第 5 章

CHAPTER 5

综 合 案 例

本章结合前面介绍的内容，将给出 7 个综合设计案例。本章案例的选择考虑的主要因素如下。

1. 案例实现的复杂度

案例主要面向教学，案例不能太简单或过于复杂，应既能锻炼基于 Verilog HDL 的数字系统设计能力，又能单独作为案例进行教学内容安排。

2. 案例内容的知识领域

案例内容的选择涵盖高等学校电子信息相关专业的主要知识内容，内容涵盖数值计算、信号生成、数字信号处理、数字通信等。

3. 案例的实现方法

案例注重对学生的综合能力的培养，除了熟练运用 Verilog HDL 知识实现数字系统设计以外，还锻炼学生善于结合现成可用的 IP 核以及第三方软件的能力，在实现比较复杂的系统功能的同时提高设计效率。通过对本章案例的学习，可以为实现更加复杂的工程案例奠定坚实的基础。

4. 国产芯片生态建设

本章的 FFT 幅频特性分析案例采用高云半导体的 FFT IP 核和复数乘法器 IP 核，旨在为国产芯片及 EDA 工具的生态建设贡献一份微薄之力。

本章的 7 个案例题目分别为：

- 数值计算；
- 信号生成；
- 数字混频；
- 数字滤波；
- FFT 幅频特性分析；
- BPSK 调制解调；
- DBPSK 调制解调。

5.1 数值计算

1. IEEE 754 单精度浮点乘法运算

32 位 IEEE 754 标准的浮点数的数据格式如图 5.1 所示。

图 5.1　IEEE 754 单精度浮点数据格式

IEEE 754 标准的单精度浮点数的数据格式为 3 部分。

(1) 符号(sign)：占用 1 位，1 代表负，0 代表正。

(2) 指数(exponent)：占用 8 位，存储数据的科学计数法中的指数部分，但增加了一个偏移量。

(3) 尾数(mantissa)：占用 23 位，存储数据的小数部分。

如图 5.2 所示，p、q 是两个 IEEE 754 标准浮点数，每个数据由三部分组成：符号 s、指数 e 和尾数 m。当实现 $s=p\times q$ 时，两个符号位进行 XOR 运算，表达式为 $s_s=e_p$ XOR e_q。指数位分别从偏移量 127 中减去，然后求和，最后根据 IEEE 754 标准将 127 加到和中，表达式为 $e_s=e_p-127+e_q-127+127$。尾数计算的表达式为 $rt=1.m_p\times 1.m_q$，rt 的二进制表示为 $rt_1rt_0.rt_{-1}rt_{-2}\cdots rt_{-46}$。尾数最终的结果需要根据 rt_1 的取值进行选择，若 rt_1 为 1，则尾数结果为 $rt_0rt_{-1}rt_{-2}\cdots rt_{-22}$；若 rt_1 为 0，则尾数结果为 $rt_{-1}rt_{-2}\cdots rt_{-23}$。指数根据 rt_1 是否为 1 进行调整。如果 rt_1 为 1，则指数需要加 1；如果 rt_1 为 0，则指数不变。

