# 第4章　电阻测量模块

杭州艾研信息技术有限公司

2014 年 11 月

# 申明

杭州艾研信息技术有限公司保留随时对其产品进行修正、改进和完善的权利，同时也保留在不作任何通告的情况下，终止其任何一款产品的供应的权利。用户在下订单前应及时获取相关信息的最新版本，并验证这些信息是当前的和完整的。

可通过如下方式获取最新信息、技术资料和技术支持：

技术支持电话：0571-86134572

技术支持邮箱：support@hpati.com

产品&资料下载中心：http://www.hpati.com/products/

互动论坛：http://www.hpati.com/bbs/forum.php

公司地址：浙江省杭州市西湖区留和路16号新峰商务楼B306

# 第4章　电阻测量模块

精确电阻测量并非易事，电阻测量模块提供了两种高精度测电阻的应用范例：电桥法测电阻和恒流源法测电阻。电桥法是将待测电阻代替原来已经平衡的电桥的一个桥臂，采用基准源为电桥供电，通过测量因电桥桥臂失衡产生的电压来获得待测电阻的值；恒流源法是设计一个在一定负载范围内能提供精确恒定电流的恒流源，待测电阻作为恒流源负载，测量待测电阻两端的电压即可获得电阻阻值。 为了提供性能比较，模块同时搭配两种不同的放大采集电路：一路采用仪用运算放大器 **INA333** 提供高共模抑制比的信号放大功能，再由 TIVA Cortex M4 自带的 12 位 ADC 进行数据采集；另一路直接采用 Δ∑型 ADC **ADS1100** 进行采集，并通过 $I^2C$ 将数据传给 TIVA Cortex M4。通过在模块上的跳线配置可以实现两种不同的前端测试电路和两种放大采集电路的交叉搭配，测试并比较不同配置方式的性能指标，可获得对不同测量方式和不同电路特性的认识。

# 电阻测量模块介绍

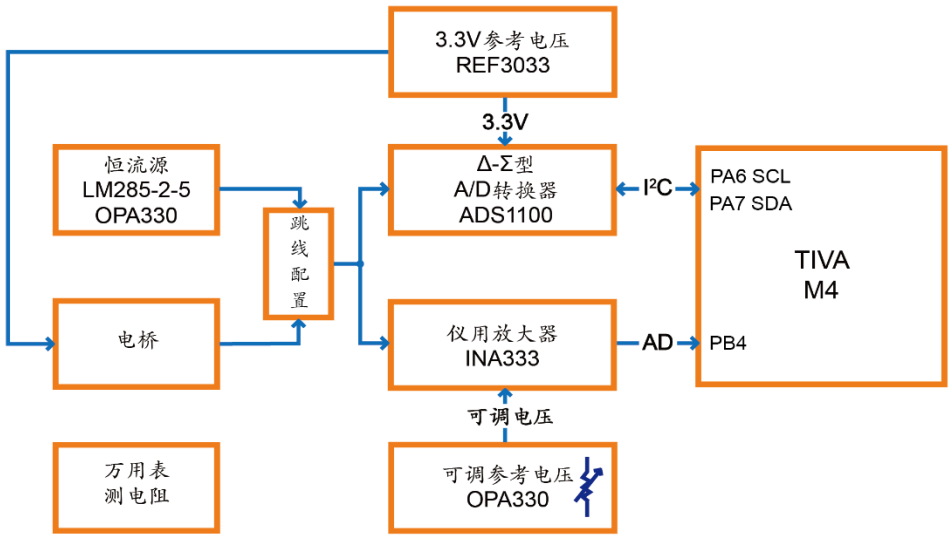
电阻测量模块提供了两种测电阻的方式和两种放大采集电路，通过跳线配置，可以组合成四套不同的测量电路。

两种测电阻的方式为：电桥法测电阻和恒流源法测电阻。电桥法是将待测电阻代替原来已经平衡的电桥的一个桥臂，采用基准源REF3033为电桥供电，通过测量电桥桥臂失衡产生的电压来计算获得待测电阻的值；恒流源法采用恒压芯片LM285-2-5和运算放大器OPA330组成一个在一定负载能力的恒流源，待测电阻作为恒流源负载，测量待测电阻两端的电压即可获得电阻阻值。

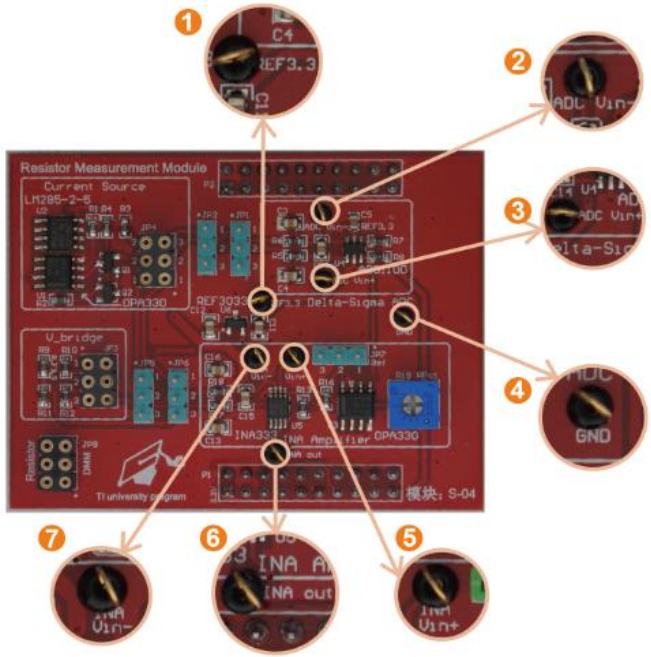
待测信号微弱并包含比较大比重的共模成份，需要放大采集电路来抑制共模并放大。电阻测量模块同时搭配两种不同的放大采集电路：一路采用仪用运算放大器INA333提供高共模抑制比的信号放大功能，再由TIVA Cortex M4自带的12位ADC进行数据采集；另一路直接采用Δ-Σ型ADC ADS1100进行采集，并通过I²C与TIVA Cortex M4连接，实现控制和采集数据的传送。

通过在模块上的跳线配置可以实现两种不同的前端测试电路和两种放大采集电路的交叉搭配，测试并比较不同配置方式的性能指标，获得对不同测量方式和各种模拟知识的理解。本模块涉及的知识点主要包括：共模信号和差模信号；仪用运算放大器的原理和特性；Δ-Σ型ADC的原理和应用；采用运算放大器和恒压源形成的恒流源电路；MCU的ADC和I²C的应用。

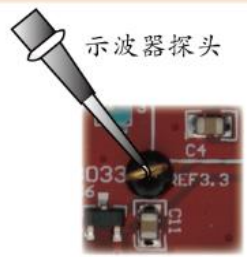
支持的实验项目：恒流源 + Δ-Σ型ADC测电阻；电桥 + Δ-Σ型ADC测电阻；恒流源 + 仪用运算放大器测电阻；电桥 + 仪用运算放大器测电阻



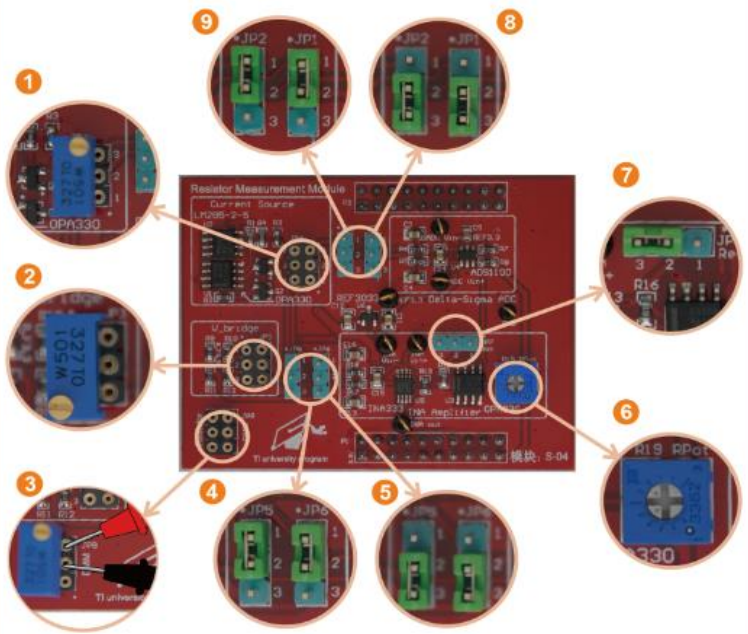
# 模块上测试环的布局说明




1. REF3033基准源输出，3.3V，用于测试基准源的输出性能
2. Δ-Σ型ADC ADS1100的信号输入负端
3. Δ-Σ型ADC ADS1100的信号输入正端，与 2 一起组成差分信号输入，用于测试输入到ADC的信号
4. 信号地
5. 仪用运算放大器INA333的信号输入正端，与 7 一起组成差分信号输入对
6. 仪用运算放大器INA333的输出端。进入AD采样前的信号，改变被测电阻或调节INA333的参考电平大小，都可以看到输出信号的变化
7. 仪用运算放大器INA333的信号输入负端



# 模块上跳线位与测试位的布局说明



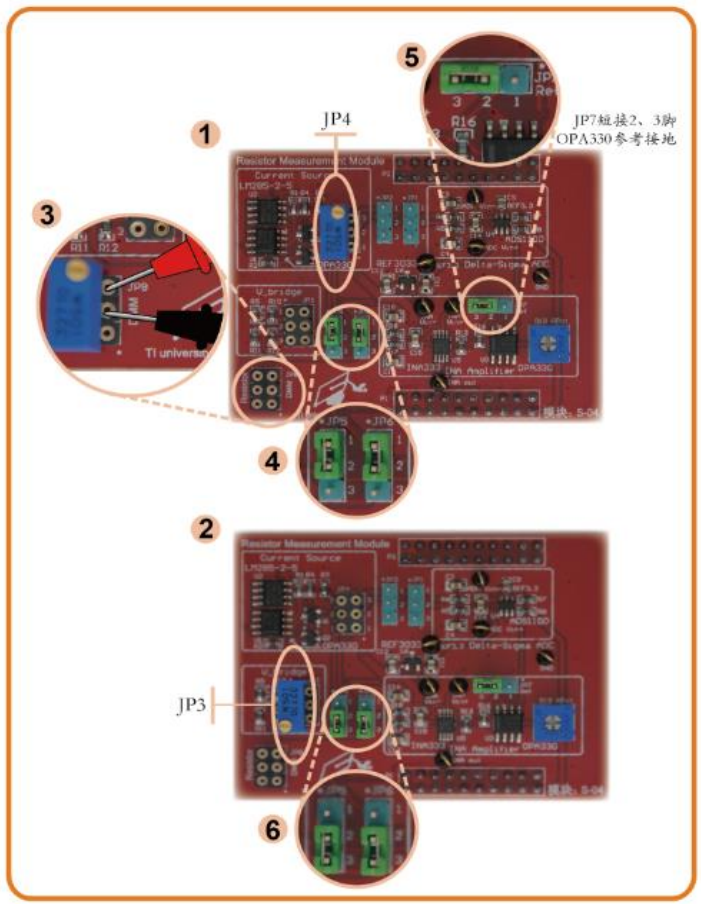

1. 圆针插座JP4，用于插装待测电阻，可以选用固定阻值的电阻，也可以是如图所示的多圈可调电阻。请注意模块上的丝印标记引脚号（1，2，3），2，3两个引脚间的电阻阻值为待测值
2. 圆针插座JP3，功能与1相同，也是2，3引脚间为待测值。请注意这个圆针插座的1，2，3标记顺序与1是相反的！
3. 圆针插座JP8。电阻测量实验时，通常需要先用高精度万用表来测一下待测电阻，再用模块上的测量电路测量待测电阻，比对测量结果来衡量测量电路的性能。用万用表直接测电阻时，两个表笔很难同时对上电阻的两个管脚，JP8的设计只为方便用万用表测量电阻，测量时只需将待测电阻放到插座上测量边上另一排圆针触点即可
4. 跳线位JP5,JP6，同时短接它们的1，2引脚，组成恒流源+仪用运算放大器的测量电路
5. 跳线位JP5,JP6，同时短接它们的2，3引脚，组成电桥+仪用运算放大器的测量电路
6. 单圈式电位器R19，用于调节INA333的参考电平
7. 跳线位JP7，用于选择INA333的参考电平是否受OPA330的输出电压控制，短接2、3号引脚，INA333参考电平选为零，不受参考电平调节电路影响
8. 跳线位JP1,JP2，同时短接它们的2，3引脚,组成电桥+Δ-Σ ADC式的测量电路
9. 跳线位JP1,JP2，同时短接它们的1，2引脚,组成恒流源+Δ-Σ ADC式的测量电路

## 实验一：恒流源+仪用运算放大器测电阻与电桥+仪用运算放大器测电阻的比较

**实验目的：** 掌握用电压基准源和运算放大器组成恒流源的方法；掌握并理解电桥法测电阻的原理与应用；理解并掌握仪用运算放大器的REF端的应用；理解仪用运算放大器的共模输入与输出范围的关系。

**实验步骤：**

1. 理解下图所示的等效原理图。
2. 自行编写或从网站(http://www.hpati.com)上下载软件代码，将代码烧写到TIVA内。有关TIVA操作请参考前三章的内容。
3. 将TIVA Launchpad、LCD模块、电阻测量模块分别插到母板的三个不同的槽位上。
4. 先将电路配置成：恒流源+仪用运算放大器。用跳线帽短接如 4 5 所示的三个位置的插针；上电。
5. 将一只500Ω的电位器如 3 所示插在JP8的三个孔中；调节金色旋钮；用万用表的两个表笔点测要接入的阻值,记录万用表读数。这个值作为待测电阻的理论值。记录数据到表4-1。
6. 将电位器如 1 所示插到JP4的三个孔中；要接入电阻的两个脚分别对应JP4的2、3孔；观察LCD模块上的显示值，这个值为实测值。记录数据到表4-1。调整电阻值，重复5，6两个步骤，记录数据并分析电阻测量的误差和测量的范围。
7. 改变电路配置成：电桥+仪用运算放大器。将JP5、JP6上的跳线帽换插到2、3两脚，如 6 ；将电位器如 2 所示插到JP3的三个孔中；要接入电阻的两个脚分别对应JP3的2、3孔；测量不同情况下的电阻的理论值和实测值，分析测量误差和测量范围。

