2013 年全国大学生电子设计竞赛

简易频率特性测试仪（E 题）

【本科组】

2013 年 9 月 7 日

摘　　要

摘要：本系统是基于 FPGA 和 STM32作为主系统，由控制两片高速 DAC900，产生两路相正交的扫频信号(1MHZ—40MHZ)，而 STM32作为整个系统的主控系统。通过 STM32控制扫频信号输出实现闭环稳幅，再通过乘法器 AD835实现扫频信号与被测网络出来信号的混（倍频），再通过 RC 低通滤波，用 STM32内部 AD 采集测出并显示幅频特性与相频特性，显示在彩屏中，是一款很不错的简易频率特性测试仪。很好的实现了题目的要求以及大部分的发挥部分。

关键字：扫频信号　FPGA　STM32　混频　乘法器

【Abstract】 The system is based on FPGA and STM32 as the main system, controlled by the two high-speed DAC900, generates two orthogonal sweep signal (1MHZ-40MHZ), while the STM32 master control system as a whole system. STM32 sweep signal output via closed-loop control leveled, and then through the multiplier AD835 achieve sweep signal and the measured network out mixed signals (frequency), and then through an RC low-pass filter, with the STM32 internal AD collection and displays the measured amplitude-frequency characteristic and phase-frequency characteristics, shown in color, the simplicity is a very good frequency characteristics tester. Achieved a very good and most of the requirements of the subject play a part.

Key Word: Frequency sweep signal　FPGA　STM32 mixing　multiplier

简易频率特性测试仪（E 题）

【本科组】

1 系统方案

本系统主要由正交扫频信号源模块、AD 模块、显示模块、电源模块组成，下面分别论证这几个模块的选择。

1.1 正交扫频信号源模块的论证与选择

方案一：基于 DDS AD9851 的扫频信号源，通过微处理器 STM32 控制 AD9851 频率控制字的设置。产生扫频的正弦波。AD9851 是一种高度集成的设备，采用先进的 DDS 技术，再加上内部高速度、高性能 D / A 转换器，和比较器，使一个数字可编程频率合成器和时钟发生器功能化。当参照准确的时钟源， AD9851 可以产生一个稳定的频率和相位且可数字化编程的模拟正弦波输出。此正弦波可直接用作时钟源，在其内部转化为方波成为灵活的时钟发生器。可通过两片 AD9851 调整相位产生两路正交的正弦波，实现扫频。整个系统结构简单，系统稳定。容易实现。系统结构如图

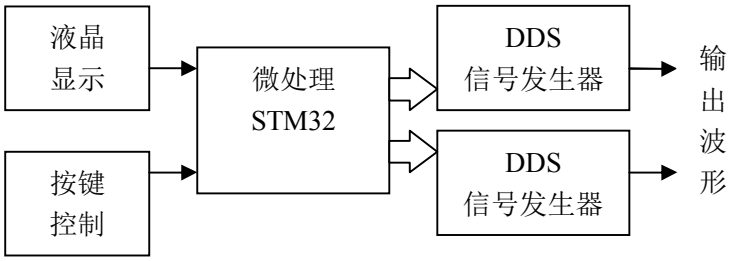

图 1-1 方案一系统结构图

方案二：采用AD9851制作扫频信号源发生器，可以产生正交信号，因为AD9851具有可以调节两路输出信号的相位功能，完全可以满足所产生的正交扫频信号误差的绝对值≤5º。但是，所输出的正交信号的很难保证信号电压的峰峰值≥1V，因此需要利用自动反馈网络去进行调整。将所产生的信号利用ADS831采集送给MCU STM32，由MCU STM32进行判断，当幅值没有达到要求时，MCU STM32则调动TPL0501数字电位器与VCA 810程控运放所组成的反馈网络将信号反馈到VCA 822（-10dB到10dB）放大模块中，适当调整其幅值，最终将调整后的信号进行输出即可满足输出信号电压的峰峰值≥1V，幅度平坦度≤5%的要求。但是鉴于该方案的反馈网络的设计过于困难，与及存在一定的未知干扰的原因，不选用该方案。

方案三：基于FPGA的扫频信号源，扫频信号源是扫频仪的重要组成部分，用于产生测试的正弦扫频信号。其输出的扫频信号应是等幅的，扫频范围应是可调的，扫频规律可以是线性扫频或对数扫频。以FPGA为平台，运用DDS技术实现扫频信号源。DDS输出的一般是数字化的正弦波，因此还需经过D / A转换器和低通滤波器才能得到一个可用的模拟信号。通过STM32位控制系统，实现液晶显示和按键控制。系统结构如图

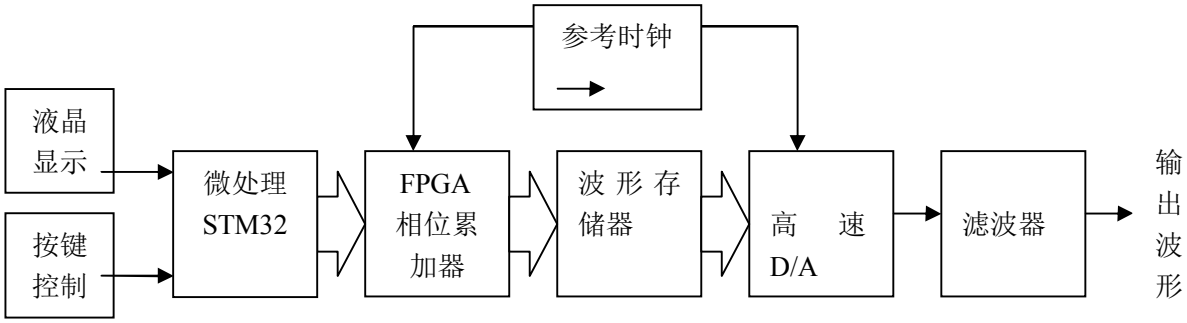

图 1-2 方案三系统结构图

综合以上三种方案，第三种方案系统稳定频率，实现起来容易，所以选择方案三。

1.2 被测网络的论证与选择

根据题目要求，被测网络为一个 RLC 串联谐振电路，且输入，输出阻抗均为 50 欧姆，被测网络铜带中心频率为 20MHZ，绝对绝对值误差≤5%，有载品质因数 Q=4，误差绝对值≤5%，有载电压增益≥ -1dB，频率步进为 100KHz。

