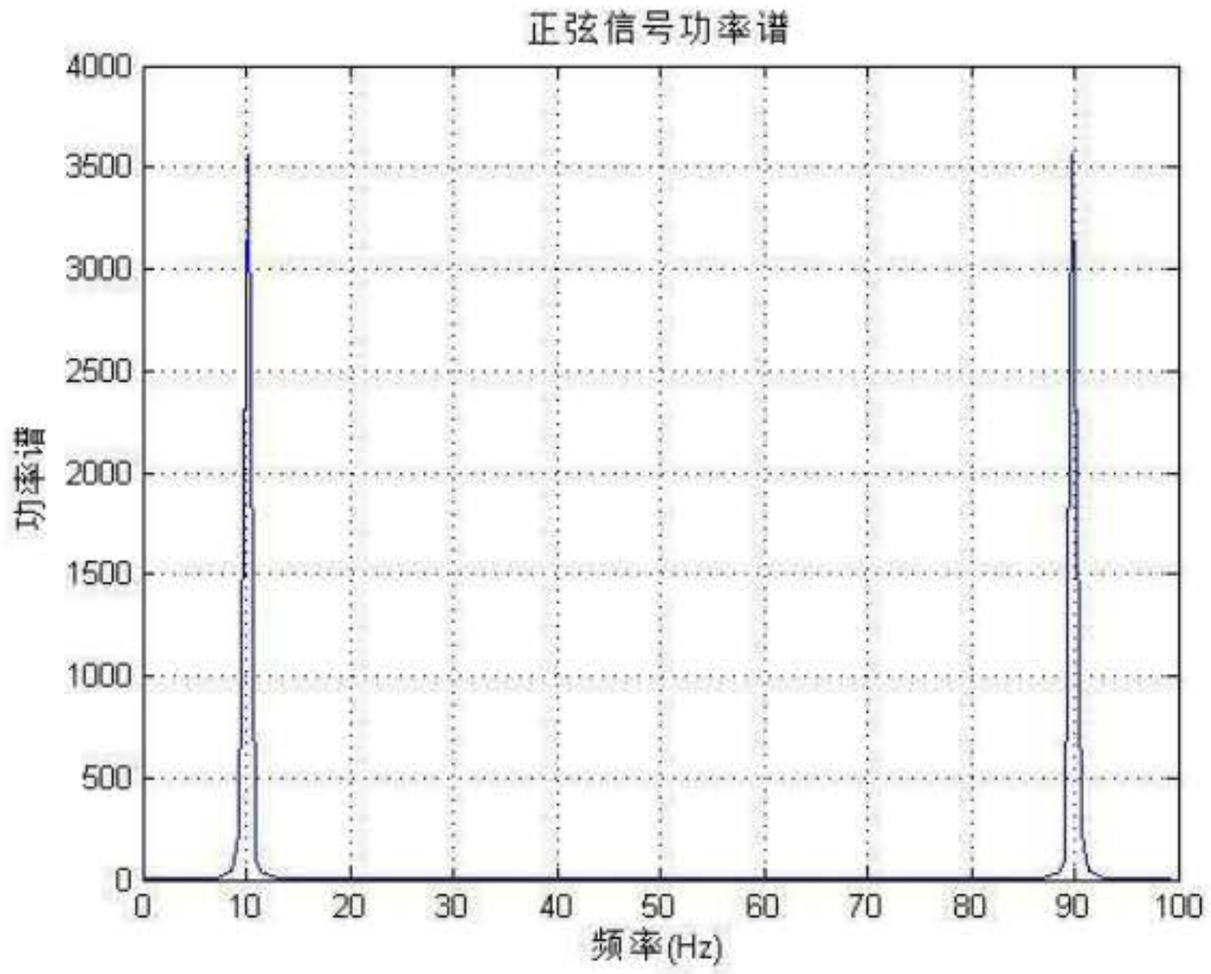


图 4 功率谱

```
%求对数谱
ln=log(sq);
figure(5);
plot(f,ln);
xlabel('频率(Hz)');
ylabel('对数谱');
title('正弦信号对数谱');
grid;
```