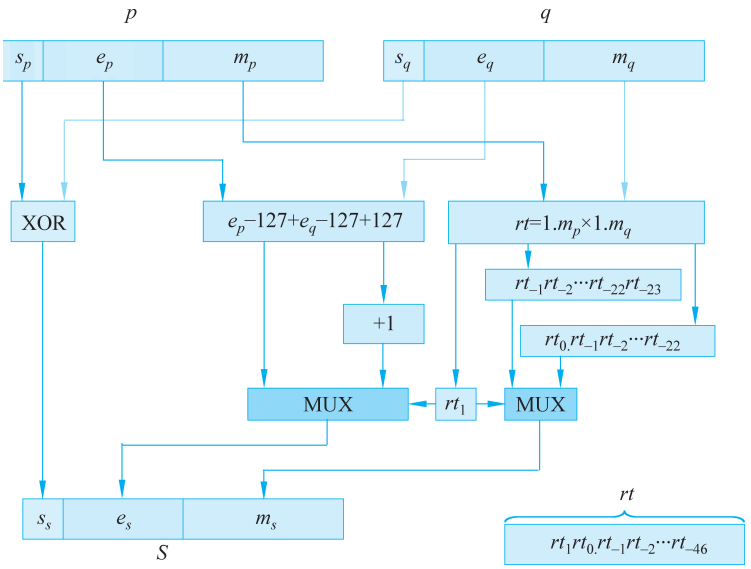


图 5.2　IEEE 754 单精度浮点乘法实现结构图

【例 5.1】　IEEE 754 单精度浮点乘法实现。

```
module mult_754 (input rst_n,clk,
                 input [31:0] din1,din2,
                 output reg[31:0] rt_o);

reg [7:0] exp_rt;
reg [7:0] exp_rt_o;
reg [47:0] meta_rt;
```

```
reg [22:0] meta_rt_o;
reg [31:0] din1_r,din2_r;
wire [31:0] rt;

//数据输入
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
      begin
        din1_r<='b0;
        din2_r<='b0;
      end
    else
      begin
        din1_r<=din1;
        din2_r<=din2;
      end
end

//指数计算
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
        exp_rt<=8'b00000000;
    else
    begin
         if ((din1_r[30:23]==8'b0)&&(din2_r[30:23]==8'b0))
              exp_rt<=8'b0;
         else
              exp_rt<=(din1_r[30:23]-'d127)+(din2_r[30:23]-'d127)+'d127;
     end
end

//尾数计算
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
        meta_rt<=48'b0;
    else
        meta_rt<={1'b1,din1_r[22:0]}*{1'b1,din2_r[22:0]};
    end

//位数处理
always@*
  begin
    case(meta_rt[47])
       1'b1: meta_rt_o<=meta_rt[46:24];
        default:meta_rt_o<=meta_rt[45:23];
     endcase
  end
```

```
//指数处理
  always@*
  begin
    case(meta_rt[47])
      1'b1: exp_rt_o<=exp_rt+1'b1;
       default:exp_rt_o<=exp_rt;
    endcase
  end

//按 IEEE 754 单精度浮点数格式生成结果
assign  rt[31]=din1_r[31]^din2_r[31];
assign  rt[30:23]=exp_rt_o;
assign  rt[22:0]=meta_rt_o;

//结果输出
always@(*)
  begin
     rt_o<=rt;
  end
endmodule
```

2. Mitchell 算法浮点乘法器

(1) Mitchell 算法原理。

乘法运算变为加法运算,最常见的方法是引入对数运算,如式(5.1)所示。

$$\begin{cases} p \times q = a^s \\ s = \log_a p + \log_a q \end{cases} \tag{5.1}$$

式(5.1)中的 p 和 q 是 a 进制下的非负数,虽然将乘法转为加法,但对数 $\log_a p$、$\log_a q$ 和指数 a^s 的计算同样不是一件容易的事情,所以要利用式(5.1)做乘法,关键在于要实现快速对数和指数运算。Mitchell 算法给出了对数和指数的近似计算方法。

假设数据 p 的二进制表示形式如式(5.2)所示。

$$z_n z_{n-1} \cdots z_1 z_0 . z_{-1} \cdots z_{-(m-1)} z_{-m} \tag{5.2}$$

其中 $z_n = 1$,且各个 $z_i \in \{0,1\}$,则 p 可以写成式(5.3)的形式。

$$p = 2^n + \sum_{i=-m}^{n-1} z_i 2^i = 2^n \left(1 + \sum_{i=-m}^{n-1} z_i 2^{i-n}\right) \tag{5.3}$$

令 $x = \sum_{i=-m}^{n-1} z_i 2^{i-n}$,并对式(5.3)求对数可得式(5.4)。

$$\log_2 p = \log_2 2^n (1+x) = n + \log_2 (1+x) \tag{5.4}$$

取 $\log_2(1+x) \approx x$,由式(5.4)可得到式(5.5)。

$$\log_2 p = \log_2 2^n (1+x) \approx n + x \tag{5.5}$$

其中 n 是整数部分,其值为 p 的二进制整数部分的位数减 1。x 是小数部分,其二进制表示如式(5.6)所示。

$$0 . z_{n-1} \cdots z_1 z_0 z_{-1} \cdots z_{-(1-m)} z_{-m} \tag{5.6}$$

比较式(5.2)和式(5.6)可以发现,x 可以由 p 通过小数点平移得到。式(5.5)中的 n 和 x 都可以由 p 的二进制表示得到,实现了对数的近似计算。

指数的近似计算是对数运算的逆运算，因此，可得式(5.7)描述的 Mitchell 算法中对数和指数的近似计算表达式。式(5.7)中的 n 为整数，$x\in[0,1)$。

$$\begin{cases}\log_2 2^n(1+x)\approx n+x\\ 2^{n+x}\approx 2^n(1+x)\end{cases}\tag{5.7}$$