方案一：RLC 串联谐振电路中 R 取值为0Ω，由于谐振条件为$\omega_0 \mathrm{L}=\frac{1}{\omega_0 \mathrm{C}}$；$\omega_0=2\pi \mathrm{f}_0$（其中$\mathrm{f}_0$已知，为中心频率$20\mathrm{M}Hz$）电路中的 LC 容易选择从而达到中心频率$20\mathrm{M}Hz$的时候起振，但是对于$\mathrm{Q}$值的理论计算应为$\mathrm{Q}=\frac{\omega_0 \mathrm{L}}{\mathrm{R}}=\frac{1}{\omega_0 \mathrm{CR}}=\frac{1}{\mathrm{R}}\sqrt{\frac{\mathrm{L}}{\mathrm{C}}}$由于 R 取值为 0；Q 值应为$\infty$；理论上没有办法达到题目的设计要求，因此该方案不合适。

方案二：RLC 串联谐振电路中$\mathrm{R}>0\Omega$，确保谐振条件$\omega_0 \mathrm{L}=\frac{1}{\omega_0 \mathrm{C}}$；$\omega_0=2\pi \mathrm{f}_0$的前提下，根据$\mathrm{Q}=\frac{\omega_0 \mathrm{L}}{\mathrm{R}}=\frac{1}{\omega_0 \mathrm{CR}}=\frac{1}{\mathrm{R}}\sqrt{\frac{\mathrm{L}}{\mathrm{C}}}$，得$\mathrm{R}=\frac{1}{\mathrm{Q}}\sqrt{\frac{\mathrm{L}}{\mathrm{C}}}$，因此可得 R 与 Q 成反比的关系，这时只需要选择合适的 R 值即可保证 Q 值=4，该电路为 RLC 串联谐振，容易设计，当$R=213.6\Omega; L=6.8\mu H; C=6.8pH$时可得到通带中心频率为 20MHz；有载品质因数为 4 的理想谐振电路；经过实验，设计的最终结果为$R=309.94\Omega(\text{取}330\Omega//5.1k\Omega); L=6.8\mu H; C=6.8pH$此时的通带中心频率为 19.68MHz；有载品质因数为 4.03 完全满足题目的设计要求。

论证：对比方案一和方案二，很明显方案二是最优选择。

综合以上三种方案，选择方案三。

1.3 混频电路的论证与选择

对于混频电路，前级信号同频信号，很明显就是需要完成倍频电路的设计。因此，优先考虑乘法器去完成是一个很不错的选择，但是也并不绝对。

方案一：采用晶体管倍频器实现信号的混频。晶体管倍频的电路结构与晶体管丙类谐振功率放

大器基本相同，当晶体管丙类工作时，输出集电极电流中基波分量的幅值最大，谐波次数越高，对应的振幅越小。因此必须要采取良好的输出带通滤波器，才能保证滤除倍频信号。电路原理图如下图 1-3 所示：

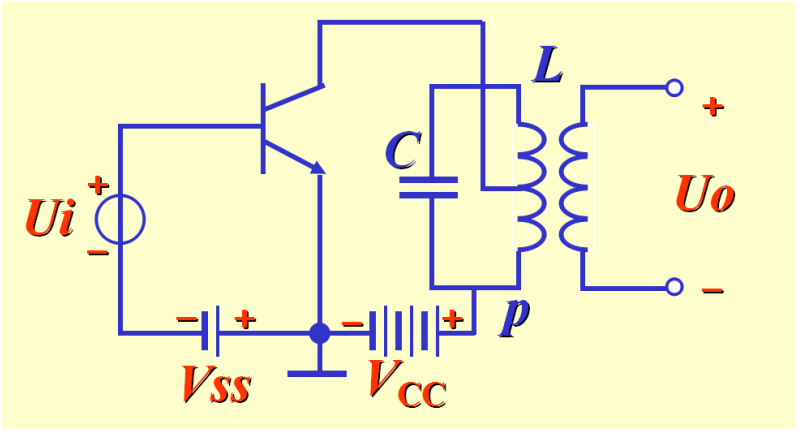

图1-3晶体管混频电路图

晶体管混频电路具有增益高，噪声低，设计简单的优点，但是混频失真大，本振严重泄漏，因此采用该电路不是很好。

方案二：如果采用集成的乘法器芯片，那么对于宽带精密模拟乘法器 MPY634 具有宽频带、高精度、四象限模拟乘法器，带宽 10MHz，在四象限范围内精度可达±0.5%，能够完成混频、倍频、调制、解调等各种电路的设计，但是由于题目的信号源频率范围为 1MHz～40MHz，而两路频率范围为 1MHz～40MHz 的倍频之后得到 2MHz～80MHz 的高频信号和直流分量。很明显，当混频后的频率大于 10MHz 时，MPY634 已经无法胜任了。因此使用乘法器 MPY634 去设计该混频电路是不可取的。

方案三：采用 ADI 公司的 AD835 乘法器，AD835 是个完备的 4 象限电压输出模拟放大器，有用输出为 250Mhz 的 3db 带宽（小信号上升时间为 1ns）。满量程（-1 V 至+1 V）上升至下降时间为 2.5 ns，0.1%建立时间通常为 20 ns,很明显，乘法器 MPY634 无法完成的工作， AD835 完全可以解决，AD835 具有的优势还不止这些，对于混频电路的设计也十分简单，只要尽量保证外围电路少，不对它干扰即可。

方案的对比：方案一混频失真大，本振严重泄漏，故不采取此方案；方案二无法满足混频后2MHz～80MHz 的输出，故不采取此方案；方案三比较可靠，各项指标都可以达到，综合考虑采用方案三。

1.2 低通滤波器的论证与选择

方案一：采用巴特沃斯二阶无源低通滤波器，该低通滤波器的传递函数为

$G_{(s)}=\dfrac{1}{R^2C^2S^2+3RCS+1}$;查表知，归一化后的 R=1；C=L=1.4142；由于理论的信号输出的是直流量，而无源滤波器是在频率较高的时候选用才具有一定的优势的。故不考虑此方案。