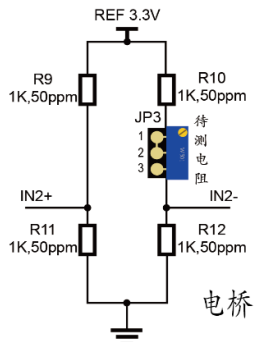



**附加实验内容：**

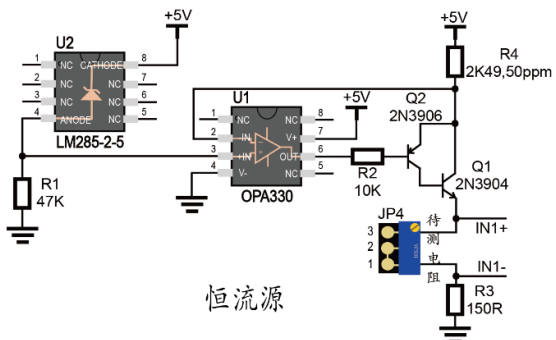
1、调节R19，通过OPA330(U3)来改变INA333的REF，测量INA333的输出与REF端电平的关系。

2、将待测电阻接到恒流源组态，改变电阻值，观测输入共模信号和差模信号输出范围的关系。

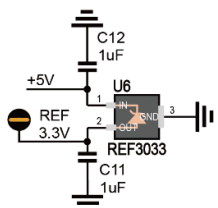
## 实验一：等效原理图



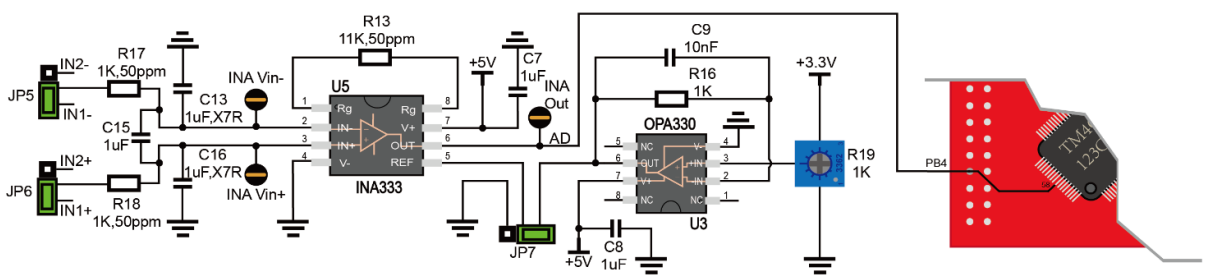

电桥




恒流源



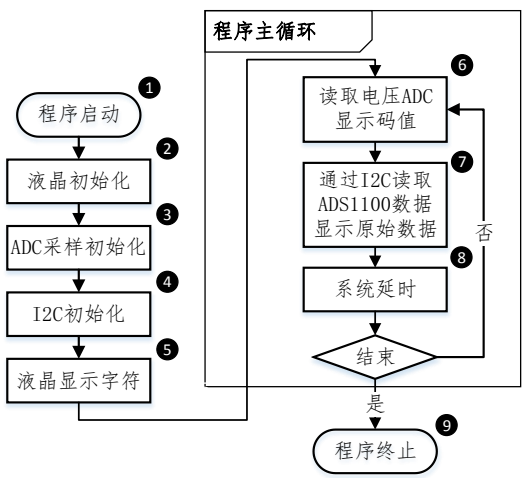

3.3V参考电压




仪用放大

LaunchPad

# 软件流程图




**电阻测量模块流程图**

1. USB线连接TIVA LaunchPad和计算机，使用CCS软件烧写程序。完成烧写后，TIVA LaunchPad上电后自行运行程序。

2. 液晶（LCD）初始化详细内容参见第三章LCD模块相关内容。

3. ADC采样外设端口的电平根据和参考电平的比较，将数字量化输出。PB4采集电平转化的ADC采样值。

4. ADS1100芯片通过I2C通信协议与TIVA LaunchPad进行通信和数据交换。外设端口PA6、PA7设置为I2C的时钟和数据信号线，实现与TIVA的通信。

5. LCD显示诸如INA333 Measure：xx V，ADS1100 Measure：xx V等信息，便于实验过程中观察实验数据变化。

6. 周期性的通过ADC采样获取电平的ADC采样值，并在LCD直接显示出ADC的转化码值。

7. 周期性的通过I2C通信采样获取ADS1100采样码值并通过LCD直接显示。

8. 为了防止LCD显示内容过快的重复刷新，通过主动系统延时将主循环控制在较低的刷新频率上。（如 5Hz）

9. 当TIVA LaunchPad掉电后，程序会跳出主循环,程序终止。

# 关键代码分析

根据软件流程图的分析可知。获取两种不同方式的电压数字量，一个需要初始化 Tiva LaunchPad 的 ADC 功能并实时采样；一个需要初始化 I2C 通信并将接收到的数据合成为有效数据。

## ADC 的初始化和数据采样

1、 ADC 初始化

```
/*************************************************************************
 * 初始化ADC获取滚轮电压值,用于电桥电路测量电阻
 *      _________|
//              |
//      M4   PB4|<--ADC0        模数转换信号源
//      _________|
*************************************************************************/
#define ADC_BASE       ADC0_BASE          // 使用ADC0
#define SequenceNum    3                  // 使用序列3
void Init_ADC_Detect(){
    // 使能 ADC0外设
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

```
    // 使能Port B外设端口
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    // 选择PB4作为模数装换ADC的管脚
    ROM_GPIOPinTypeADC(GPIO_PORTB_BASE, GPIO_PIN_4);
    // 配置采样序列的触发源和优先级
    ROM_ADCSequenceConfigure(ADC_BASE, SequenceNum, ADC_TRIGGER_PROCESSOR, 0);
    // 配置采样序列发生器的步进
    ROM_ADCSequenceStepConfigure(ADC_BASE, SequenceNum, 0, ADC_CTL_CH10 | ADC_CTL_IE |
                          ADC_CTL_END);
    // 使能一个采样序列
    ROM_ADCSequenceEnable(ADC_BASE, SequenceNum);
    // 清除采样序列中断源
    ROM_ADCIntClear(ADC_BASE, SequenceNum);
}
```

2、 ADC 数据采样

在程序主循环中以一定更新频率的不断采样 ADC 外设端口的电压值

```
while(1)
    {
        // 对while做125ms的延时，每秒刷新频率为8Hz
```

```
        ROM_SysCtlDelay(SysCtlClockGet() / 3 / 30);
        //ADC测电阻
        ADCProcessorTrigger(ADC_BASE, SequenceNum);
        // 等待完成AD转换
        while(!ADCIntStatus(ADC_BASE, SequenceNum, false))
        {
        }
        // 清楚ADC中断标志位
        ADCIntClear(ADC_BASE, SequenceNum);
        // 读取ADC值
        ADCSequenceDataGet(ADC_BASE, SequenceNum, pui32ADC0Value);
        // 根据参考电平3.3V将获取的数字量转化为实际电压值
        sample_Bridge_Average = (pui32ADC0Value[0] * 3300) / 4096;
        ...
    }
```

## 通过 I²C 控制 ADS1100，采集电压值

1、 I²C 通信初始化

```
/*****************************************************************************
```

```
 * 初始化AI2C获取ADS1100上的ADC电压数据,用于恒流源测量电阻
 *      ————————|
//              |
//      M4    PA6|<--SCL     I2C协议时钟信号
//            PA7|<--SDA     I2C协议数据信号
//      _________|
*************************************************************************/
void Init_I2C_Comm()
{
    // 使能I2C1外设
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);

    // 使能PortA外设端口
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // 配置PA6、PA7为上拉端口
    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);

    // PA6配置为I2C协议时钟信号、 PA7配置为I2C协议数据信号
    GPIOPinConfigure(GPIO_PA6_I2C1SCL);
```

```
    GPIOPinConfigure(GPIO_PA7_I2C1SDA);
    GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7);
    GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);

    // 初始化I2C主机模块。设置总线速度和使能主机模块
    I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);

    // 使能I2C主机模块
    I2CMasterEnable(I2C1_BASE);
}
```

2、 获取 ADS1100 通过 I2C 协议传输到 Tiva 的 ADC 数据

```
/*************************************************************************
 * 获取ADS1100上采集到的ADC数据
 * 通信协议:    1、设置读取的I2C从机地址（ADS1100）;
 *              2、获取16Bit ADC电压数据中的高8Bit;
 *              3、获取16Bit ADC电压数据中的低8Bit;
 *              4、得到ADS1100的配置信息
*************************************************************************/
```

```
uint32_t I2C_ADC_OpReg_MSB_i;   // 保存通过I2C读取ADS1100的16位AD的高字节
uint32_t I2C_ADC_OpReg_LSB_i;   // 保存通过I2C读取ADS1100的16位AD的高字节
uint32_t I2C_ADC_ConfigReg_i;
#define DELAY_6MS      (SysCtlClockGet() / 3) / 150000
void CatchI2C()
{
    // 恒流源测电阻
    I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);

    //*************************************************
    I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);
    while(I2CMasterBusy(I2C1_BASE));
    I2C_ADC_OpReg_MSB_i = I2CMasterDataGet(I2C1_BASE);
    //*************************************************
    I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
    while(I2CMasterBusy(I2C1_BASE));
    I2C_ADC_OpReg_LSB_i = I2CMasterDataGet(I2C1_BASE);
    //*************************************************
    I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);
    while(I2CMasterBusy(I2C1_BASE));
    I2C_ADC_ConfigReg_i = I2CMasterDataGet(I2C1_BASE);
```