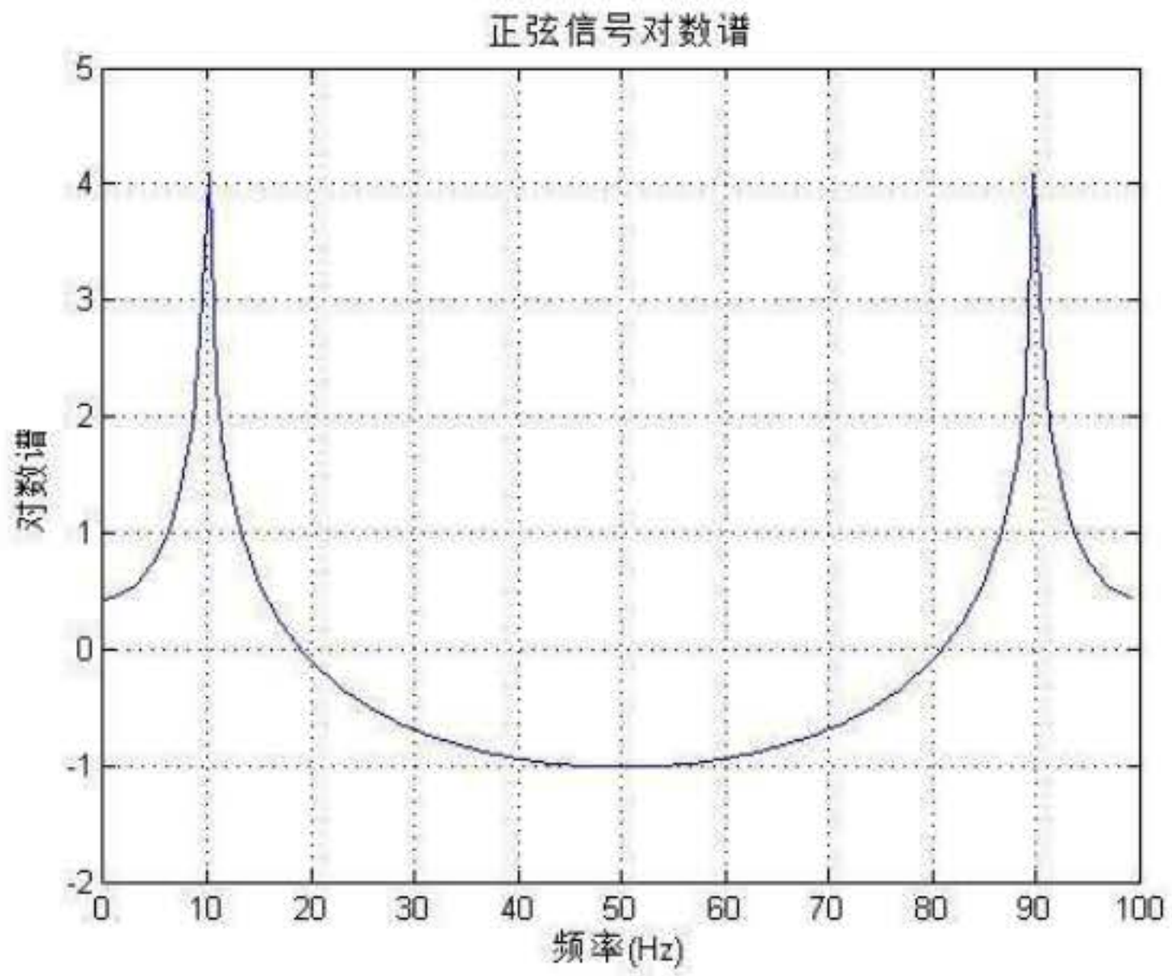


图 5 对数谱

```
%用 IFFT 恢复原始信号
xifft=ifft(y);
```

```
magx=real(xifft);
ti=[0:length(xifft)-1]/fs;
figure(6);
plot(ti,magx);
xlabel('t');
ylabel('y');
title('IFFT 转换的正弦信号');
grid;
```