方案二：由于理论的信号输出的是直流量，因此选用简单的 RC 低通滤波器，即可达到滤波的作用，低通滤波器电路如下图 1-4 所示：

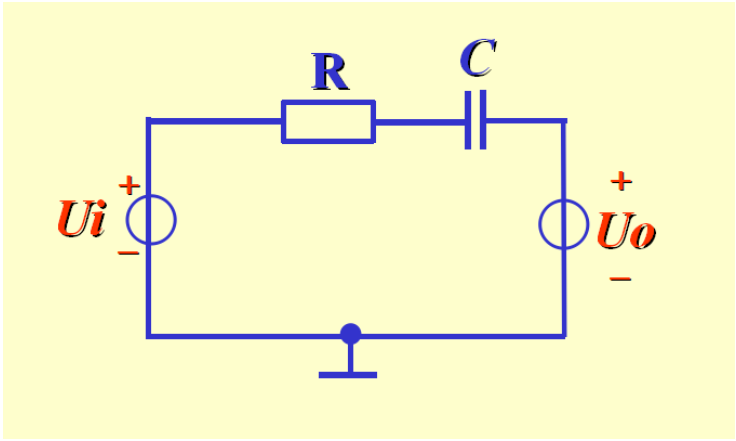

图1-4低通滤波器电路图

综合考虑采用方案二。

2 系统理论分析与计算

2.1 零中频正交解调原理的分析

2.1.1 零中频正交解调的原理

无论中频信号的调制方式如何,都可以先使用正交解调,然后再依据调制方式处理恢复信号。正交解调也叫正交基带变换,其目的是去掉调制信号中的载频,将信号变到零中频(基带)。一个载频为 ω_c 的实调制信号可以表示为:

$$x(t)=a(t)\cos[\omega_c t+\theta(t)]$$

则其复信号解析式为:

$z(t)=a(t)\cos[\omega_c t+\theta(t)]+ja(t)\sin[\omega_c t+\theta(t)]$ 其中 $a(t)$ 表示信号的瞬时包络,$\varphi(t)=\omega_c t+\theta(t)$ 表示信号的瞬时相位,而 $\omega(t)=d\varphi(t)\Pi dt=\omega c+\theta'(r)$ 表示信号的瞬时角频率。各种调制方式的信号调制信息都包含在这 3 个特征量中。经正交解调后得到的零中频信号(基带信号)为:

$$Z_B(t)=a(t)\cos\theta(t)+ja(t)\sin\theta(t)$$

$$=Z_{BI}(t)+jZ_{BQ}(t)$$

式中

$$Z_{BI}(t)=a(t)\cos\theta(t)$$

$$Z_{BQ}(t)=a(t)\sin\theta(t)$$

$Z_{BI}(t)$ 和 $Z_{BQ}(t)$ 分别为基带信号中的同相分量和正交分量,或称 I 路分量和 Q 路分量。

本系统使用自主设计的 DA900 模拟产生的正交扫频两路信号分别为 $A\cos\omega t$和$\mathrm{A}\sin\omega t$，因为经过被测网络之后的信号有一定的相移θ和信号的幅值有一定的衰减，这里设经过被测网络之后的信号为$\alpha\mathrm{A}\cos(\omega t+\theta)$，其中$\alpha<1$。

实现正交解调的方法如图下 2-1 所示。中频信号经正交解调后, 其信号调制信息都包含在 I、Q 两路分量信号中。依据信号调制方式对 I、Q 两路分量信号作相应的运算处理就可以完成具体调制信号的恢复。

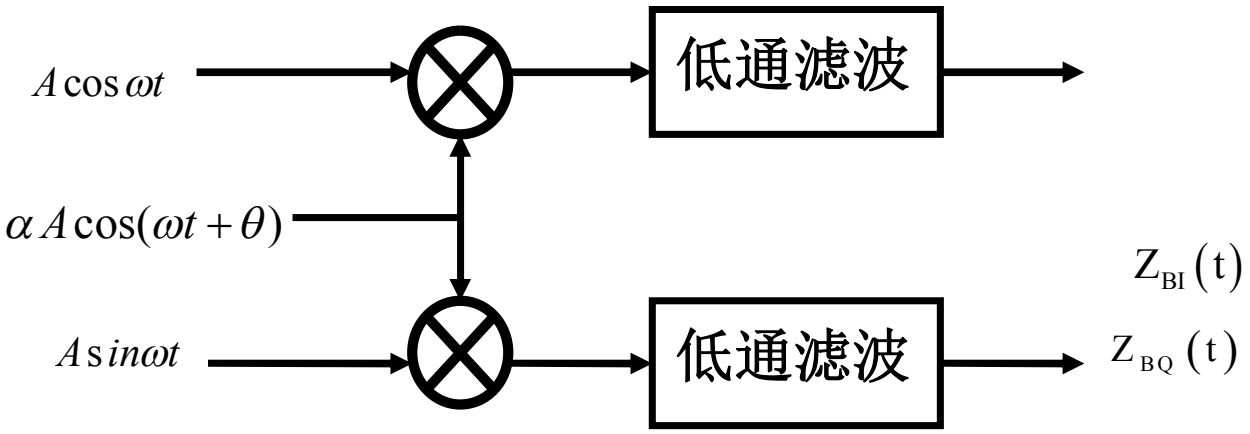

图 2-1 正交解调的方法

2.2 系统实现的推导计算

2.2.1 被测网络 RLC 串联谐振电路中 RLC 值的理论推导与计算

被测网络由 RLC 串联谐振电路组成，输入阻抗和输出阻抗$\mathrm{R_i}=\mathrm{R_o}=50\Omega$；要求被测网络通带中心频率为 20MHz，误差的绝对值≤5%；有载品质因数为 4，误差的绝对值≤5%；有载最大电压增益≥-1dB；如下图 2-2 所示：