```
}
```

3、 将 ADC 数据转化为实时电压数据

```
// 将高8位数据放置高8位上
temp1 = (I2C_ADC_OpReg_MSB_i & 0x000000FF) << 8 ;
// 将低8位数据放置低8位上
temp2 = (I2C_ADC_OpReg_LSB_i & 0x000000FF);
// 合并高低位数据为完整的采样数据
temp3 = (int16_t)(temp1 + temp2);
```

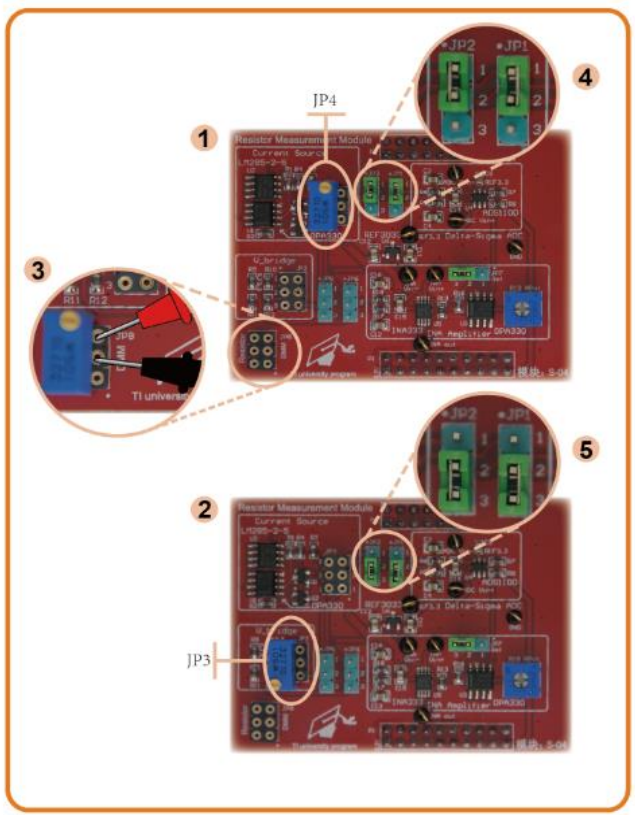
4、 得到的 temp3 就是最终 ADS1100 获得的 16 位电压数据采样值。记录下后可用于后面的数据测试和分析。

## 实验二：恒流源+Δ-Σ型ADC测电阻与电桥+Δ-Σ型ADC测电阻的比较

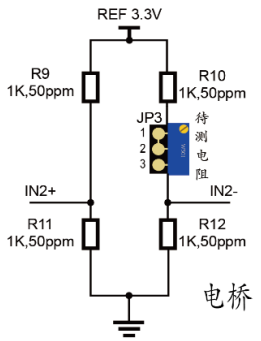
**实验目的：** 掌握用电压基准源和运算放大器组成恒流源的方法；掌握并理解电桥法测电阻的原理与应用；理解Δ-Σ型ADC差分输入方式和对共模的抑制；学习I²C控制的ADC的应用方式。

**实验步骤：**

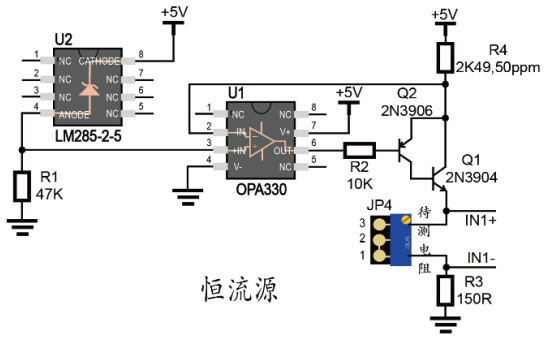
1. 理解下图所示的等效原理图。
2. 自行编写或从网站(http://www.hpati.com)上下载软件代码，将代码烧写到TIVA内。有关TIVA操作请参考前三章的内容。
3. 将TIVA Launchpad、LCD模块、电阻测量模块分别插到母板的三个不同的槽位上。
4. 先将电路配置成：恒流源+Δ-Σ型ADC。用跳线帽短接如 **4** 所示的两个位置的插针；上电。
5. 将一只500Ω的电位器如 **3** 所示插在JP8的三个孔中；调节金色旋钮；用万用表的两个表笔点测要接入的阻值；记录万用表读数。这个值作为待测电阻的理论值。记录数据到表4-2。
6. 将电位器如 **1** 所示插到JP4的三个孔中；要接入电阻的两个脚分别对应JP4的2、3孔；观察LCD模块上的显示值，这个值为实测值。记录数据到表4-2。调整电阻值，重复5，6两个步骤，记录数据并分析电阻测量的误差和测量的范围。
7. 改变电路配置成：电桥+Δ-Σ型ADC。将JP1、JP2上的跳线帽换插到2、3两脚，如 **5**；将电位器如 **2** 所示插到JP3的三个孔中；要接入电阻的两个脚分别对应JP3的2、3孔；测量不同情况下的电阻的理论值和实测值，分析测量误差和测量范围。



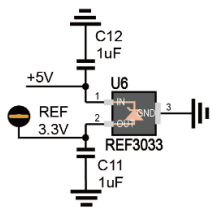
## 实验二：等效原理图



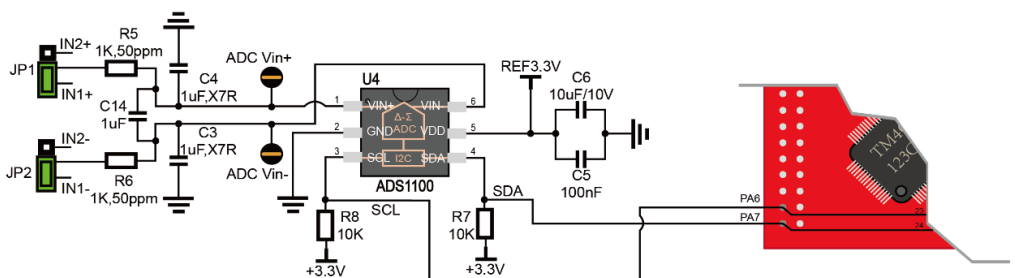

电桥




恒流源




3.3V参考电压




模数转换

LaunchPad

# 第5章　程控增益放大模块

杭州艾研信息技术有限公司

2014 年 11 月

# 申明

杭州艾研信息技术有限公司保留随时对其产品进行修正、改进和完善的权利，同时也保留在不作任何通告的情况下，终止其任何一款产品的供应的权利。用户在下订单前应及时获取相关信息的最新版本，并验证这些信息是当前的和完整的。

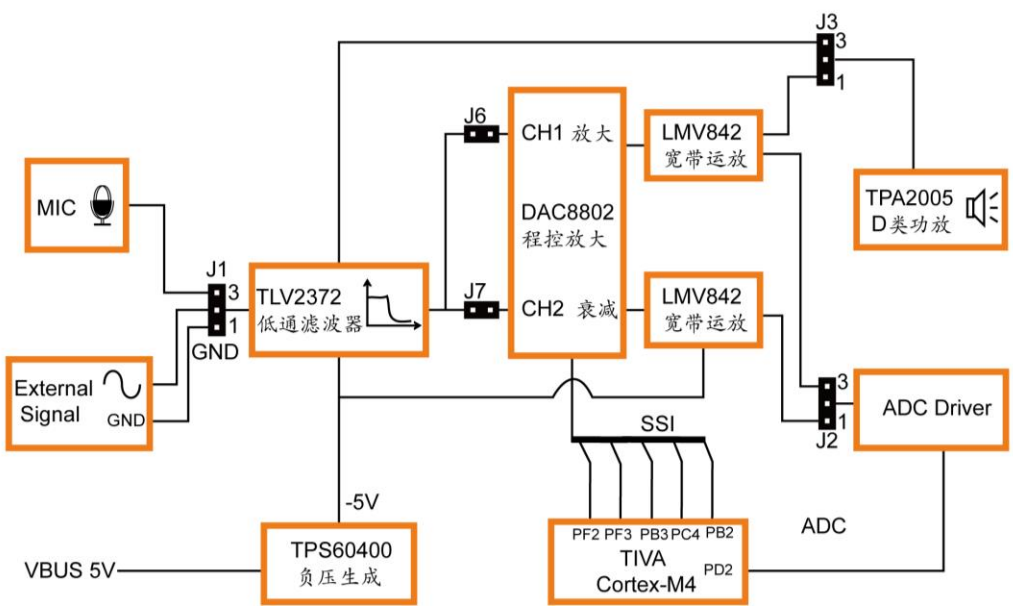
可通过如下方式获取最新信息、技术资料和技术支持：
技术支持电话：0571-86134572
技术支持邮箱：support@hpati.com
产品&资料下载中心：http://www.hpati.com/products/
互动论坛：http://www.hpati.com/bbs/forum.php
公司地址：浙江省杭州市西湖区留和路16号新峰商务楼B306

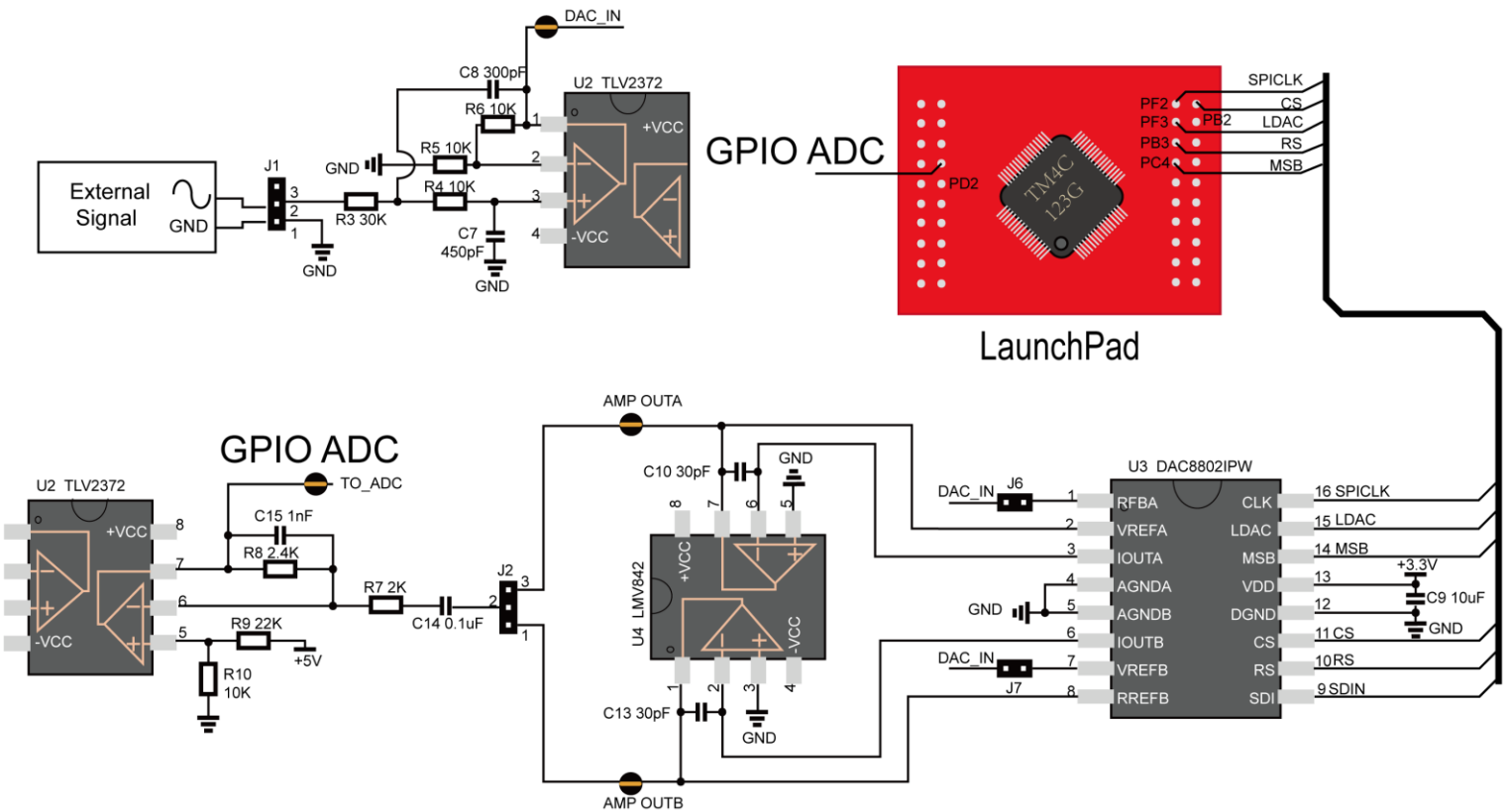
# 第5章　程控增益放大模块

# MDAC程控放大模块介绍

## 实验简介

本模块的核心部分为由一片14位双通道串行接口乘法数模转换器DAC8802芯片组成的程控放大和衰减部分，这部分中DAC8802芯片通过SSI总线与TIVA实现通信，由TIVA传输的CODE值决定放大/衰减幅值。另外，还可选择将外部信号或者程控放大衰减获得的信号作为输入，经过TPA2005D芯片得到音频输出信号，并通过喇叭外放。




DAC_IN
C8 300pF
U2 TLV2372
R6 10K
R5 10K
R4 10K
External Signal
GND
J1
R3 30K
C7 450pF
+VCC
-VCC
GPIO ADC
PD2
PF2
PF3
PB3
PC4
PB2
SPICLK
CS
LDAC
RS
MSB
LaunchPad
GPIO ADC
AMP OUTA
TO_ADC
C15 1nF
R8 2.4K
R7 2K
C14 0.1uF
J2
R9 22K
+5V
R10 10K
C10 30pF
U4 LMV842
C13 30pF
AMP OUTB
DAC_IN
J6
J7
U3 DAC8802IPW
RFBA
VREFA
IOUTA
AGNDA
AGNDB
IOUTB
VREFB
RREFB
CLK
LDAC
MSB
VDD
DGND
CS
RS
SDI
16 SPICLK
15 LDAC
14 MSB
13
12
11 CS
10 RS
9 SDIN
+3.3V
C9 10uF

1. 程序成功烧写后，TIVA LaunchPad上电后可启动程序运行。

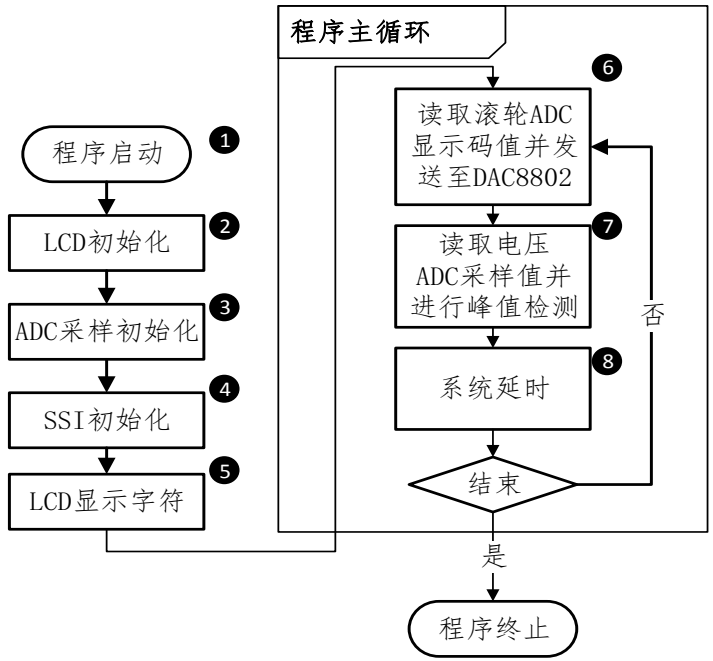
2. LCD初始化包括LCD端口使能、SSI通信协议配置、LCD配置初始化、LCD清屏4个步骤。每个步骤详情请见本书第三章。

3. ADC采样初始化有两个部分：滚轮电阻采样初始化(PE0)和峰值检测模块电压采样初始化(PD2)。ADC采样初始化步骤：1、使能ADC模块外设 2、配置相关GPIO为ADC功能 3、ADC采样序列配置 4、ADC采样序列步进配置 5、使能采样序列并清除中断标志。

4. SSI初始化负责完成与DAC8802之间通信的所有信号线的配置。包括PF2、PF3、PF1、PB2、PB3、PC4，其中PF2、PF3、PF1配置成SSI通信端口，其余这使能为端口输出。SSI初始化步骤：1、使能SSI外设模块。 2、配置相关GPIO复用功能为SSI模块功能并为SSI模块通信使用。 3、SSI通信模式、时钟频率设置和数据位设置。 4、使能SSI。

5. 在LCD上显示Peak detection voltage: xx V，等信息，便于实验过程中观察实验数据变化。

6. 在程序主循环中，读取滚轮ADC采样值,TIVA的AD是12-Bit，而DAC8802的DA是14-Bit，需要通过相应的转换得到DAC8802对应的码值并将码值发送至DAC8802，同时将码值显示到LCD上。读取ADC采样值的程序部分详见本书第三章。




7. 读取电压（PD2）ADC采样值，通过相应的转换公式转换成实际电压值并进行峰值检测，将检测到的峰值显示在LCD上，便于观察。此处读取ADC采样值的程序部分可参考本书第三章。

8. 为防止LCD过快重复刷新，系统进行延时将主循环控制在一个合理的执行速度内。

在程序设计过程中主要涉及到 LCD 显示，ADC 采样以及 SSI 通信，其中 LCD 显示和 ADC 采样功能设置可参考本书第三章相关内容。

Xxx SPI(SSI)通信

DAC8802 采用 SPI（相当于 Tiva M4 的 SSI 协议）通信协议进行数据传输。DAC8802 是 14-bit 的 DAC，而串行数据锁存在 DAC8802 的串行输入寄存器 (serial input register) 中，该寄存器为 16-bit 即两个字节长度（2-bit 的地址:A1-A0，以及 14 的 DA 数据：D13-D0）。寄存器数据格式如下：

表 xx SPI 寄存器数格式

| Bit | B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0(LSB) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Data | A1 | A0 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

DAC8802 内部有两个 DAC，分别为 DAC A 和 DAC B，其中 DAC A 为放大功能，DAC B 为衰减功能，程序控制 DAC8802 的放大和衰减功能就是选择使能其中一个 DAC 模块。该两个模块的选择通过串行输入寄存器中地址位：A1 和 A0 进行选择。

表 xx 地址位设置

| A1 | A0 | 使能 DAC 模块 |
|---|---|---|
| 0 | 0 | None |
| 0 | 1 | DAC A |
| 1 | 0 | DAC B |
| 1 | 1 | DAC A 和 DAC B |

程序上的实现可以采用#define 宏定义，然后在发送数据时将定义好的地址值加上待发送数据即可。宏定义代码如下：

```
#define DAC_A  0x4000   //DAC A
#define DAC_B  0x8000   //DAC B
#define DAC_AB 0xC000   //DAC A和DAC B
```

SPI(SSI)通信配置函数

DAC8802 通过 CS（低电平有效），SDI，SCK 三线控制数据的传输，其对应 Tiva M4 中的 SSIFss,SSITx，SSIClk 线，相应配置代码如下：