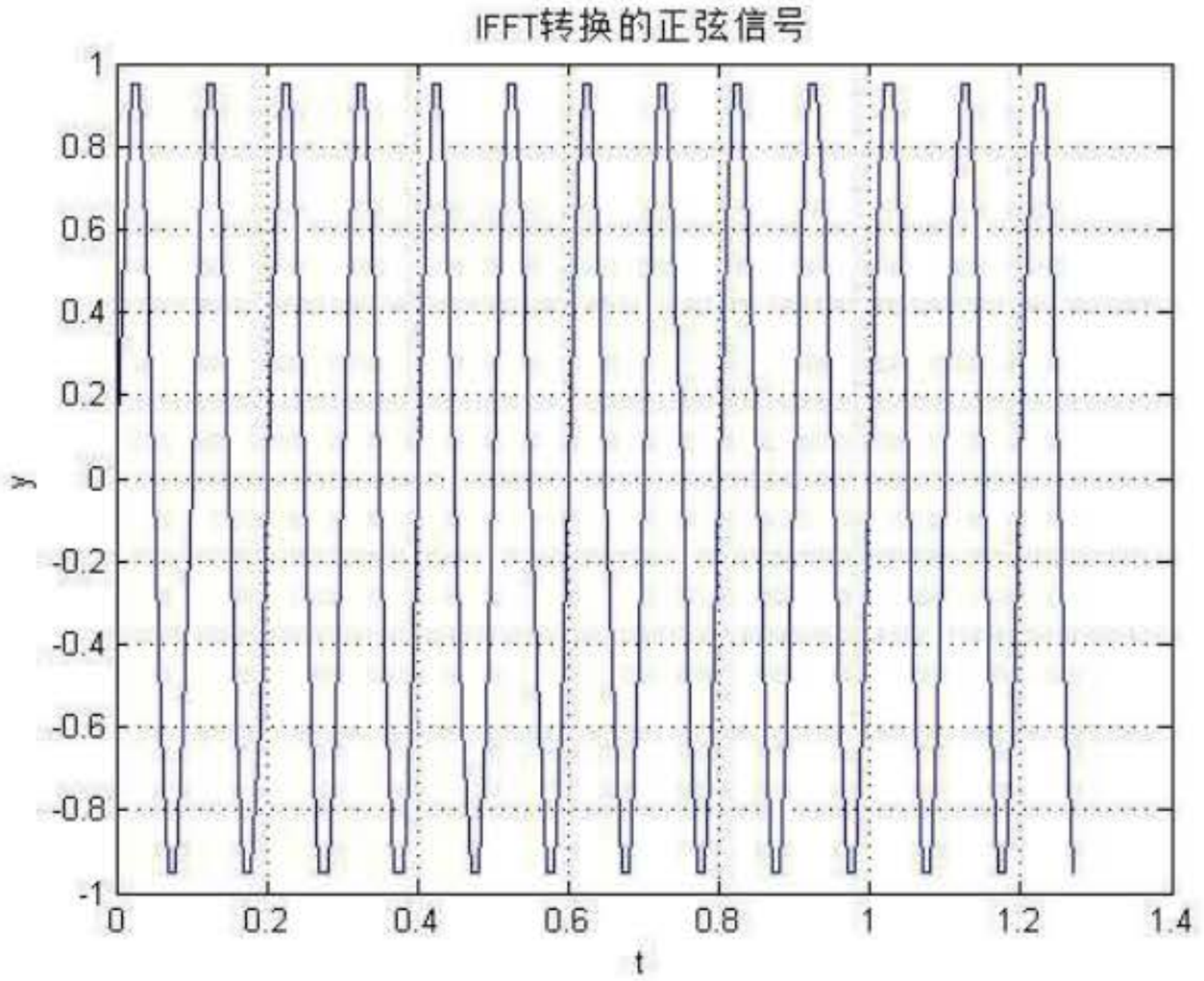


图 6 IFFT 转换

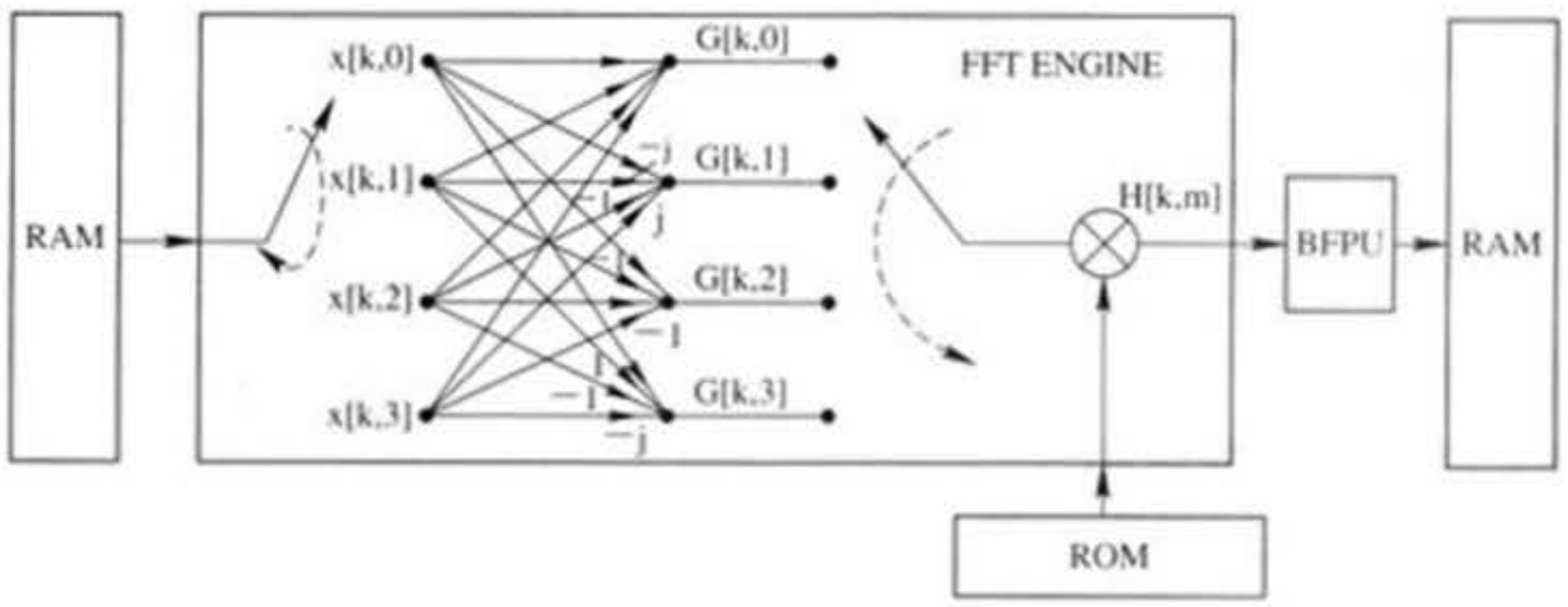
2、用 Verilog 语言在 FPGA 上实现 16 点 FFT 算法

1，总体设计方案：逐一对每个模块进行设计，采用 Verilog HDL 语言对各个模块进行设计，用 Quartus II 工具进行仿真。采用单输入主要功能模块包括，算术逻辑单元，蝶形运算单元，输入输出存储单元，系统顶层控制单元