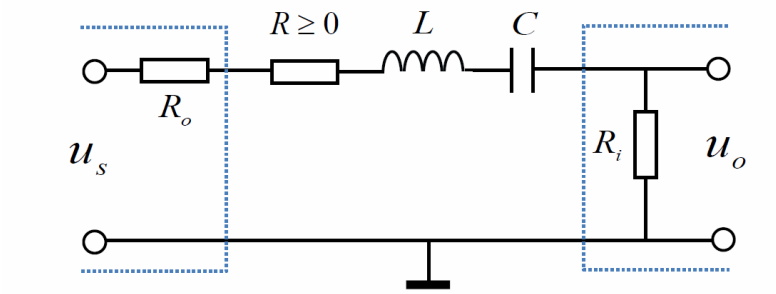

图 2-2RLC 串联谐振电路

在 RLC 串联谐振电路中，首先应该满足谐振条件，中心频率：

f_0=20MHz；

谐振频率：

$\omega_0=2\pi\mathrm{f}_0$；

当$\omega_0 \mathrm{L}=\frac{1}{\omega_0 \mathrm{C}}$时，电路将谐振。

电路中的 LC 容易选择从而达到中心频率$20\mathrm{M}Hz$的时候起振，但是对于Q值的理论计算应为：

$$\mathrm{Q}=\frac{\omega_0 \mathrm{L}}{\mathrm{R}}=\frac{1}{\omega_0 \mathrm{CR}}=\frac{1}{\mathrm{R}}\sqrt{\frac{\mathrm{L}}{\mathrm{C}}}$$

易知$\mathrm{R}=\frac{\omega_0 \mathrm{L}}{\mathrm{Q}}$；

有以上公式就可以推算出比较合理的 RLC 值满足该谐振电路的要求了。

2.2.2 整个信号流向的理论推导与计算

本系统的正交信号由 FPGA 控制 DAC900 模块进行模拟产生。所产生的正交信号 1 路和 2 路信号分别为$A\cos\omega t$和$\mathrm{A}\sin\omega t$，经过被测网络之后的信号第 3 路信号，为：$\alpha \mathrm{A}\cos(\omega t+\theta)$

1 路和 3 路信号经过乘法器之后：

$$A\cos\omega t\times\alpha \mathrm{A}\cos(\omega t+\theta)$$

$$=\frac{1}{2}\alpha A^2[\cos(2\omega t+\theta)+\cos\theta]$$

经过低通滤波器之后，会把高频分量滤掉，只剩下低频分量和直流量，由于该信号没有低频分量，因此，仅剩下直流分量：

$$=\frac{1}{2}\alpha A^2\cos\theta$$

经过 ADC 数模转换器之后，将会采集到个直流量的数据，最后将数据送向 MCUSTM32 单片机。

同理，对于 2 路和 3 路信号经过乘法器之后：

$$A\sin\omega t\times\alpha \mathrm{A}\cos(\omega t+\theta)$$

$$=\frac{1}{2}\alpha A^2[\sin(2\omega t+\theta)+\sin\theta]$$

经过低通滤波器之后，会把高频分量滤掉，只剩下低频分量和直流量，由于该信号没有低频分量，因此，仅剩下直流分量：

$$=\frac{1}{2}\alpha A^2\sin\theta$$

经过 ADC 数模转换器之后，将会采集到个直流量的数据，最后将数据送向 MCU STM32 单片机。

2.2.3 电压增益、相移和频率的还原推导与计算

接着上面的推算，MCU STM32 将采集到两路直接流量的数据，很明显是某个频率点对应的，其频率就是信号的扫频频率，在 FPGA 中运行时，就可以轻松将其直接读取出来，然后发给 MCU STM32 进行显示即可，频率还原成功。

此时，取一个频率点作为基础，下面推导并还原该频率点下的电压增益值$\frac{1}{2}\alpha A^2$和相移值θ，MCU STM32 中接收到的两路直流量$\frac{1}{2}\alpha A^2\cos\theta$和$\frac{1}{2}\alpha A^2\sin\theta$。

由于 MCU STM32 中接收到的两路直流量$\frac{1}{2}\alpha A^2\cos\theta$ 和$\frac{1}{2}\alpha A^2\sin\theta$ 都是已知的，这里，假设

$$\begin{cases} x=\frac{1}{2}\alpha A^2\cos\theta \rightarrow ① \\ y=\frac{1}{2}\alpha A^2\sin\theta \rightarrow ② \end{cases}$$，其中x,y均为已知值，由ADC采集得知。

由①÷②得$\frac{x}{y}=\cot\theta$ 其中,$\frac{x}{y}$，接下来制作$\cot\theta$ 与θ 值对应的数值表，然后查表可知道θ 的值，θ 值还原成功。

为了方便计算与还原，将电压增益值$\frac{1}{2}\alpha A^2$ 用z 代替， 则此时式①和式②可化简为：

$$\begin{cases} x=z\cos\theta \rightarrow ① \\ y=z\sin\theta \rightarrow ② \end{cases}$$，其中x,y均为已知值。

此时，由①^{2}和②^2 得

$$\begin{cases} x^2=z^2\cos^2\theta \rightarrow ③ \\ y^2=z^2\sin^2\theta \rightarrow ④ \end{cases}$$，其中x,y均为已知值。