```
/**************************************************************************
* @brief SSI模块使能，并且设置相关端口初始状态
```

```
* @param none
* @return none
*       _________________
*                        |
*          PF2(SSI1Clk)|-->SPICLK  时钟信号端
*   TIVA   PF3(SSI1Fss)|-->SYNC    帧信号端
*          PF1(SSI1Tx) |-->SDIN    SSI数据发送端（LM4F120->DAC8802）
*          PB2(GPIO)   |-->LDAC
*          PB3(GPIO)   |-->RS
*          PC4(GPIO)   |-->MSB
*       _________________|
*
**********************************************************************/
void ssi_en()
{
    //使能外设SSI1模块
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
    //使能SSI1使用的外设GPIOF
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //SSI1端口功能使能
    //PF2复用功能配置为SSI1CLK，时钟线
    ROM_GPIOPinConfigure(GPIO_PF2_SSI1CLK);
    //PF3复用功能配置为SSI1FSS，片选线
    ROM_GPIOPinConfigure(GPIO_PF3_SSI1FSS);
    //PF1复用功能配置为SSI1TX，数据发送线
```

```
    ROM_GPIOPinConfigure(GPIO_PF1_SSI1TX);

    //LDAC置高
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);
    ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
    //RS置高
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);
    ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
    //MSB置高
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4);
ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4, GPIO_PIN_配置PF1，PF2，PF3供外设SSI1使用
    ROM_GPIOPinTypeSSI(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
    //端口模式:1M,16位数据
    ROM_SSIConfigSetExpClk(SSI1_BASE, ROM_SysCtlClockGet(),
                   SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 16);

    //使能SSISSIEnable(SSI1_BASE);
}
```

程序还涉及到 DAC8802 另外三根线的配置：LDAC 置高，RS 置高，MSB 置高。其中 LDAC 和 RS 都是低电平有效。LDAC 控制 DAC8802 的输出，RS 和 MSB 信号线连接至 DAC8802 内部的上电复位模块，复位时若 MSB=0，则所有寄存器值为 0x0000,若 MSB=1，则所有寄存器值为 0x2000。

DAC8802 的时序图如下：

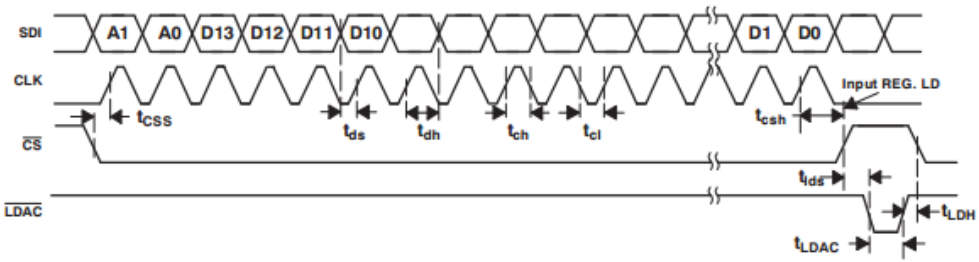



图 xx 时序图

根据时序图可以完成 DAC8802 跟 Tiva M4 之间的数据传输。在 SPI(SSI)配置程序中，时序图中的 SDI，CLK 以及 CS 线都配置成了 Tiva M4 中的 SSI 功能，在数据传输过程中这三根信号线上的电平变化都由 Tiva M4 的 SSI 模块自行控制。而 LDAC 线配置成普通的 GPIO 功能，则改线上的电平变化需要自行控制。SPI(SSI)传输程序代码如下：

```
/************************************************************
*   @brief      向dac8802发送数据
*   @param      unsigned long val，取值范围0~16384
*   @return     0， 参数不正确；
*              1，传输成功；
************************************************************/
unsigned char ssi_send_2_dac8802(unsigned long val)
{val > 16384) return 0;
    ROM_SSIDataPut(SSI1_BASE, DAC_AB + val);    //发数据+
    while(ROM_SSIBusy(SSI1_BASE));              //等待发送完成

    //数据发送结束时，LDAC线需要一个电平的跳变（H->L->H）
    ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0);
    delay();
    ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
```

```
    delay();
    return 1;
}
```

程序传输中使用 ROM_SSIDataPut 就可完成数据发送，发送的数据格式地址+数据即程序中的 DAC_AB+val,表示的是使能 DAC A 和 DAC B 同时使用两个 DAC 模块，val 即为需要发送的数据。此时同时完成放大和衰减，如果只要单独使用放大或者衰减，则发送数据时自需要变成 DAC_A+val 或 DAC_B+val。在数据发送完成后需要完成一个 LDAC 信号线的电平转变，完成 DAC8802 模块的输出。

# 程控放大和衰减

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、MDAC模块连接完成，准备实验。

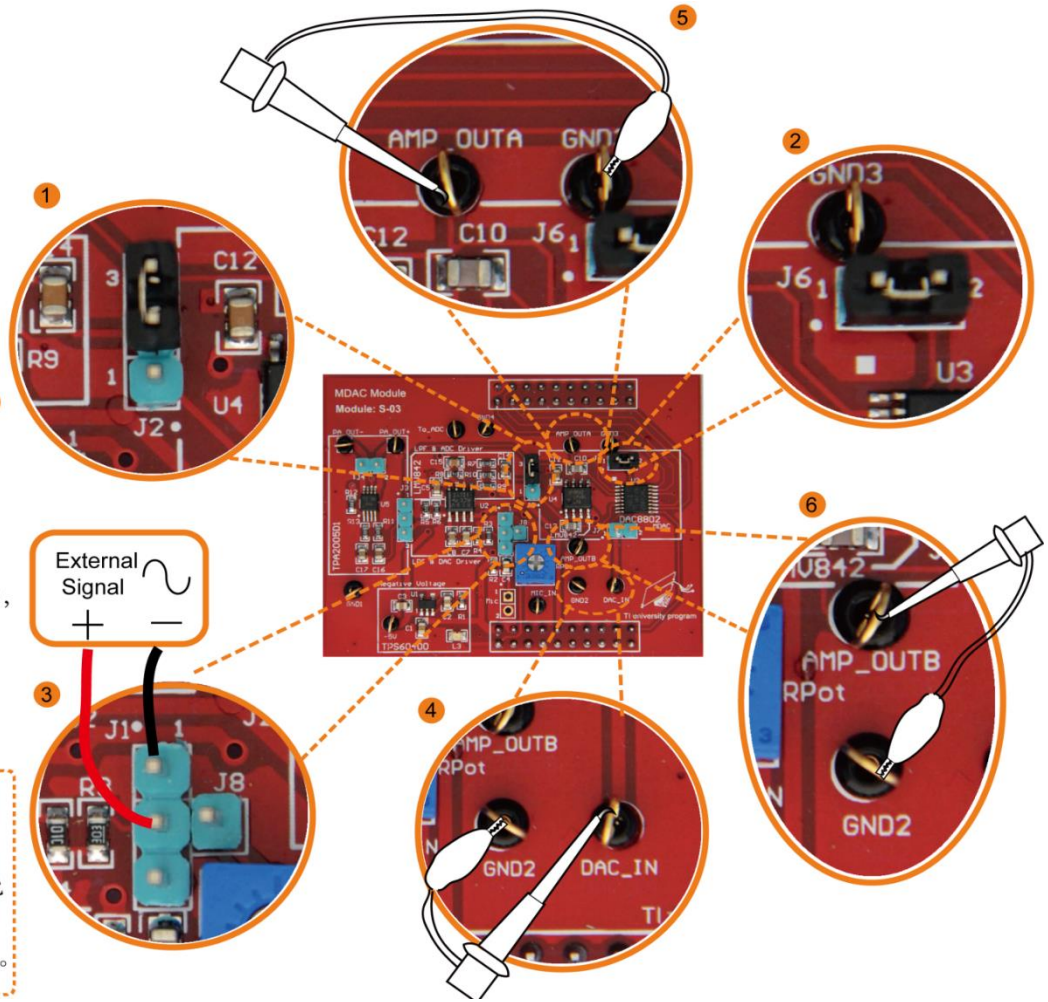
3、在MDAC模块上完成跳线连接，如图 1 ，2 所示，短接J2的 2 ，3 以及J6。

4、用杜邦线连接信号源，信号正极接J1的2，信号负极接J1的1，如图 3 所示。输入信号以1V，25Hz为宜。

5、打开TIVA开关，用示波器双踪观察DAC_IN以及AMP OUTA，如图 4 ，5 所示。调节滚轮，观察波形变化。

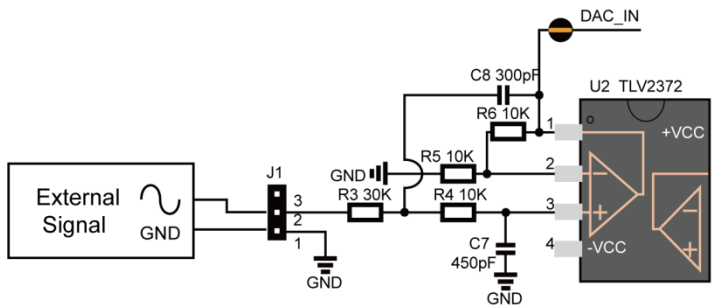
6、断电后，短接J2的 1 ，2 ，拔掉J6，短接J7。

7、用示波器双踪观察DAC_IN以及AMP OUTB，如图 6 所示。调节滚轮，观察波形变化。




**注意**

连接仪表及跳线时断开电源。并且电路初始状态输入到TIVA的管脚可能存在负压会导致TIVA的运行问紊乱。因此所有跳线完成之前系统不能上电。


DAC_IN
C8 300pF
U2 TLV2372
R6 10K
1
+VCC
R5 10K
2
J1
GND
External
Signal
GND
3
R3 30K
R4 10K
3
2
C7
450pF
4
-VCC
1
GND
GND

# 音频输入的信号调理

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、MDAC模块连接完成，准备实验。

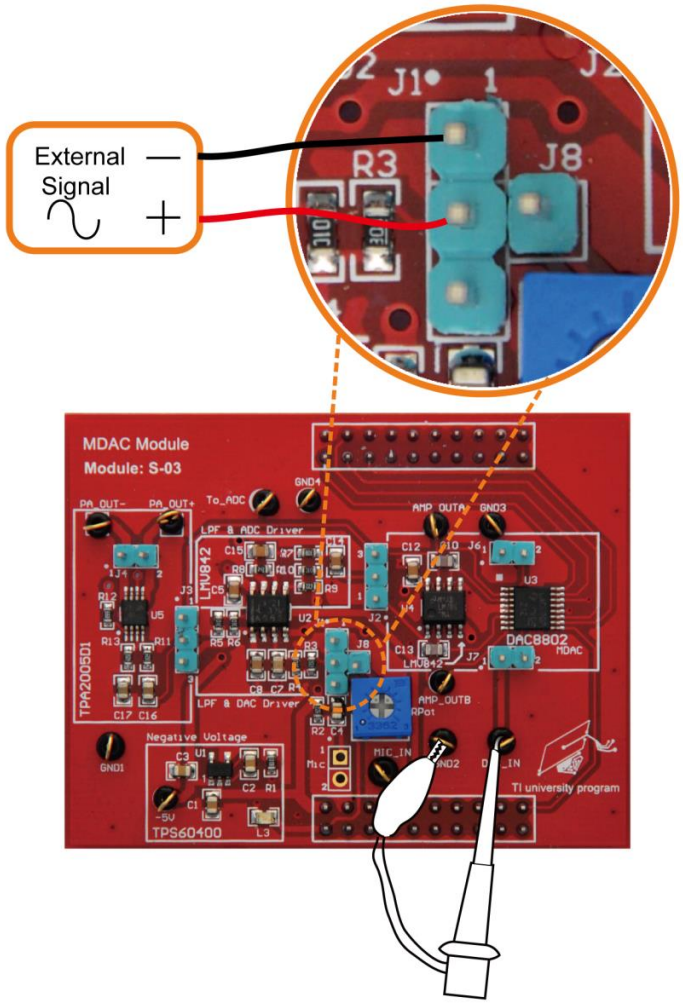
3、用杜邦线连接信号源，信号正极接J1的2，信号负极接J1的1，如右图所示。

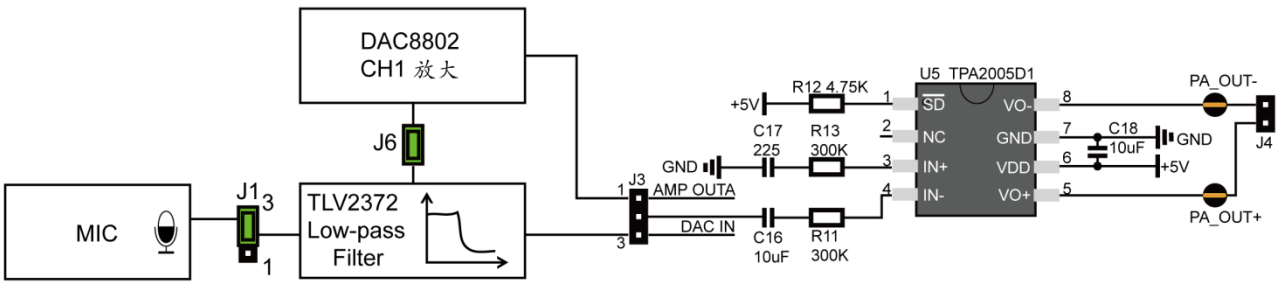
4、打开TIVA开关，用示波器观察DAC_IN的波形。

5、改变输入信号频率，用示波器观察输出的结果。

**注意**

连接仪表及跳线时断开电源。




DAC8802
CH1 放大
J6
MIC
J1
3
1
TLV2372
Low-pass
Filter
J3
1
3
GND
AMP OUTA
DAC IN
+5V
R12 4.75K
C17
225
R13
300K
C16
10uF
R11
300K
U5 TPA2005D1
SD
NC
IN+
IN-
VO-
GND
VDD
VO+
1
2
3
4
8
7
6
5
C18
10uF
GND
+5V
PA_OUT-
PA_OUT+
J4

# 音频输出功率推动

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、MDAC模块连接完成，准备实验。

3、在MDAC模块上完成跳线连接，如图1，2，3，4短接J6，J2的2,3，J3的2,3，J1的2,3。

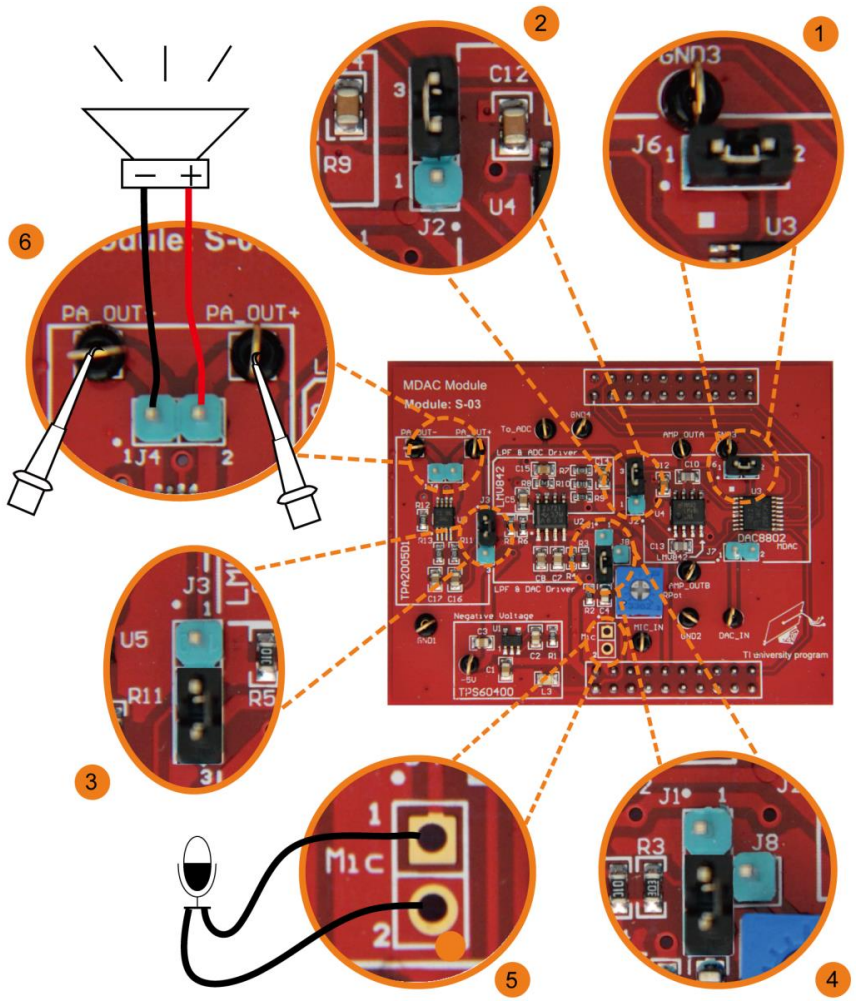
4、用杜邦线将话筒连接到MIC，如图5所示。将扩音器连接到J4，如图6所示。

5、打开TIVA开关，用示波器双踪观察PA_OUT+，PA_OUT-以及两个通道相减的波形。

6、断电后，短接J3的1，2，打开TIVA开关。调节滚轮，听声音的变化。

**注意**

连接仪表及跳线时断开电源。



# 第9章　宽带压控增益模块

杭州艾研信息技术有限公司

2014 年 11 月

# 申明

杭州艾研信息技术有限公司保留随时对其产品进行修正、改进和完善的权利，同时也保留在不作任何通告的情况下，终止其任何一款产品的供应的权利。用户在下订单前应及时获取相关信息的最新版本，并验证这些信息是当前的和完整的。

可通过如下方式获取最新信息、技术资料和技术支持：

技术支持电话：0571-86134572

技术支持邮箱：support@hpati.com

产品&资料下载中心：http://www.hpati.com/products/

互动论坛：http://www.hpati.com/bbs/forum.php

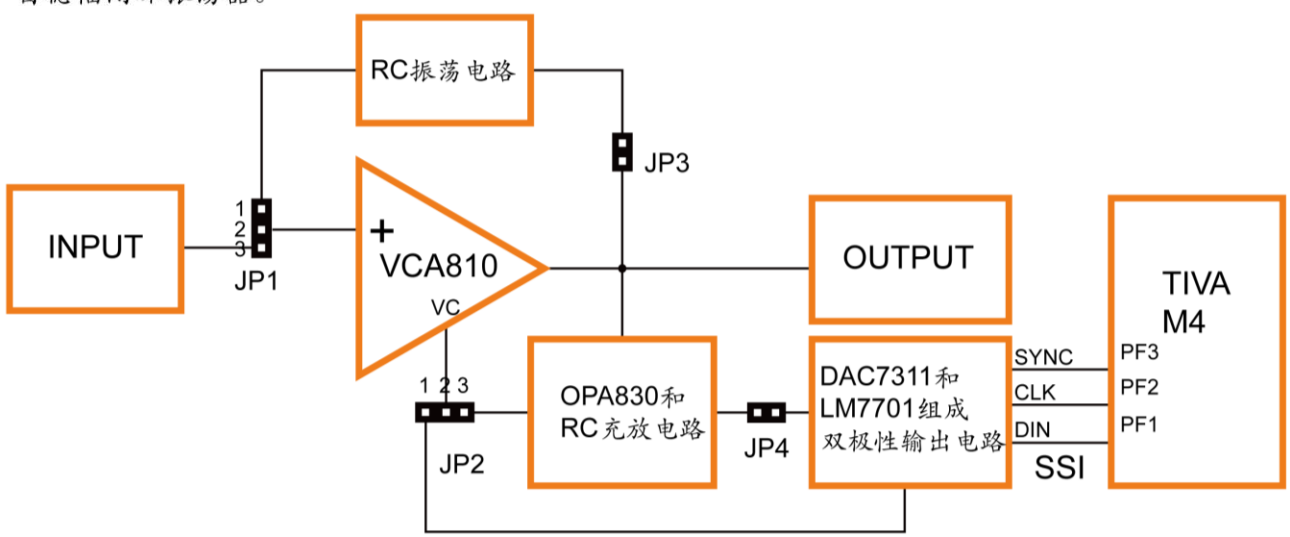
公司地址：浙江省杭州市西湖区留和路16号新峰商务楼B306

# 第9章　宽带压控增益模块

# 高速压控增益模块介绍

## 实验简介

本本模块采用一片高增益可调节范围宽带压控放大器VCA810组成高速压控增益电路。其中，VCA810的增益控制是通过调节控制输入端VC的电压来实现的。不同的控制电压，就可以得到不同的增益值，从而获得不同增益的输出值。该控制电压可以直接从DAC双极性输出电路中得到，并可通过液晶模块的滚轮调节大小。另外通过跳冒的选择，可以提供多种实验电路：1、宽带压控增益放大与衰减；2、正反馈RC振荡器；3、自稳幅闭环振荡器。



实验程序使用按键 S3 选择当前程序工作在实验 B，实验 C 还是实验 D，这三个实验程序上唯一的区别是：在程序刚运行时三个实验的 DAC7311 的初始值不同，其实部分都是一致的；使用按键 S1 和 S2 完成 VC 端电压值的调节，同时改变 DAC7311 的工作值，在液晶上同步显示当前 VC 端电压值；通过 SSI 传输协议改变 DAC7311 的工作值；

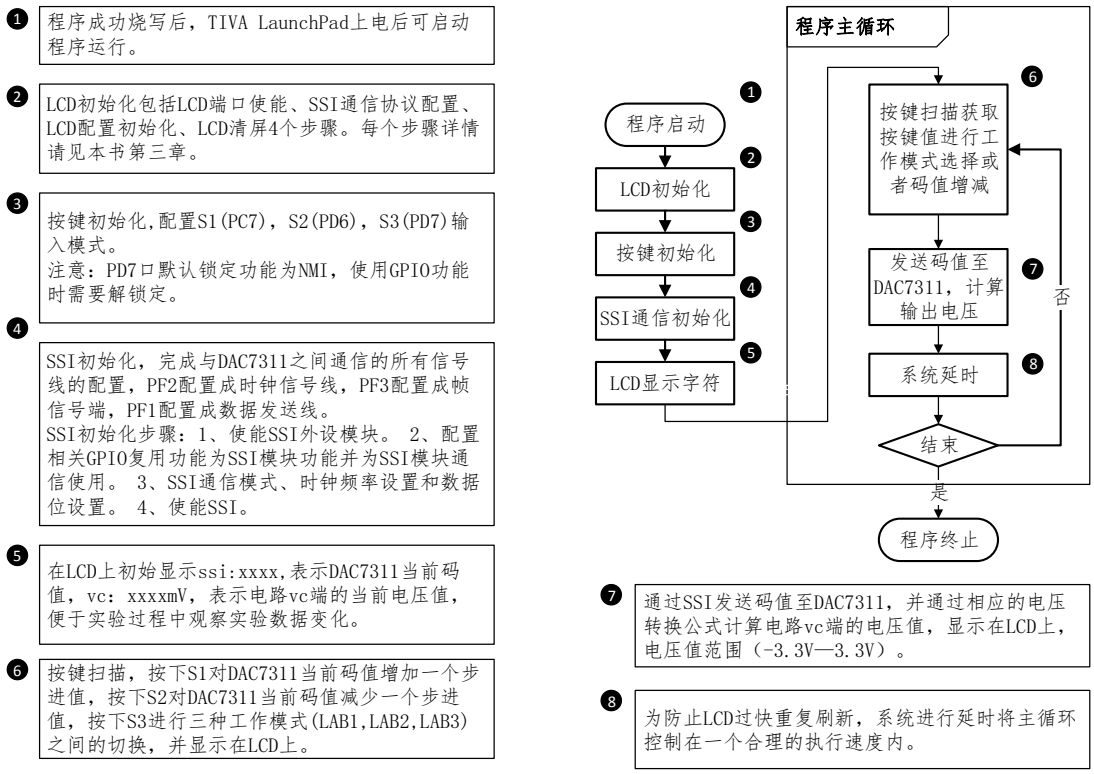



图 xx 程序流程图

LCD 显示部分程序可参考本书第三章。

按键功能

实验程序需要使用 LCD 开发板上的所有按键：S1、S2、S3。程序使用按键扫描完成端口状态的读取，在按键的初始化配置中需要注意的是 S3 的配置，因为 S3 连接在端口 PD7，而 PD7 口已经被锁定为 NMI（non-maskable interrupt，不可屏蔽中断）功能，所以在使用该端口时需要先解除锁定，使其能够配置成 GPIO 功能。解除锁定代码如下：