(2) Mitchell 算法误差分析。

假设有两个数 $p=2^{n_1}(1+x_1)$和 $p=2^{n_2}(1+x_2)$，由 Mitchell 算法可得式(5.8)。

$$\log_2 p+\log_2 q=n_1+n_2+x_1+x_2\tag{5.8}$$

① 当 $x_1+x_2<1$ 时，近似的指数结果为 $2^{n_1+n_2}(1+x_1+x_2)$，近似程度如式(5.9)所示。

$$\frac{2^{n_1+n_2}(1+x_1+x_2)}{2^{n_1}(1+x_1)2^{n_2}(1+x_2)}=\frac{1+x_1+x_2}{1+x_1+x_2+x_1x_2}\tag{5.9}$$

② 当 $x_1+x_2\geqslant 1$ 时，近似的指数结果为 $2^{n_1+n_2+1}(x_1+x_2)$，近似程度如式(5.10)所示。

$$\frac{2^{n_1+n_2+1}(1+x_1+x_2)}{2^{n_1}(1+x_1)2^{n_2}(1+x_2)}=\frac{2(x_1+x_2)}{1+x_1+x_2+x_1x_2}\tag{5.10}$$

以上两种情况都是在 $x_1=x_2=0.5$ 时取到最小值，其结果为 8/9，即最大的误差为 1/9≈11.1%。

(3) 基于 Mitchell 算法的单精度浮点乘法运算实现。

如图 5.3 所示，p、q 是两个 IEEE 754 标准浮点数，每个数据由三部分组成：符号 s、指数 e 和尾数 m。当使用 Mitchell 算法实现 $s=p\times q$ 近似时，两个符号位进行 XOR 运算，表达式为 $s_s=e_p$ XOR e_q。指数位分别从偏移量 127 中减去，然后求和，最后根据 IEEE 754 标准将 127 加到和中，表达式为 $e_s=e_p-127+e_q-127+127$。尾数计算的表达式为 $C_s.m_s=0.m_p+0.m_q$，C_s 是进位。对指数根据 C_s 是否为 1 进行调整。如果 C_s 为 1，则指数需要加 1；如果 C_s 为 0，则指数不变。