由③+④ 得

$x^2+y^2=z^2$，其中x,y均为已知值。

接下来制作z^2与z 对应的数值表，然后查表可知道z 的值，而z 值就是电压增益值$\frac{1}{2}\alpha A^2$。因此，电压增益也可以成功还原。

3 电路与程序设计

3.1 电路的设计

3.1.1 系统总体框图

系统总体框图如下图 3-1 所示，

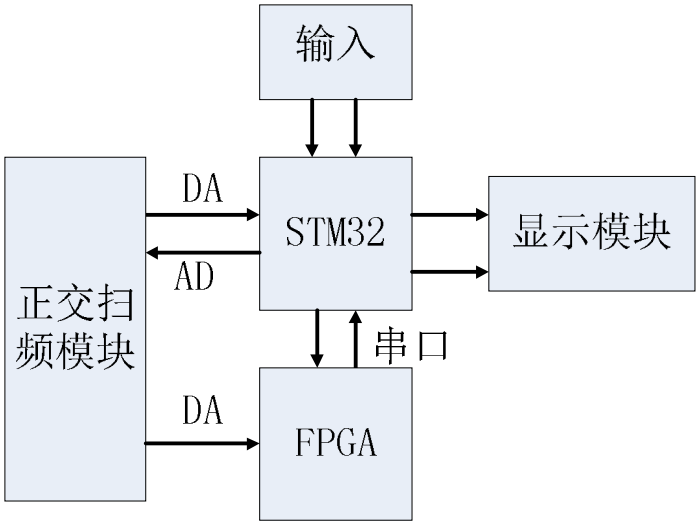

图 3-1　　系统总体框图

3.1.2 STM32 子系统框图与电路原理图

1、MCU STM32 子系统框图，如下图 3-2 所示，

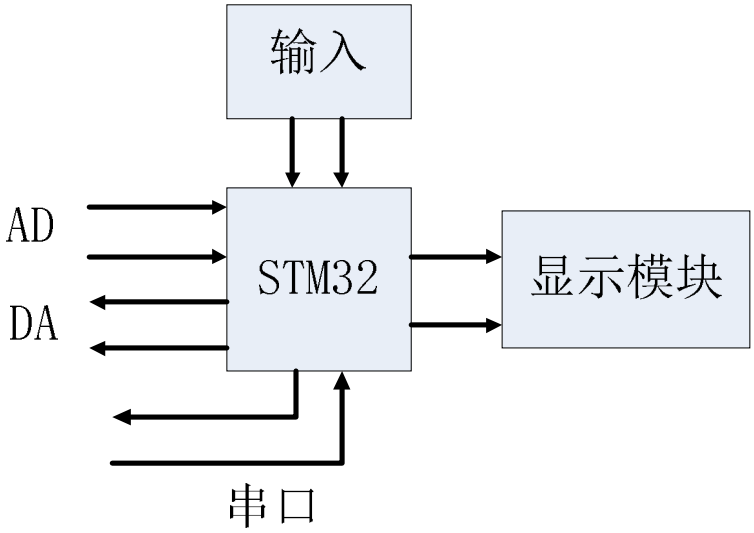

图 3-2　　STM 32 子系统框图

2、FPGA 子系统电路，如下图 3-3 所示，

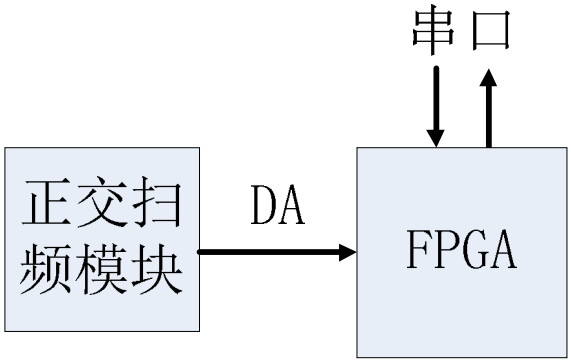

图 3-3　　FPGA 子系统电路

3.1.3 正交信号源子系统模块电路原理图

1、正交信号源子系统电路，如下图 3-4 所示，

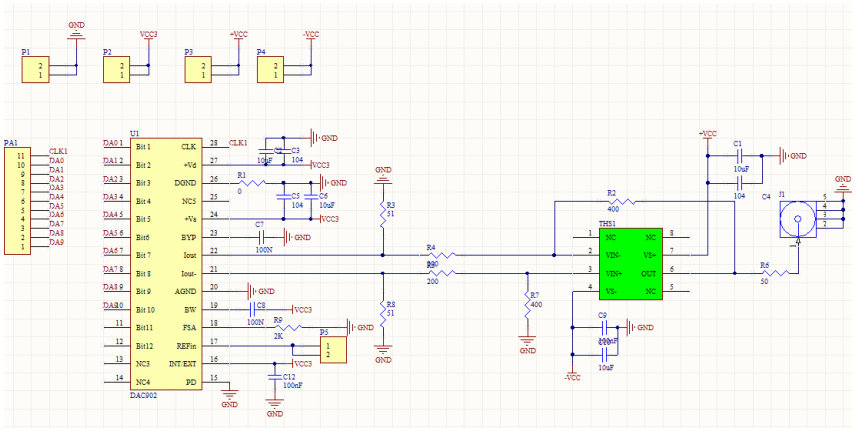

图 3-4 正交信号源子系统电路图

3.1.4 RLC 电路原理图

1、RLC 被测网络电路图如下图 3-4 所示，

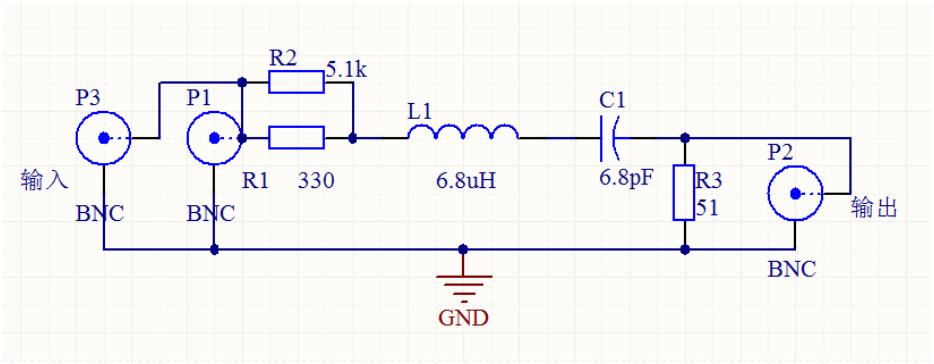

图 3-5　　RLC 被测网络电路图

3.1.5 乘法器与低通滤波器子系统框图与电路原理图

1、乘法器与低通滤波器子系统电路图如下图 3-6 所示，

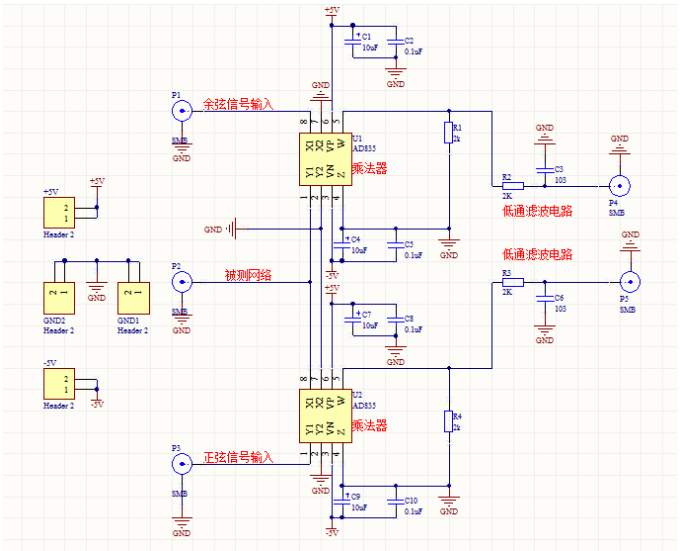