```
//解锁
HWREG(GPIO_PORTD_BASE+GPIO_O_LOCK)  |= GPIO_LOCK_KEY;
HWREG(GPIO_PORTD_BASE+GPIO_O_CR)    |= (1<<7);
HWREG(GPIO_PORTD_BASE+GPIO_O_DEN)   &=(~(1<<7));
HWREG(GPIO_PORTD_BASE+GPIO_O_PDR)   &= (~(1<<7));
```

```
HWREG(GPIO_PORTD_BASE+GPIO_O_PUR)  &= (~(1<<7));
HWREG(GPIO_PORTD_BASE+GPIO_O_AFSEL) &=(~(1<<7));
```

完成解锁后 PD7 口就可以跟 PC7，PD6 一样配置初始化使用。

按键使用端口的初始化程序代码如下：

```
 * @brief   对端口C、D进行按键初始化
 * @param   none
 * @return  none
 *      ________
 *             |
 *          PC7|<--Button1
 *  TIVA     PD6|<--Button2
 *          PD7|<--Button3
 *      _________|
 *  注：PD7口默认锁定功能为NMI，使用其GPIO功能时需要解锁定再配置成GPIO功能
 ************************************************************/
void Init_Key()
{
    //-------------------------------------------------------------
-
    //解锁
    HWREG(GPIO_PORTD_BASE+GPIO_O_LOCK)  |= GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE+GPIO_O_CR)   |= (1<<7);
    HWREG(GPIO_PORTD_BASE+GPIO_O_DEN)  &=(~(1<<7));
    HWREG(GPIO_PORTD_BASE+GPIO_O_PDR)  &= (~(1<<7));
    HWREG(GPIO_PORTD_BASE+GPIO_O_PUR)  &= (~(1<<7));
    HWREG(GPIO_PORTD_BASE+GPIO_O_AFSEL) &=(~(1<<7));
    //-------------------------------------------------------------
-
    //初始化外设GPIO
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    // 设置PD为2MA，若上拉输出
    ROM_GPIOPadConfigSet(GPIO_PORTC_BASE,  GPIO_PIN_7,
                         GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
    ROM_GPIOPadConfigSet(GPIO_PORTD_BASE,  GPIO_PIN_6,
                         GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
```

```
    ROM_GPIOPadConfigSet(GPIO_PORTD_BASE,  GPIO_PIN_7,
                         GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
    //设置GPIO输入模式
    ROM_GPIODirModeSet(GPIO_PORTC_BASE, GPIO_PIN_7,
                         GPIO_DIR_MODE_IN);
    ROM_GPIODirModeSet(GPIO_PORTD_BASE, GPIO_PIN_6,
                         GPIO_DIR_MODE_IN);
    ROM_GPIODirModeSet(GPIO_PORTD_BASE, GPIO_PIN_7,
                         GPIO_DIR_MODE_IN);
}
```

按键扫描程序直接使用 ROM_GPIOPinRead()进行读取按键所在端口的状态值，三个按键对应不同的功能：S3 选择实验项目，S1 增大 DAC7311 工作值使 VC 端电压增大，S2 减小 DAC7311 工作值使 VC 端电压减小。程序扫描三个端口即 PC7：S1 按键端口；PD6：S2 按键端口；PD7：S3 按键端口。

按键扫描程序代码如下：

```
/*************************************************************************
**
 * @brief   按键扫描函数
 * @param   none
 * @return  0x00            没有键按下
 *          0x01            按下PC7，S1
 *          0x02            按下PD6，S2
 *          0x03            按下PD7，S3
 *      ________
 *             |
 *          PC7|<--Button1
 *  TIVA    PD6|<--Button2
 *          PD7|<--Button3
 *      ________|
 *
 *************************************************************************
*/
unsigned char scan_key(void)
{
    if (ROM_GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_7) == 0x00)
```