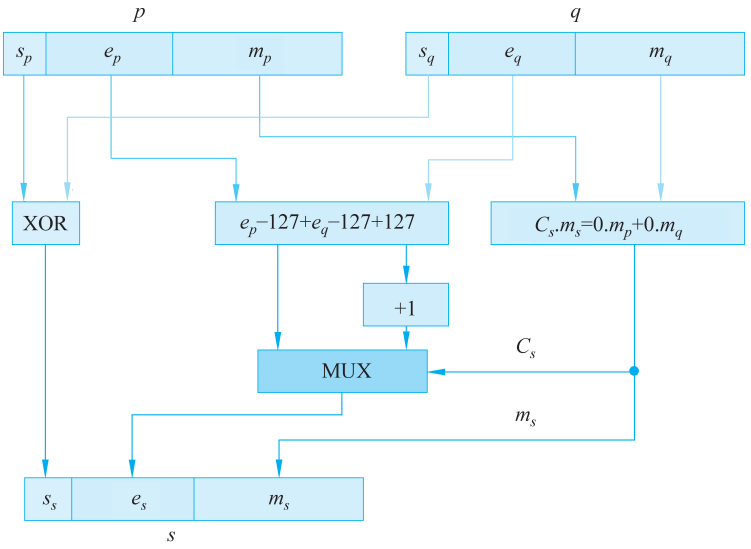


图 5.3　基于 Mitchell 算法的 IEEE 754 单精度浮点乘法实现结构图

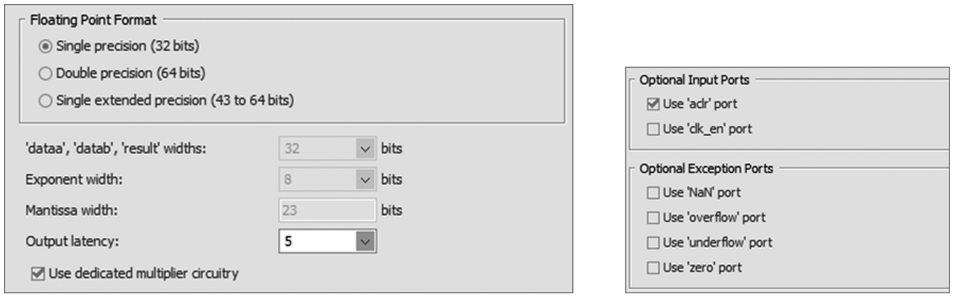
【例 5.2】　基于 Mitchell 算法的 IEEE 754 单精度浮点乘法实现。

```
module mitchell_754 (input rst_n,clk,
                     input [31:0] din1,din2,
                     output reg[31:0] rt_o);

reg [7:0] exp_rt;
reg [7:0] exp_rt_o;
reg [23:0] meta_rt;
reg [22:0] meta_rt_o;
reg [31:0] din1_r,din2_r;
wire [31:0] rt;

//数据输入
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
      begin
       din1_r<='b0;
       din2_r<='b0;
      end
else
    begin
       din1_r<=din1;
       din2_r<=din2;
    end
end

//指数计算
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
        exp_rt<=8'b00000000;
    else
      begin
         if ((din1_r[30:23]==8'b0)&&(din2_r[30:23]==8'b0))
              exp_rt<=8'b0;
         else
               exp_rt<=(din1_r[30:23]-'d127)+(din2_r[30:23]-'d127)+'d127;
      end
  end

//尾数计算
always@(posedge clk, negedge rst_n)
  begin
    if(!rst_n)
        meta_rt<=24'b0;
    else
        meta_rt<=din1_r[22:0]+din2_r[22:0];
  end

//尾数处理
```

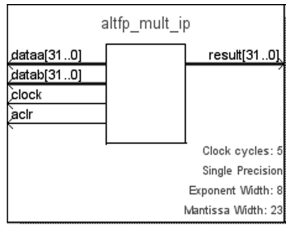
```
always@ *
   begin
     meta_rt_o<=meta_rt[22:0];
   end

//指数处理
always@ *
   begin
     case(meta_rt[23])
       1'b1: exp_rt_o<=exp_rt+1'b1;
       default:exp_rt_o<=exp_rt;
     endcase
  end

//按 IEEE 754 单精度浮点数格式生成结果
assign rt[31]=din1_r[31]^din2_r[31];
assign rt[30:23]=exp_rt_o;
assign rt[22:0]=meta_rt_o;

//结果输出
always@(*)
  begin
      rt_o<=rt;
   end
endmodule
```

3. 基于 IP 核的单精度浮点乘法运算实现

基于单精度浮点乘法器 IP 核的单精度浮点乘法运算实现(Altera IP 核)如图 5.4 所示。



(a) IP核参数设置



(b) 单精度浮点乘法器IP核端口

图 5.4　基于单精度浮点乘法器 IP 核的 IEEE 754 单精度浮点乘法运算

【例 5.3】 基于 IP 核的 IEEE 754 单精度浮点乘法实现。

```
module mult_ip_754 (input rst_n,clk,
                    input [31:0] din1,din2,
                    output reg [31:0] rt_o);

wire [31:0] rt;

altfp_mult_ip  altfp_mult_ip_inst (
    .aclr (!rst_n),
    .clock (clk),
    .dataa (din1),
    .datab (din2),
    .result (rt)
    );

always@ *
  begin
    rt_o<=rt;
  end
endmodule
```

仿真结果如下。

Testbench 代码如下：

```
`timescale 1ns/1ps
module tb_mult_float;

reg rst_n;                          //定义复位信号
reg clk;                            //定义时钟信号
reg [31:0] din1;                    //定义输入信号 1
reg [31:0] din2;                    //定义输入信号 2

wire [31:0]  rt_mult;               //例 5.1 浮点乘法器输出结果
wire [31:0]  rt_mitch;              //例 5.2 浮点乘法器输出结果
wire [31:0]  rt_mult_ip;            //例 5.3 浮点乘法器输出结果

//例化例 5.1 设计的浮点乘法器
mult_754 u1 (
    .clk(clk),
    .din1(din1),
    .din2(din2),
    .rst_n(rst_n),
    .rt_o(rt_mult));

//例化例 5.2 设计的 Mitchell 近似算法乘法器
mitchell_754 u2 (
    .clk(clk),
    .din1(din1),
```

```
    .din2(din2),
    .rst_n(rst_n),
    .rt_o(rt_mitch));

//例化例 5.3 设计的基于 IP 核的浮点乘法器
mult_ip_754 u3 (
    .clk(clk),
    .din1(din1),
    .din2(din2),
    .rst_n(rst_n),
    .rt_o(rt_mult_ip));

//生成复位信号、输入测试信号
initial
begin
    rst_n=1'b0;
    clk=1'b0;
  #100 rst_n=1'b1;
       din1=32'b01000001010000100101000111101011 1;      //66.32
       din2=32'b00111111100000000000000000000000;      //1.0
  #20  din2=32'b01000000000000000000000000000000;      //2.0
  #20  din2=32'b01000000010000000000000000000000;      //3.0
  #20  din1=32'b00111110101000111101011100001010;      //0.32
       din2=32'b00111110101000111101011100001010;      //0.32
  #20  din1=32'b01000001010001001100110011001100;      //12.3
       din2=32'b01000000100100011110101110000101;      //4.56

end

//生成周期为 20ns 的时钟信号
always
  begin
    #10 clk=~clk;
    end
endmodule
```