图 3-6 乘法器与低通滤波器子系统电路图

3.1.6 电源

电源由变压部分、滤波部分、稳压部分组成。为整个系统提供±5V 或者±12V 电压，确保电路的正常稳定工作。这部分电路比较简单，都采用三端稳压管实现，故不作详述。

3.2 程序的设计

3.2.1 程序功能描述与设计思路

1、程序功能描述

根据题目要求软件部分主要实现键盘的设置和显示。

1）键盘实现功能：设置频率值、频段、电压值以及设置输出信号类型。

2）显示部分：显示电压值、频段、步进值、信号类型、频率。

2、程序设计思路

3.2.2 程序流程图

1、主程序流程图，如下图 3-7 所示，

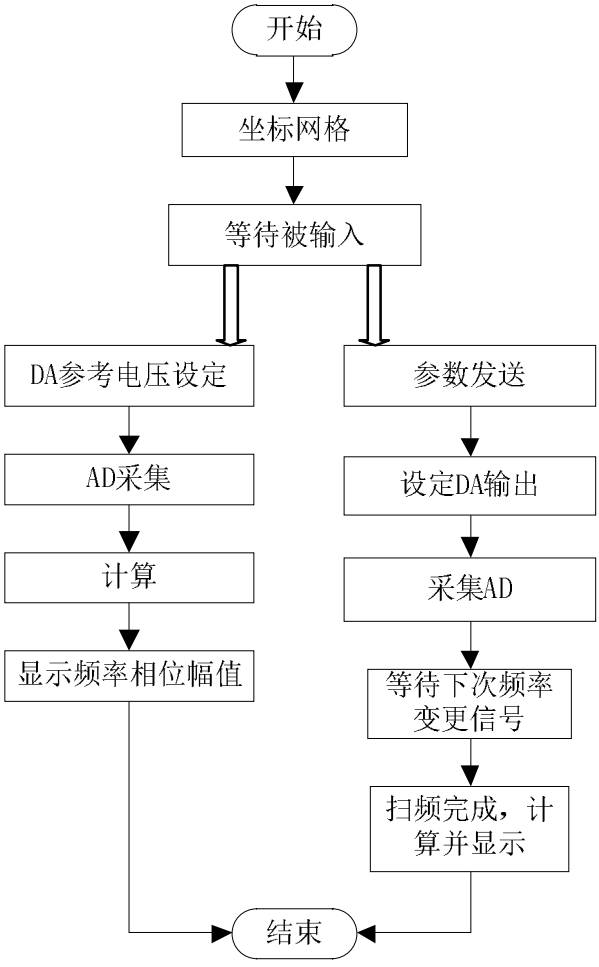

图 3-7 主程序流程图

2、MCU STM32 子程序流程图，如下图 3-8 所示，

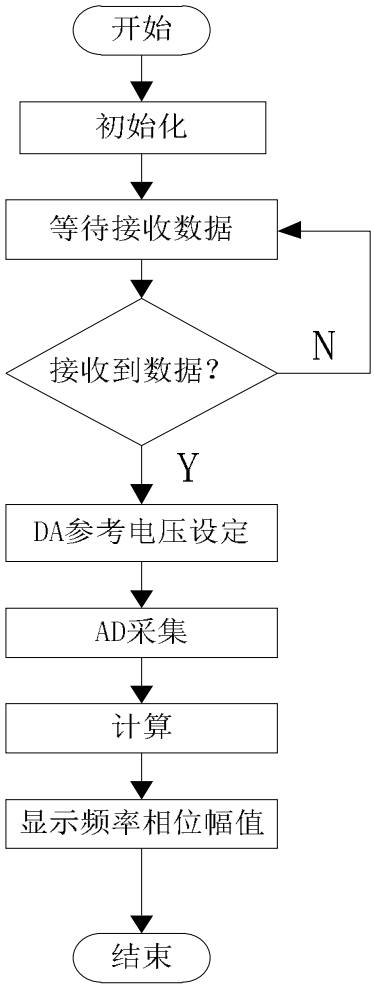

图 3-8 MCU STM32 子程序流程图

3、FPGA 子程序流程图，如下图 3-9 所示，

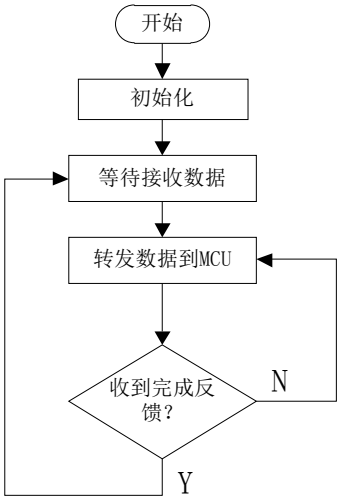

图 3-9 FPGA 子程序流程图

4 测试方案与测试结果

4.1 测试方案

1、硬件测试

4.1.1 被测网络的硬件测试

被测网络 RLC 谐振电路的中心频率和有载品质因数如下图 4-1 所示，

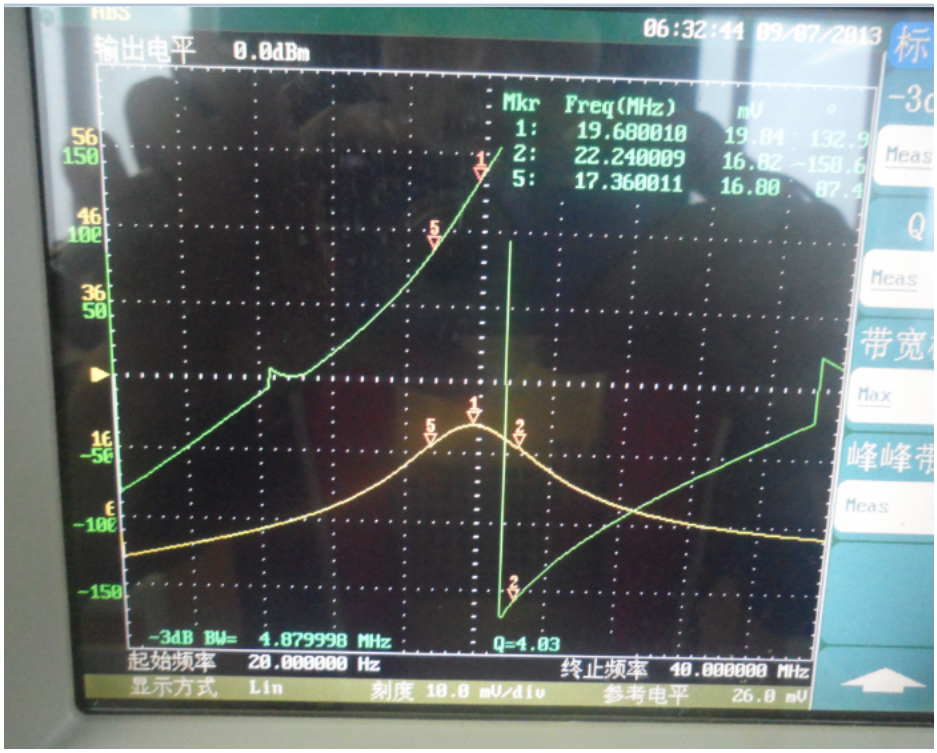

图 4-1RLC 谐振电路硬件测试

由上图可以看到实测被测网络通带中心频率为 19.68MHz。

误差分析：