```
    {
        // 延时约10ms，消除按键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000)); KEY抬起
        while (ROM_GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_7) == 0x00);
        // 延时约10ms，消除松键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000));
        return 0x01;
    }ROM_GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_6) == 0x00)
    {约10ms，消除按键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000));
        // 等待KEY抬起
        while (ROM_GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_6) == 0x00);
        // 延时约10ms，消除松键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000));
        return 0x02;
    }
    if (ROM_GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_7) == 0x00)
    {
        // 延时约10ms，消除按键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000));
        // 等待KEY抬起
        while (ROM_GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_7) == 0x00);
        // 延时约10ms，消除松键抖动
        ROM_SysCtlDelay(10*(ROM_SysCtlClockGet() / 3000));
        return 0x03;
    }
   return 0;
}
```

按键扫描函数的返回值表示实验中按下了不同按键，根据该值就可以完成工作模式转换或者是 VC 端电压的增减。按键响应程序代码如下：

```
key_val = scan_key();          //键扫描
if(key_val)
{
    switch(key_val)
    {
        //按下S1（按键1），增加
```

```
        case 0x01:
            //......
            break;
        //按下S2（按键2），减小
        case 0x02:
            //......
            break;
        //按下S3（按键3），工作模式的切换（实验之间的切换）
        //切换工作模式主要改变的是dac7311工作的默认码值（VC端）
        //开始工作的电压。
        case 0x03:
            switch(Key3_PressCount)
            {
                //工作模式1：带宽压控增益放大与衰减
                case 1:
                    //......
                    break;
                //工作模式2：正反馈RC震荡器
                case 2:
                    //......
                    break;
                //工作模式3：自稳幅闭环振荡器
                case 3:
                    //......
                    break;
                default:
                    break;
                }
                Key3_PressCount++;
                if(Key3_PressCount > 3)
                {
                    Key3_PressCount = 1;
                }
            default: break;
        }
    }
}
```

SSI 通信功能

DAC7311 与 Tiva M4 之间通过 SSI（SPI）通信完成。DAC7311 是 12-bit 的 DAC，其内部有一个 16-bit 的寄存器。寄存器格式如下：

| Bit | B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Data | PD1 | PD2 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | X | X |

寄存器低两位为无效位，B2~B13 为数据位，高两位 B14（PD2），B15（PD1）为控制选择位表示 DAC7311 不同的工作模式：一种正常工作模式，三种 power-down 工作模式。在实验程序只使用正常工作模式，这两位都配置为 0。

DAC7311 由 SYNC，SCLK，DIN 三线控制。SYNC 为信号选择线，相当于 Tiva M4 的 SSIFss 信号线。SCLK 为时钟信号线，相当于 Tiva M4 的 SSIClk 信号线。DIN 为数据线，相当于 Tiva M4 的 SSITx 信号线。SSI（SPI）通信程序设置如下：

```
/*************************************************************************
**
 *  @brief  SSI初始化设置
 *  @param  none
 *  @return none
 *
 *      ________________
 *                     |
 *          PF2(SSI1Clk)|-->SPICLK  时钟信号端
 *  TIVA    PF3(SSI1Fss)|-->SYNC    帧信号端
 *          PF1(SSI1Tx) |-->SDIN    SSI数据发送端（LM4F120->DAC7811）
 *      ________________|
 *
*************************************************************************
*/
void ssi_en()
{
    //使能SSI1外设
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
    //使能SSI1使用的GPIOF外设
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    //配置PF2复用功能为SSI1CLK，时钟线
```

```
    ROM_GPIOPinConfigure(GPIO_PF2_SSI1CLK);
    //配置PF3复用功能为SSI1FSS，片选线
    ROM_GPIOPinConfigure(GPIO_PF3_SSI1FSS);
    //配置PF1复用功能为SSI1TX，数据发送线
    ROM_GPIOPinConfigure(GPIO_PF1_SSI1TX);

    //配置PF1，PF2，PF3供SSI1使用
    ROM_GPIOPinTypeSSI(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
                       GPIO_PIN_3);
    //端口模式:1M,16位数据
    ROM_SSIConfigSetExpClk(SSI1_BASE, ROM_SysCtlClockGet(),
                     SSI_FRF_TI, SSI_MODE_MASTER, 1000000, 16);
    ROM_SSIEnable(SSI1_BASE);
}
```

DAC7311 寄存器的低两位无效，所以在 SSI 通信程序实现中需要将发送的数据左移两位，然后再通过 SSI 进行传输。程序代码实现如下：

```
/*************************************************************************
********
 *  @brief  发送数据到dac7311
 *  @param  val    取值范围0~4095
 *  @return 0   参数不正确;
 *          1   传输成功;
 ************************************************************/
unsigned char ssi_send_2_dac7311(unsigned long val)
{
    if(val > 4095) return 0;          //判断参数正确与否
    val = val << 2;                   //左移两位，DAC7311内部寄存器低两位无
效
    ROM_SSIDataPut(SSI1_BASE, val); //发送数据
    while(ROM_SSIBusy(SSI1_BASE));  //等待发送完成
    return 1;
}
```

实验程序需要在 LCD 上显示当前 VC 端的电压值，VC 端的电压可以通过 DAC7311 的输出电压换算得到。

DAC7311 的输出电压计算公式：

$$V_{out} = V_{DD} \times \frac{D}{2^n}$$

式中$V_{out}$是 DAC7311 的输出电压，$V_{DD}$是 DAC7311 的输入电压，$D$是 Tiva M4 通过 SSI 传输给 DAC7311 的工作值，$D \in [0, 4095]$，$n$是 DAC7311 的 DA 位数，该值为 12。

电路中$V_{DD} = 3.3V$，故 DAC7311 的输出电压为$0 \sim 3.3V$，该电压通过 LMP7701 构成的双极性输出电路将电压值转换为$-3.3V \sim 3.3V$，即为 VC 端的电压。在程序设计中不涉及负值的处理，故可以将$-3.3V \sim 3.3V$在程序中提升为$0 \sim 6.6V$。程序中可以通过如下方式计算出 VC 端的电压。

$$\Delta V = 2 \times V_{DD} \times \frac{D}{2^n} \qquad \Delta V \in [0, 6.6V]$$

故 VC 端电压为：

$$VC = \begin{cases} 3.3 - \Delta V & 0 \le \Delta V \le 3.3 \\ \Delta V - 3.3 & 3.3 < \Delta V \le 6.6 \end{cases}$$

程序中扩大 1000 倍计算，显示成 mV 档。程序代码如下：

```
//计算VC端的电压值
VC_Value = (dac7311_val*3300*2) / 4096;
//根据3300（中间值，进行正负显示处理）
int show_val;
if(VC_Value > 3300)
{
    show_val = VC_Value - 3300;
}
else
{
    show_val = 3300 - VC_Value;
}
```

程序代码中的 VC_Value 即为计算公式中的$\Delta V$，show_val 计算得出的都是正值，而根据程序数据跟 VC 端真实电压的对应关系可知当 VC_Value<3300（$\Delta V < 3.3$）时 VC 端电压为负值，LCD 上显示负值；VC_Value>3300（$\Delta V > 3.3$）时 VC 端电压为正值 LCD 上显示正值。程序代码如下：