图 5.5 为 3 种浮点乘法器的仿真结果波形图，数值用十六进制表示。结果分析对比见表 5.1 和表 5.2。此外，由于本仿真用到了 Altera 的浮点乘法器 IP，因此在仿真时需要用到对应的 library。若用 ModelSim SE 版本，则需要进行 library 编译；若用 ModelSim AE 版本，则可直接添加需要的 library。本例需要添加的 library 为 220model_ver。

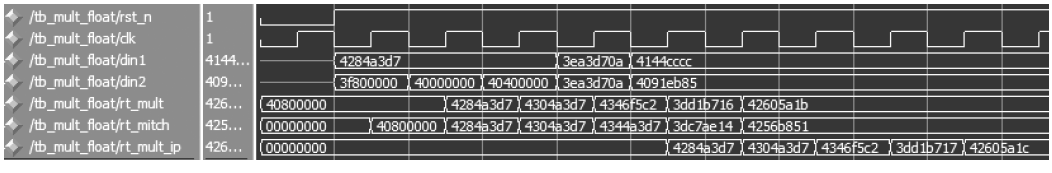


图 5.5　3 种浮点乘法器的仿真结果

表 5.1 基于 Mitchell 算法的单精度浮点乘法与原始单精度浮点乘法的结果对比

din1		din2		rt_mitch		rt_mult	
754 格式/H	十进制	754 格式	十进制	754 格式	十进制	754 格式	十进制
4284A3D7	66.32	3F800000	1	4284A3D7	66.319999694824	4284A3D7	66.319999694824
4284A3D7	66.32	40000000	2	4304A3D7	132.63999938965	4304A3D7	132.63999938965
4284A3D7	66.32	40400000	3	4344A3D7	196.63999938965	4346F5C2	198.95999145508
3EA3D70A	0.32	3EA3D70A	0.32	3DC7AE14	0.097499996423721	3DD1B716	0.10239998996258
4144CCCC	12.3	4091EB85	4.56	4256B851	53.679996490479	42605A1B	56.087993621826

表 5.2 基于 IP 的单精度浮点乘法与原始单精度浮点乘法的结果对比

din1		din2		rt_mult_ip		rt_mult	
754 格式/H	十进制	754 格式	十进制	754 格式	十进制	754 格式	十进制
4284A3D7	66.32	3F800000	1	4284A3D7	66.319999694824	4284A3D7	66.319999694824
4284A3D7	66.32	40000000	2	4304A3D7	132.63999938965	4304A3D7	132.63999938965
4284A3D7	66.32	40400000	3	4346F5C2	198.95999145508	4346F5C2	198.95999145508
3EA3D70A	0.32	3EA3D70A	0.32	3DD1B717	0.10239999741316	3DD1B716	0.10239998996258
4144CCCC	12.3	4091EB85	4.56	42605A1C	56.087997436523	42605A1B	56.087993621826

5.2 正弦波信号产生

1. 波形数据产生

借助 MATLAB 软件生成正弦波数据，MATLAB 代码如下：

```
clear;
close all
N=16;                          %一个周期取 16 个采样点
n=0:N-1;
x=0.5*sin(2*pi*n/N)+0.5;%无符号数生成表达式
%x=0.5*sin(2*pi*n/N);     %有符号数生成表达式
x=round(239*x);                %量化取整，需要根据硬件 DAC 的位宽进行量化，在此选取 8 位
%为了防止取整后溢出，所以选择系数小于 255
figure(1)
plot(n,x);
```

本例中的每个正弦波周期采样 16 个点，即将 2π 整个周期分为 16 等份，每个采样点对应的弧度值分别为 0，2π/20，(2π/20)×2，…，(2π/20)×15，对应的正弦值如表 5.3 所示，生成的正弦波如图 5.6 所示。