$$\text{中心频率误差的绝对值}=\left|\frac{\text{理论值}-\text{实测值}}{\text{理论值}}\right|\times 100\%$$

$$=\left|\frac{20-19.68}{20}\right|\times 100\%$$

$$=1.6\%$$

由1.6%<5%，故而满足题目要求。

$$Q\text{值误差的绝对值}=\left|\frac{\text{理论值}-\text{实测值}}{\text{理论值}}\right|\times 100\%$$
$$=\left|\frac{4-4.03}{4}\right|\times 100\%$$
$$=0.75\%$$

由0.75%<5%，故而满足题目要求。

4.1.2 正交扫频信号源的硬件测试

正交扫频信号源扫频输出实测如下图 4-2 所示，

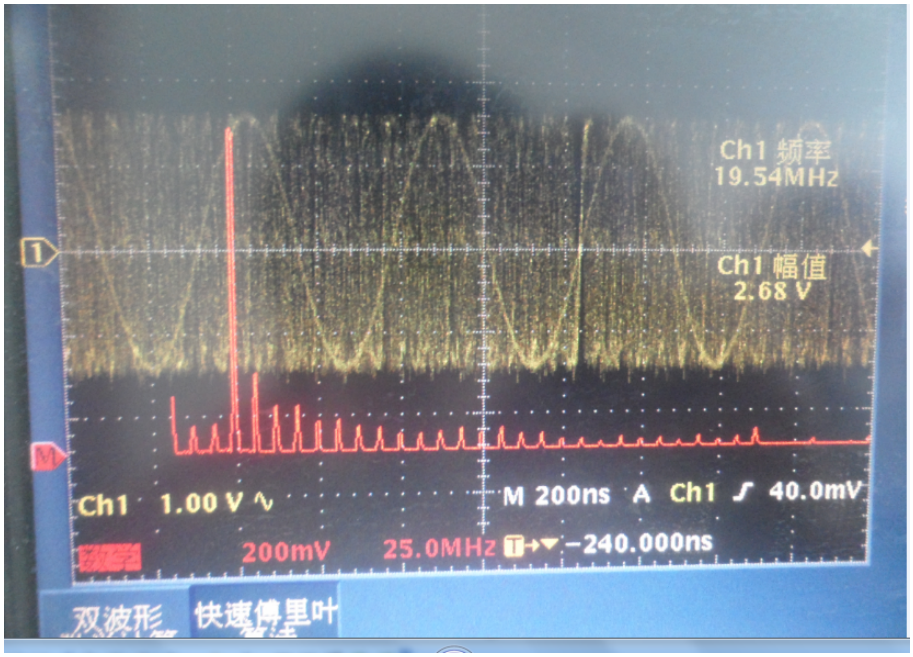

图 4-2 正交扫频信号源扫频输出图

2、软件仿真测试

被测网络 RLC 谐振电路软件仿真图下图 4-3 和图 4-4 所示，

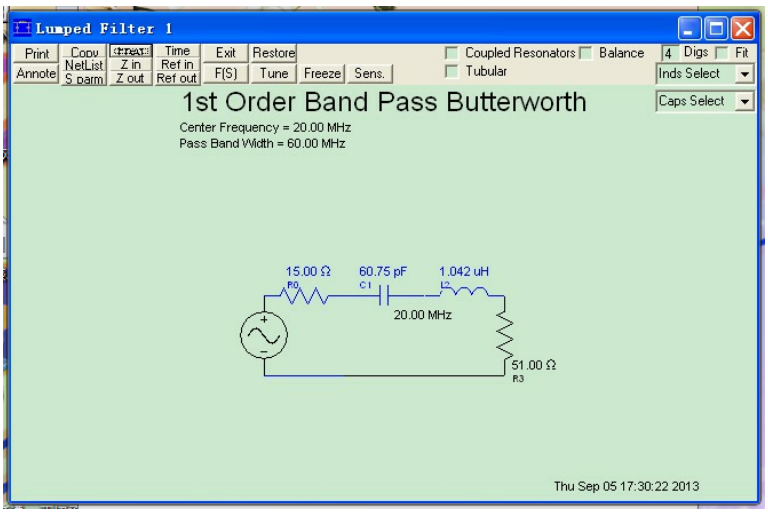

图 4-3 被测网络 RLC 谐振仿真电路图

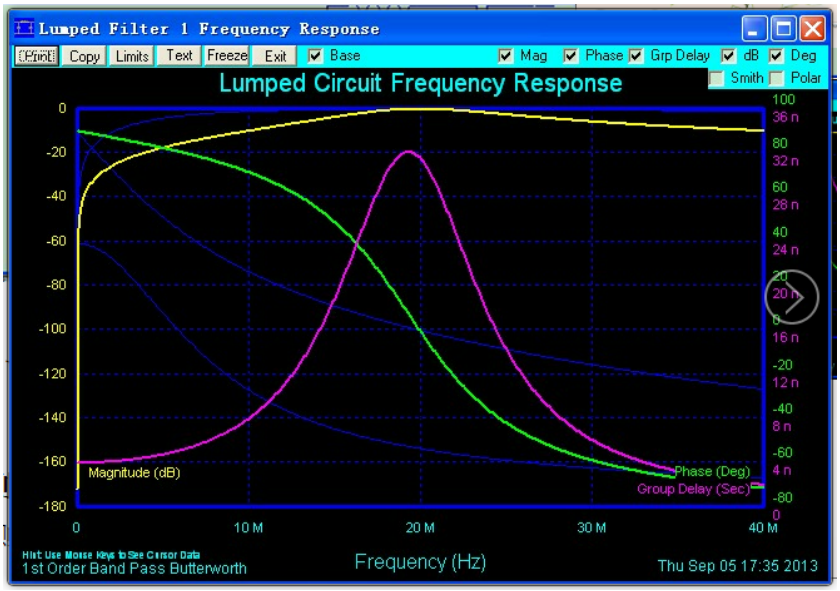

图 4-4 被测网络 RLC 谐振电路软件仿真图

4.2 测试条件与仪器

测试条件：检查多次，仿真电路和硬件电路必须与系统原理图完全相同，并且检查无误，硬件电路保证无虚焊。

测试仪器：高精度的数字毫伏表，模拟示波器，数字示波器，数字万用表，指针式万用表。

4.3 测试结果及分析

4.3.1 测试结果(数据)