```
for(i = 0; i < 4; ++i)
{
    if(VC_Value < 3300)
```

```
    {
        //在LCD上显示负号
        LCD_Draw_Char('-', LINE_TWO, 50);
    }
    else
    {
        //在LCD上清除负号
        LCD_Draw_Char(' ', LINE_TWO, 50);
    }
    //在LCD上显示VC端电压值
    LCD_Draw_Char('0' + data1[i], LINE_TWO, 60 + 8 * i);
}
```

不同的实验 VC 端以及 DAC7311 具有各自的初始默认值，VC 端的电压是通过 DAC7311 工作值计算获得。默认值在按键响应程序中设置。

A 宽带压控增益放大和衰减

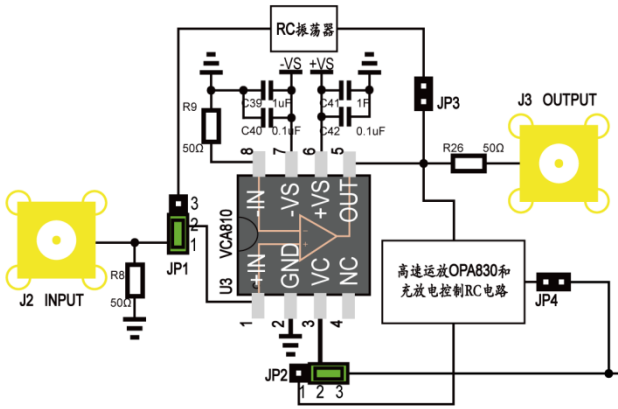
该实验初始默认值 VC 端电压需要为负压，保持在-1.13V。根据计算公式换算，在程序实现中保持 DAC7311 的工作值为 1350，计算得出的 VC 端的电压为-1.125V。

B 正反馈 RC 振荡器

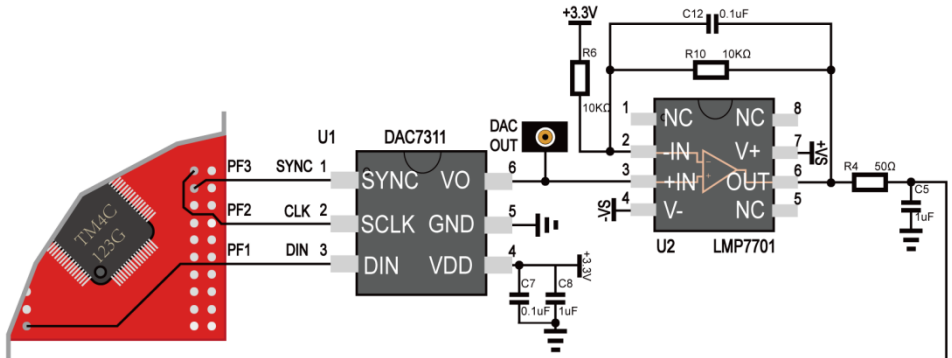
该实验初始默认值为 0 即可，此时 DAC7311 的工作值为 2048。

C 自稳幅闭环振荡器

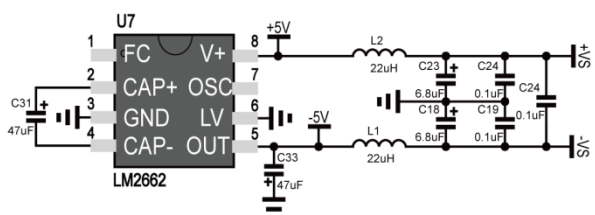
该实验初始默认值 VC 端电压需要为正压，保持在 1.17V。根据计算公式换算，在程序实现中保持 DAC7311 的工作值为 2775，计算得出 VC 端电压为 1.171V。




压控增益放大电路




双极性输出电路




电平转换及EMI滤波电路

# 宽带压控增益放大与衰减实验

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、高速压控增益模块连接完成，准备实验。

3、在高速压控增益模块上完成带宽压控增益放大与衰减实验的跳线连接，如图所示，短接图 1 JP1的1.2，以及JP2的2.3。

4、用实验套件里的两根高频连接线分别接在图 3 的J2和图 2 的J3上，其中J2连接到信号发生器，J3连接到示波器上。再万用表的红表笔接在跳线帽图 4 JP2上，黑表笔接地，用于测量VC的电压值大小。

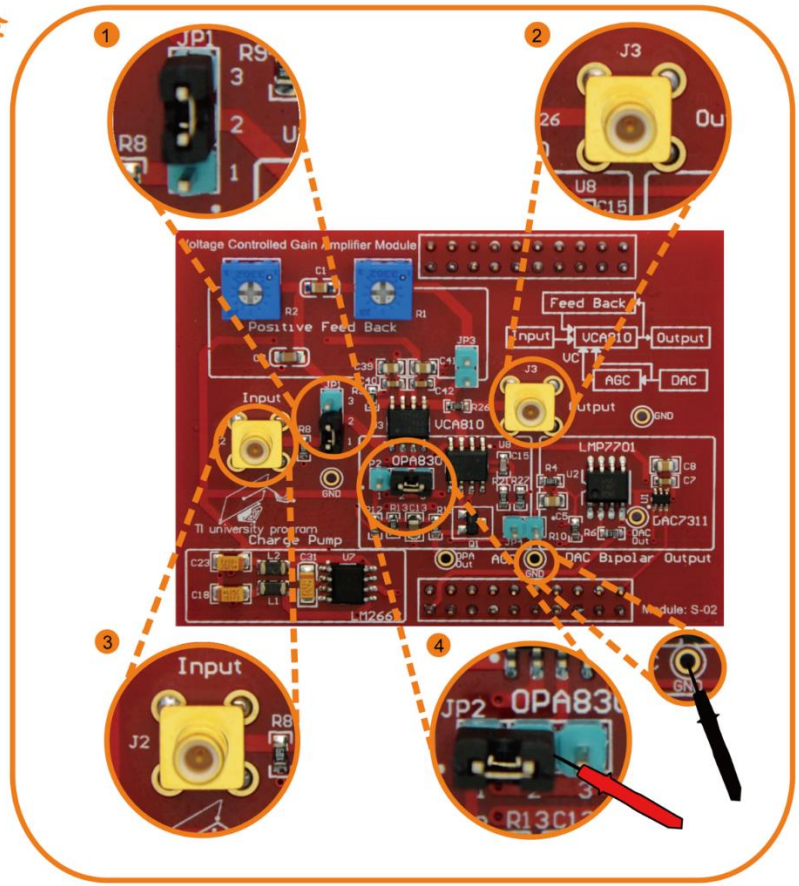
5、用信号发生器产生一个信号例如（10MHz 0.05Vpp），打开TIVA开关，通过液晶模块上的S1和S2按钮来调节VC的电压值（注意VC值要小于0），同时观察示波器上的输出信号，并记录，再计算增益与VC值的关系，查看增益线性度。

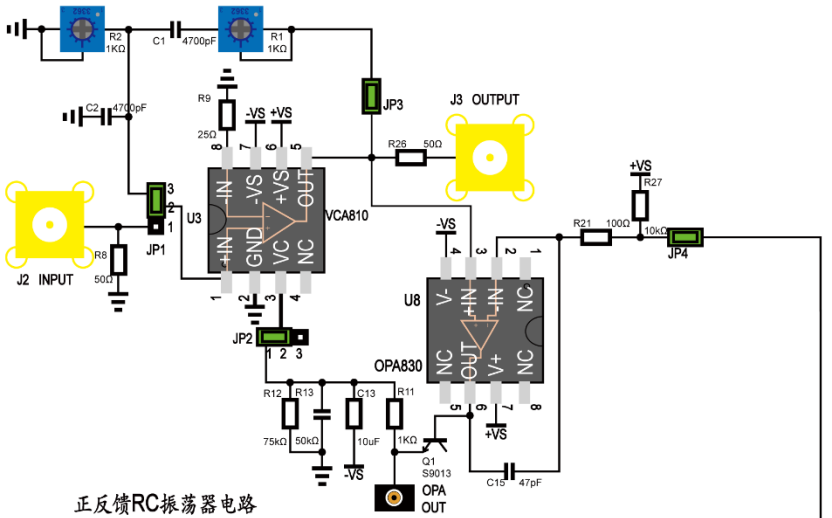
6、改变输入信号的幅值，例如（10MHz 2.5Vpp）重复步骤4，查看增益的线性度。

7、保持VC的值固定，例如（VC=-1.13V），保持输入信号的幅值，例如（Vin=1.5Vpp），改变输入信号的频率，测试增益，查看增益频率特性。

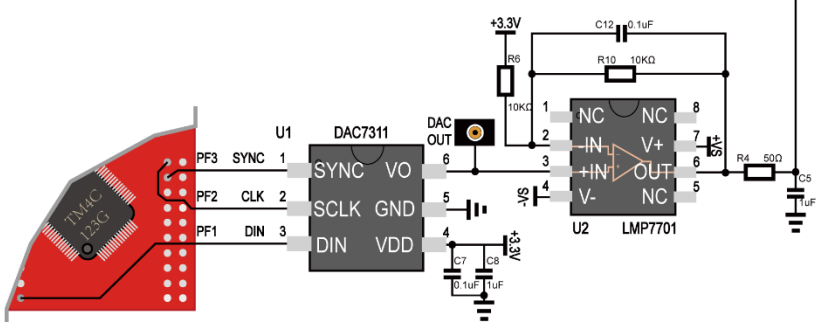
**注意**

连接仪表及跳线时断开电源。

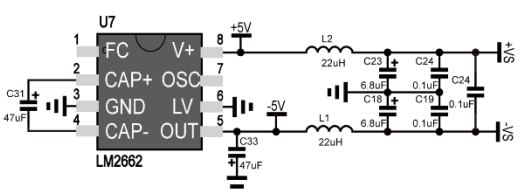




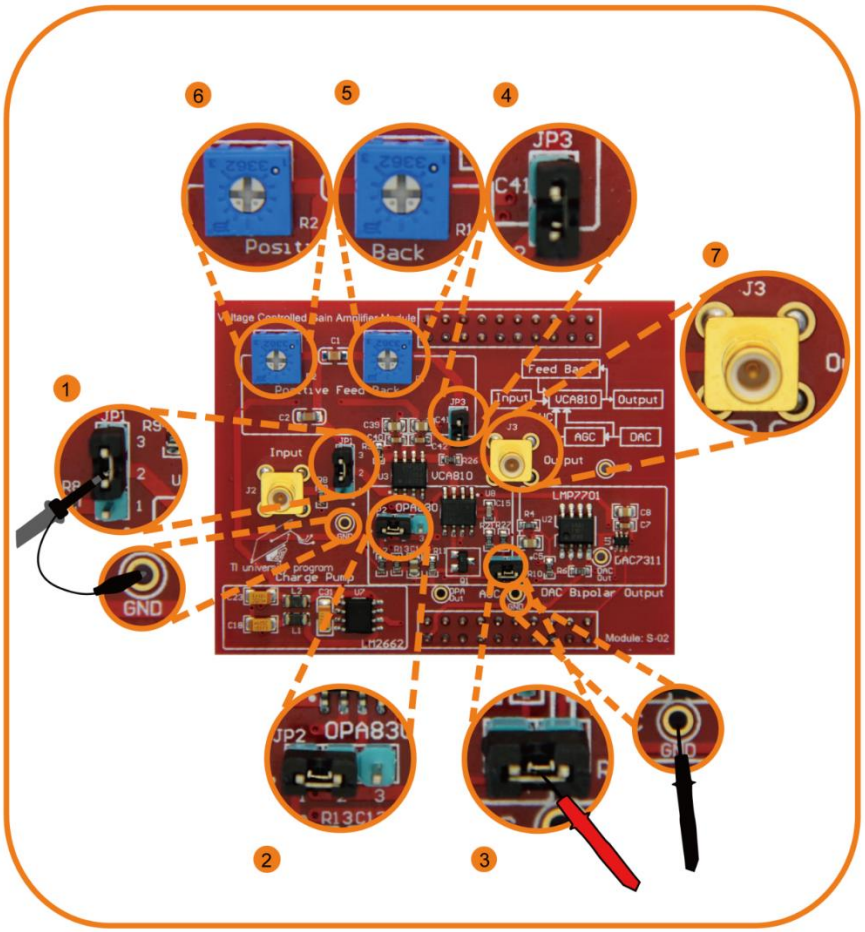

正反馈RC振荡器电路




双极性输出电路




电平转换及EMI滤波电路




# 正反馈RC振荡器实验

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、高速压控增益模块连接完成，准备实验。

3、在模块上完成正反馈RC振荡器的跳线连接，如图中的1.2.3.4所示，短接JP1的2.3，JP2的1.2,JP3以及JP4。

4、如图中1所示，用示波器测量RC振荡器产生的频率。如图中3所示，用万用表测量幅度控制电压的大小。如图中7所示，用高频连接线连接J3至示波器，观察输出

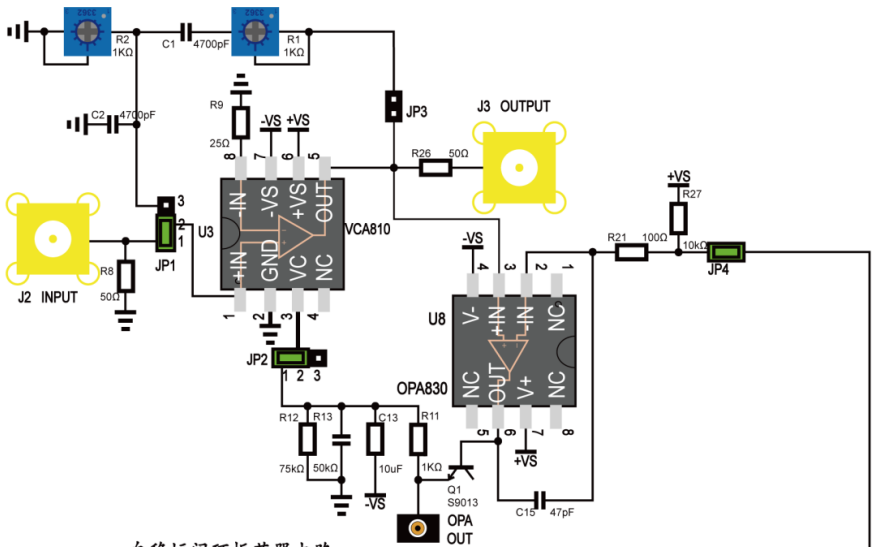
5、打开TIVA开关，通过调节滑动变阻器R1和R2的值来改变RC振荡器的输出频率。（注意同角度的调节滑变），再通过示波器可观察振荡器输出波形。

6、保持RC振荡器的输出频率不变，通过液晶模块上的S1.S2按钮来改变幅度控制电压的大小（JP4）。并测量输出幅度，查看振荡器幅度控制的线性度。

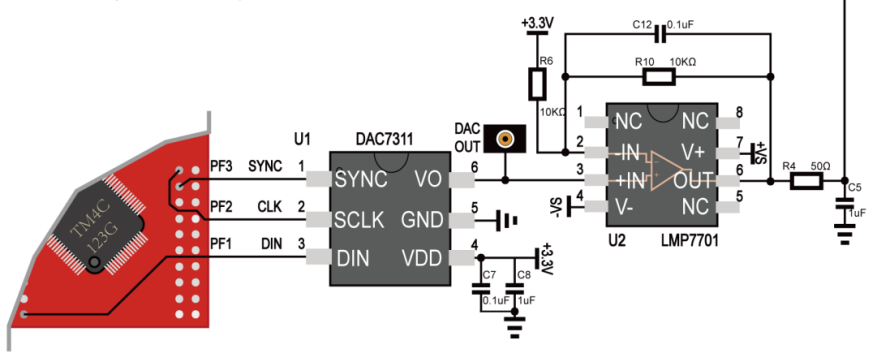
7、保持幅度控制电压大小不变，变化振荡频率，测试输出幅度，查看稳幅振荡情况。

**注意**

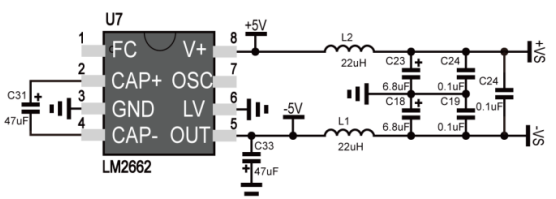
连接仪表及跳线时断开电源。




自稳幅闭环振荡器电路




双极性输出电路




电平转换及EMI滤波电路

# 自稳幅闭环振荡器实验

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、高速压控增益模块连接完成，准备实验。

3、在模块上完成自稳幅闭环振荡器的跳线连接，如图中的1.2.4所示，短接JP1的1.2，JP2的1.2,以及JP4。

4、用两根高频连接线分别接在图5的J2和图3 J3上，其中J2连接到信号发生器，J3连接到示波器上。