2，算术逻辑单元：对 16 位有符号数求和，差，乘积，提供给蝶形运算器计算。

3，蝶形运算单元：完成 16 位有符号整数的蝶形运算

4，输入输出存储单元：存储输入序列和蝶形运算完成后的输出序列。



5，各模块综合代码如下：

```
module fft_16(xn_r,xn_i,RST,CLK,START,OUT,Xk_r,Xk_i);

input  [15:0] xn_r,xn_i;              //输入的实部与虚部
input  RST,CLK,START;
                                      //FFT 启动信号与时钟信号和复位信号
output [15:0] Xk_r,Xk_i;              //FFT 输出实部与虚部
output OUT;                           //输出标志信号

reg    [15:0] Xk_r,Xk_i;
reg    OUT;
reg    OUT1,STRT1;       //级联 FFT4 的输出标志和启动信号
reg    [2:0] k,j,m,n,l,p;  //循环指针
reg    [4:0] i;

reg    [15:0] IN_r[15:0],IN_i[15:0];          //存储输入的实部与虚部
reg    [15:0] OUT_r[15:0],OUT_i[15:0];        //存储输出的实部与虚部
reg    [15:0] TRANIN_r,TRANIN_i;              //输入序列与 FFT4 输入之间
的转接
reg    [15:0] TRANOUT_r,TRANOUT_i;            //输出序列与 FFT4 输出之间
的转接
reg    [2:0]  state;

parameter    Idle=3'b000,                     //空闲
             Input=3'b001,                    //输入
             Compute0=3'b010,                 //计算引擎
             Compute1=3'b100,                 //第二级计算的控制
             Butfly=3'b101,                   //蝶形计算
             Output=3'b110;                   //输出
```

```
//定义三个移位函数
function[15:0] Shift03;//乘以0.3827

input [15:0] xn;

begin
        Shift03={xn[15],xn[15:1]}- {xn[15], xn[15], xn[15],xn[15:3]}
                + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15],xn[15:7]}
                - {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15],
                xn[15], xn[15], xn[15], xn[15], xn[15],xn[15:13]};
end
endfunction

function[15:0] Shift07;//乘以0.7071

input [15:0] xn;

begin
      Shift07={xn[15],xn[15:1]}+ {xn[15], xn[15], xn[15],xn[15:3]}
                + {xn[15], xn[15], xn[15], xn[15], xn[15:4]}
                + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15:6]}
                + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
                xn[15], xn[15],xn[15:8]}+{xn[15], xn[15], xn[15], xn[15],
                xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
                xn[15], xn[15], xn[15], xn[15],xn[15:14]};
end
endfunction

function[15:0] Shift09;//乘以0.9239

input [15:0] xn;

begin
        Shift09=xn -{xn[15], xn[15], xn[15], xn[15],xn[15:4]}
                - {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
xn[15:6]}
                + {xn[15], xn[15], xn[15], xn[15], xn[15], xn[15], xn[15],
                xn[15], xn[15],xn[15:9]};
end
endfunction
```

```
//调用 FFT4 模块计算 4 组 4 点的 FFT
//FFT4
XFFT(.CLK(CLK),.START(STRT1),.xn_r(TRANIN_r),.xn_i(TRANIN_i),.OUT(OUT
1),.Xk_r(TRANOUT_r),.Xk_i(TRANOUT_i));

always @(posedge CLK or posedge RST)
       if(RST)
           begin
             state<=Idle;     //异步复位，高电平有效
             OUT<=0;          //输出无效
          end
    else
     begin
      case (state)

            //空闲状态
            Idle: begin
                    OUT<=0;
                    if (START) //只有在空闲状态下才能启动 FFT
                       begin
                         state<=Input;
                         i<=0;
                       end
                  else
                       begin
                         state<=Idle;
                       end
             end

//16 位复数输入
Input: begin
          if(i<16)
            begin
              IN_r[i]<=xn_r;
              IN_i[i]<=xn_i;
              i<=i+1;
            end
         else if(i==16) //输入完毕进入的计算
            begin
              state<=Compute0;
              STRT1<=1;//启动 FFT4
              j<=0;     //指针复位
              k<=0;
```

```
                m<=0;
                n<=0;
              end
          end

//第一二级计算
Compute0: begin
            if(j<4) //4 组串行输入计算
              begin
              STRT1<=0;//关断 FFT4 启动信号
              if(k<4)
                begin
                  TRANIN_r<=IN_r[4*k+j];  //时选法输入
                  TRANIN_i<=IN_i[4*k+j];
                  k<=k+1;
                end
              else if(k==4)//k=4 输入完毕，则等待 FFT4
的输出
                begin
                  if((OUT1==1)&&(m<4)) //可以输出
                    begin          //这两个信号同时消
失
                    OUT_r[4*j+m]<=TRANOUT_r;//连续存
储
                    OUT_i[4*j+m]<=TRANOUT_i;
                    m<=m+1;
                  end
                else
                      begin
                       if(m==4) //输出完毕，并非没有
开始
                        begin
                        j<=j+1;    //进入下一组的计
算
                        m<=0;  //清零，以备用
                        k<=0;
                        STRT1<=1;//为下一个计算作准
备启动信号
                        end
                      end//输出完毕
                  end
              end
            else if(j==4)
              begin
```