整个系统参数的测试结果如下表所示：

预置频率（MHZ）	实测频率（MHZ）	相对误差（%）	波形失真度	相位差	相位误差（%）	U2	U1	幅度平衡误差（%）
1	1.00	0.00	无	90	0.00	1.78	1.78	0.00
1.5	1.50	0.00	无	90	0.00	1.72	1.72	0.00
2	2.00	0.00	无	90	0.00	1.51	1.51	0.00
2.5	2.54	1.60	无	90	0.00	1.46	1.44	1.39
3	2.97	1.00	无	89.9	0.11	1.46	1.45	0.69
3.5	3.48	0.57	无	89.8	0.22	1.44	1.42	1.41
4	3.89	2.75	无	89.7	0.33	1.42	1.42	0.00
4.5	4.44	1.33	无	89.7	0.33	1.40	1.44	2.78
5	4.93	1.40	无	89.8	0.22	1.39	1.42	2.11
10	9.88	1.20	无	89.5	0.56	1.30	1.33	2.26
15	14.29	4.73	无明显失真	84.9	5.67	1.32	1.33	0.75
20	19.20	4.00	无明显失真	85.6	4.89	1.28	1.31	2.29
25	24.75	1.00	无	88.5	1.67	1.29	1.25	3.20

30	29.66	1.13	无明显失真	85.2	5.33	1.20	1.23	2.44
35	34.21	2.26	无	87.6	2.67	1.20	1.22	1.64
40	39.49	1.28	无明显失真	77.2	14.22	1.18	1.21	2.48
45	44.41	1.31	无明显失真	76.6	14.89	1.17	1.20	2.50
50	49.46	1.08	无明显失真	75.5	16.11	1.10	1.15	4.35

4.3.2 测试分析与结论

根据上述测试数据，数值的相对误差都比较小，基本完成了频率特性测试仪的设计，由此可以得出以下结论：

1、两路信号的正交程度越好，相位差越接近 90°，也越容易恢复出来。

2、当频率比较小的时候，信号比较稳定，相对误差也比较小。频率越高时信号不易恢复，因为信号幅值有一定的衰减。

3、频率在 1 到 50MHz 频带下，信号均无明显失真，质量比较好。

综上所述，本设计达到设计要求。

附录 1：部分核心程序

```
void ADC3_CH12_DMA_Config(void)
{
  ADC_InitTypeDef       ADC_InitStructure;
  ADC_CommonInitTypeDef ADC_CommonInitStructure;
  DMA_InitTypeDef       DMA_InitStructure;
  GPIO_InitTypeDef      GPIO_InitStructure;

  /* Enable ADC3, DMA2 and GPIO clocks ****************************************/
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2 | RCC_AHB1Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);

  /* DMA2 Stream0 channel0 configuration **************************************/
  DMA_InitStructure.DMA_Channel = DMA_Channel_2;
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC3_DR_ADDRESS;
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADC3ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
//   DMA_InitStructure.DMA_BufferSize = 100;
  //DMA_InitStructure.DMA_BufferSize = 1;
     DMA_InitStructure.DMA_BufferSize = 2;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
//   DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  DMA_Init(DMA2_Stream0, &DMA_InitStructure);
  DMA_Cmd(DMA2_Stream0, ENABLE);


  /* Configure ADC3 Channel12 pin as analog input ******************************/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2|GPIO_Pin_3;
  //GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
```

```
        /*                          ADC                          Common                          Init
**********************************************************/
        ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;          //????
        ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;              //???
        ADC_CommonInitStructure.ADC_DMAAccessMode     =     ADC_DMAAccessMode_1;
//
        //ADC_CommonInitStructure.ADC_DMAAccessMode                                    =
ADC_DMAAccessMode_Disabled;
        ADC_CommonInitStructure.ADC_TwoSamplingDelay                                 =
ADC_TwoSamplingDelay_5Cycles;    //????
        ADC_CommonInit(&ADC_CommonInitStructure);

        /*                                    ADC3                                    Init
**************************************************************/
        ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;       //??12????
        //ADC_InitStructure.ADC_ScanConvMode = DISABLE;
        ADC_InitStructure.ADC_ScanConvMode = ENABLE;      //????,???????????
        ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;          //????
        ADC_InitStructure.ADC_ExternalTrigConvEdge   =   ADC_ExternalTrigConvEdge_None;
//??????
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;              //?????
        //ADC_InitStructure.ADC_NbrOfConversion = 1;
        ADC_InitStructure.ADC_NbrOfConversion = 2;                 //??ADC?????
        ADC_Init(ADC3, &ADC_InitStructure);

        /* ADC3 regular channel12 configuration *************************************/
        ADC_RegularChannelConfig(ADC3, ADC_Channel_12, 1, ADC_SampleTime_3Cycles);
        ADC_RegularChannelConfig(ADC3, ADC_Channel_13, 2, ADC_SampleTime_3Cycles);
        //ADC_RegularChannelConfig(ADC3,                ADC_Channel_12,                1,
ADC_SampleTime_3Cycles);

       /* Enable DMA request after last transfer (Single-ADC mode) */
        ADC_DMARequestAfterLastTransferCmd(ADC3, ENABLE);

        /* Enable ADC3 DMA */
        ADC_DMACmd(ADC3, ENABLE);

        /* Enable ADC3 */
        ADC_Cmd(ADC3, ENABLE);

          ADC_SoftwareStartConv(ADC3);             //启动转换
    }
```