蝶形计算

```
if(n==0)
   begin
     state<=Butfly;//第一级计算完毕进入
     n<=n+1;
   end
else if(n==3)
      begin
        state<=Output;  //第二级计算
        l<=0;
        p<=0;
      end
   end
end

Butfly:  begin
          if(n==1)
            begin
              IN_r[0]<=OUT_r[0];
              IN_i[0]<=OUT_i[0];
              IN_r[1]<=OUT_r[1];
              IN_i[1]<=OUT_i[1];
              IN_r[2]<=OUT_r[2];
              IN_i[2]<=OUT_i[2];
              IN_r[3]<=OUT_r[3];
              IN_i[3]<=OUT_i[3];
              IN_r[4]<=OUT_r[4];
              IN_i[4]<=OUT_i[4];

IN_r[5]<=Shift09(OUT_r[5])+Shift03(OUT_i[5]);

IN_i[5]<=Shift09(OUT_i[5])-Shift03(OUT_r[5]);

IN_r[6]<=Shift07(OUT_r[6])+Shift07(OUT_i[6]);

IN_i[6]<=Shift07(OUT_i[6])-Shift07(OUT_r[6]);

IN_r[7]<=Shift03(OUT_r[7])+Shift09(OUT_i[7]);

IN_i[7]<=Shift03(OUT_i[7])-Shift09(OUT_r[7]);
              IN_r[8]<=OUT_r[8];
              IN_i[8]<=OUT_i[8];
```

```
IN_r[9]<=Shift07(OUT_r[9])+Shift07(OUT_i[9]);

IN_i[9]<=Shift07(OUT_i[9])-Shift07(OUT_r[9]);
                            IN_r[10]<=OUT_i[10];
                            IN_i[10]<=0-OUT_r[10];

IN_r[11]<=Shift07(OUT_i[11])-Shift07(OUT_r[11]);

IN_i[11]<=0-Shift07(OUT_i[11])-Shift07(OUT_r[11]);
                            IN_r[12]<=OUT_r[12];
                            IN_i[12]<=OUT_i[12];

IN_r[13]<=Shift03(OUT_r[13])+Shift09(OUT_i[13]);

IN_i[13]<=Shift03(OUT_i[13])-Shift09(OUT_r[13]);

IN_r[14]<=Shift07(OUT_i[14])-Shift07(OUT_r[14]);

IN_i[14]<=0-Shift07(OUT_i[14])-Shift07(OUT_r[14]);

IN_r[15]<=0-Shift09(OUT_r[15])-Shift03(OUT_i[15]);

IN_i[15]<=Shift03(OUT_r[15])-Shift09(OUT_i[15]);
                             n<=n+1;
                            end
                           else
                            begin
                              if(n==2)
                                 state<=Compute1; //进入第二级计算
                            end
                          end

                 Compute1: begin
                             n<=3;
                             j<=0;
                             STRT1<=1;
                             state<=Compute0;
                          end

                 Output: begin//倒序输出
                           if(l<4)
                             begin
                                  OUT<=1;
                                  Xk_r<=OUT_r[4*p+1];
```

```
                Xk_i<=OUT_i[4*p+1];
                p<=(p==3)?0:(p+1);
                l<=(p==3)?(l+1):l;
            end
        else if(l==4) //输出完毕
            begin
                OUT<=0;
                state<=Idle;
            end

                        end

                    default: begin
                                state<=Idle;
                            end

                endcase
            end
endmodule
```

6，数据流结构仿真波形：