5、打开TIVA开关，用液晶模块上的S1.S2按钮来调节幅度控制电压的大小（JP4）,测量方式如图中的2。保持其电压为1.17V左右。

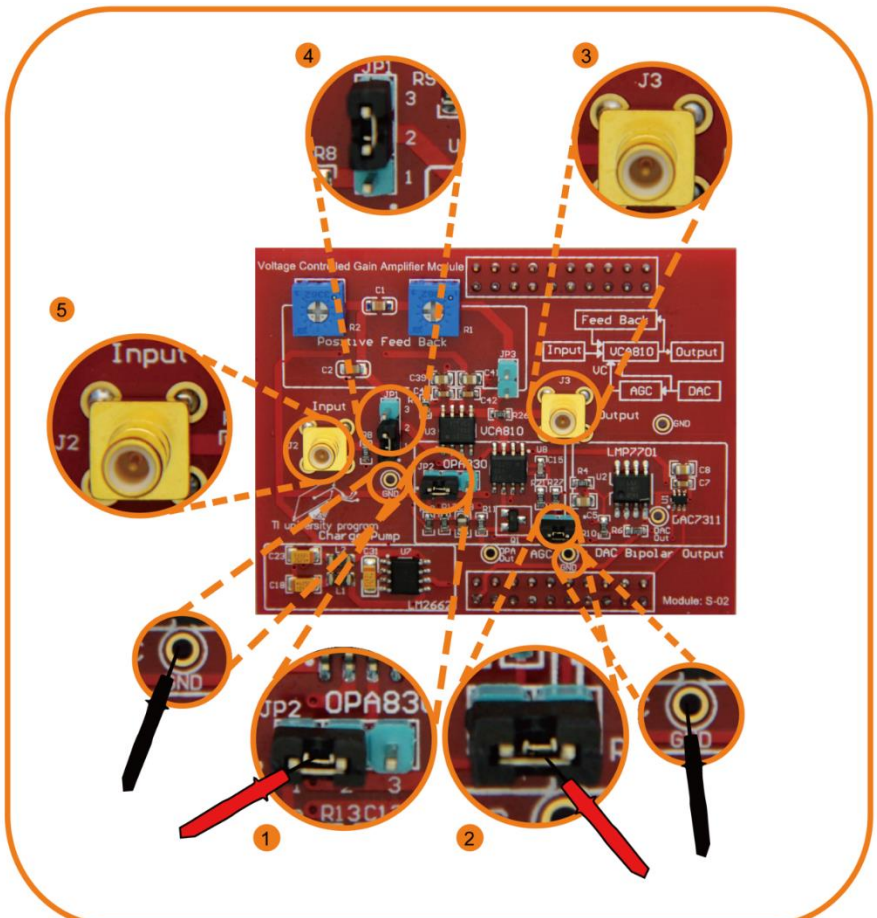
6、用信号发生器产生一个频率固定的输入信号，例如（100KHz），再改变输入信号的幅度，测试AGC的幅度稳定能力。（需测量输出端的电压值和VC的电压值）

7、使幅度控制电压为0.033V左右（JP4）。重复实验步骤5。

8、变化输入信号频率，测试AGC的幅度稳定能力。

**注意**

连接仪表及跳线时断开电源。



# 第10章　频率与相位跟踪模块

杭州艾研信息技术有限公司

2014 年 11 月

# 申明

杭州艾研信息技术有限公司保留随时对其产品进行修正、改进和完善的权利，同时也保留在不作任何通告的情况下，终止其任何一款产品的供应的权利。用户在下订单前应及时获取相关信息的最新版本，并验证这些信息是当前的和完整的。

可通过如下方式获取最新信息、技术资料和技术支持：

技术支持电话：0571-86134572

技术支持邮箱：support@hpati.com

产品&资料下载中心：http://www.hpati.com/products/

互动论坛：http://www.hpati.com/bbs/forum.php

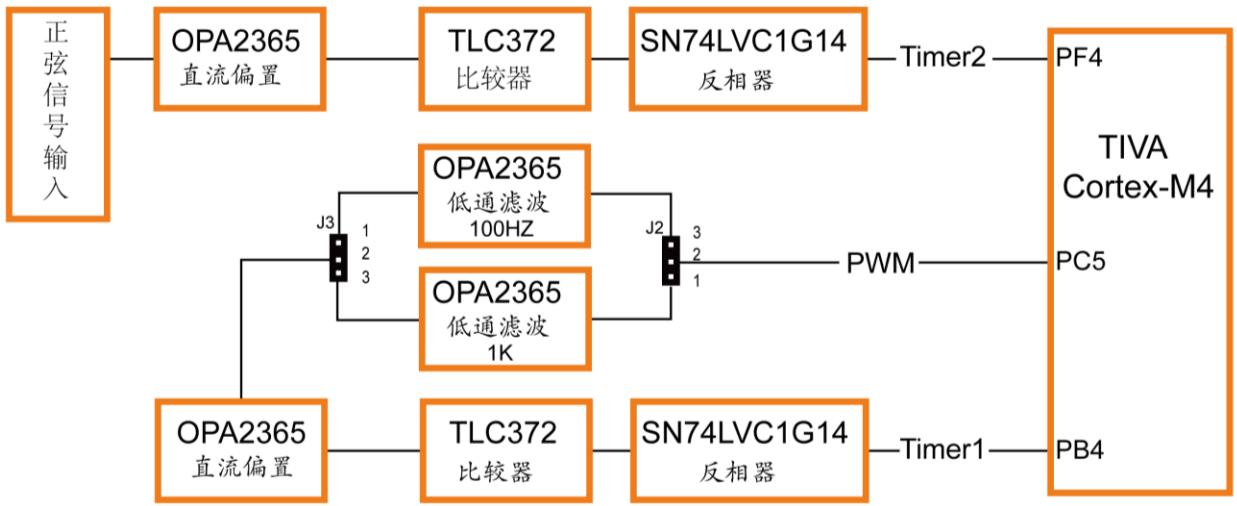
公司地址：浙江省杭州市西湖区留和路16号新峰商务楼B306

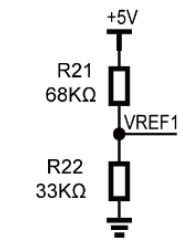
# 第10章　频率与相位跟踪模块

# 频率相位跟踪模块介绍

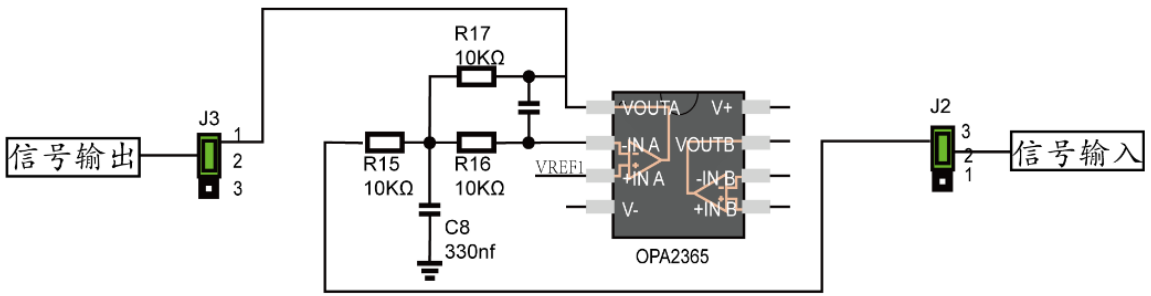
## 实验简介

本模块硬件电路主要由频率测量电路、相位跟踪电路和滤波电路三部分组成。由信号发生器产生的正弦波经过直流偏置使电路抬升电压后，再经过比较器进行整形，进入单片机，由单片机的定时器完成频率的测量。然后TIVA根据输入信号的频率输出一个SPWM信号，该信号经滤波电路，变成正弦波，送至相位跟踪电路；经偏置、整形和反向，送至TIVA的定时器。定时器对经频率测量电路和相位跟踪电路的两路信号进行鉴相，得到相位差。

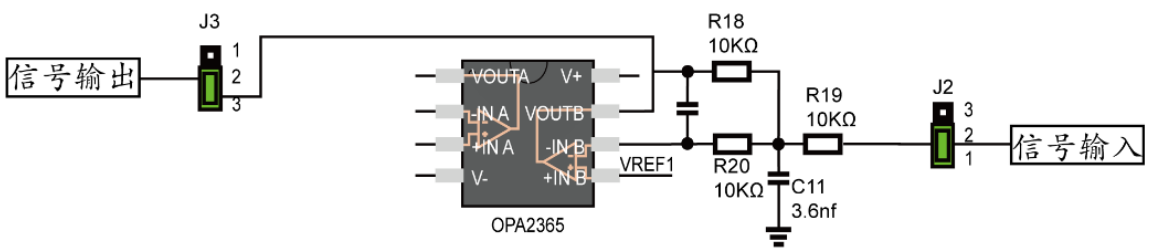





3.3V参考电压




截止频率为100Hz的低通滤波器




截止频率为1KHz的低通滤波器

# 低通滤波实验

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、频率相位跟踪模块连接完成，准备实验。

3、跳帽的连接：如图 1 和图 2 。在频率相位模块上用跳帽将J2的3、2和J3的1、2连接。

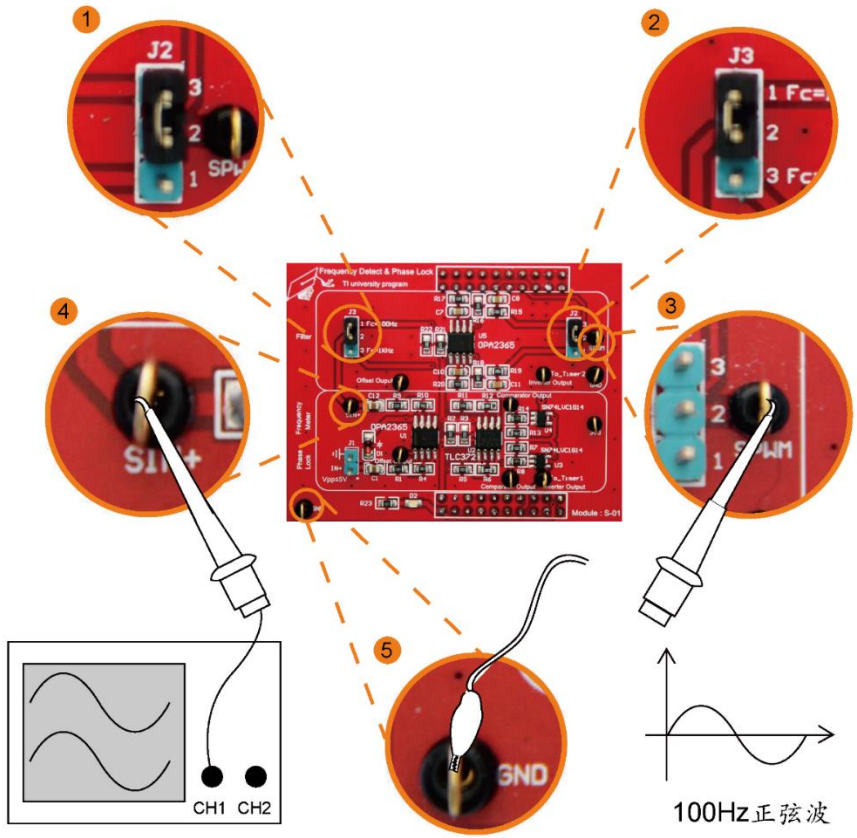
4、仪器连接：示波器表笔连接到图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

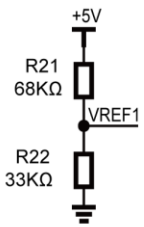
5、信号输入：将信号发生器的表笔连接到图 3 SPWM处，分别输入频率f<100Hz，100Hz<f<1KHz，f>1KHz正弦波，信号发生器同样要注意接地如图 5 。

6、打开TIVA开关，可以观察到LED点亮，在液晶上能看到信号发生器输入的正弦波的频率。同时在示波器上能观察到。滤波过后的信号波形。

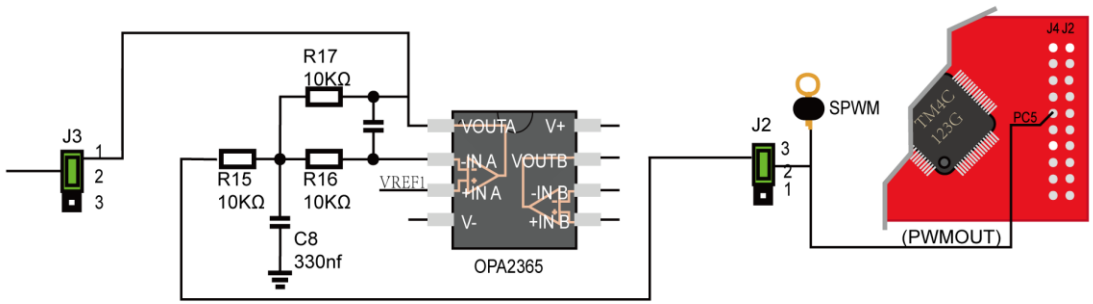
**注意**

连接仪表及跳线时断开电源。



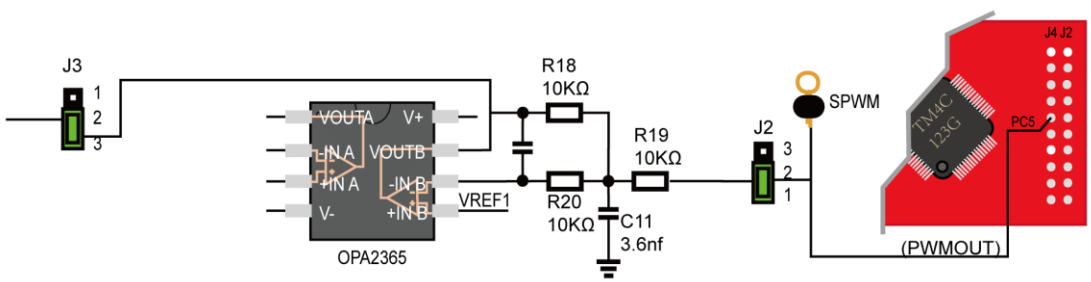



3.3V参考电压




频率为100Hz的正弦波

LaunchPad




频率为1KHz的正弦波

LaunchPad

# SPWM波的生成与正弦波发生实验

1、理解原理图以后编写Launchpad代码，代码可参考网上资源。然后烧写代码。

2、在母板上TIVA、液晶、频率相位跟踪模块连接完成，准备实验。

3、跳帽的连接：如图 1 和图 2 。在频率相位模块上用跳帽将J2的3、2和J3的1、2连接。

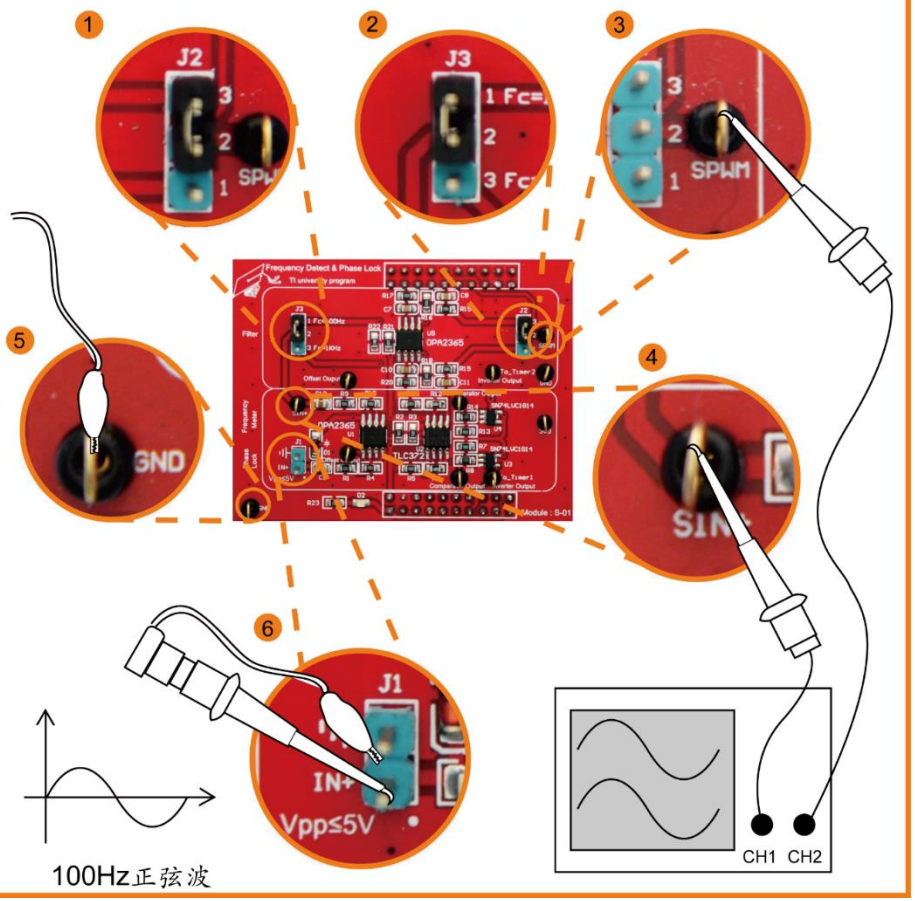
4、仪器连接：示波器两个表笔分别连接到图 3 所示的测试点。和图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

5、信号输入：将信号发生器的表笔连接到图 6 J1的IN+，输入频率100Hz的正弦波，信号发生器同样要注意接地如图 5 。

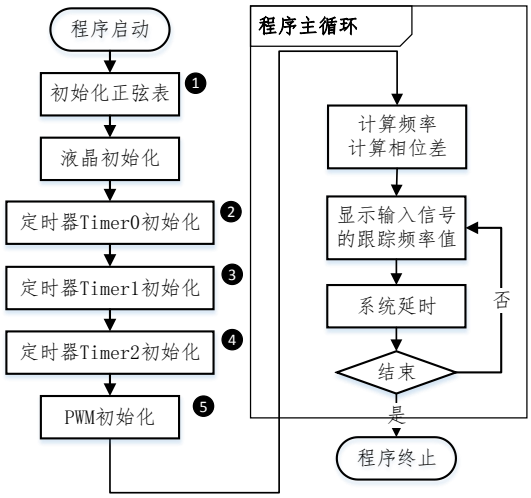
6、打开TIVA开关，可以观察到LED点亮，在液晶上能看到信号发生器输入的正弦波的频率。同时在示波器上能看到两路信号。一路是TIVA输出的占空比变化的方波。另一路是经过滤波以后的正弦波。

**注意**

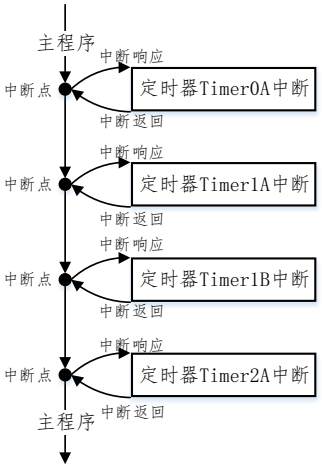
连接仪表及跳线时断开电源。另外信号发生器的输入信号改为1KHz的时候，只需要在步骤2用跳帽将J2的2、1和J3的2、3连接即可。



# 软件流程图




图x、频率相位模块流程图




图x、频率相位模块中断响应

❶ 创建2K大小数组存放标准的正弦表，用于后面生成一个SPWM波形时校对其占空比值。正弦表的最大值即为SPWM波形的最大周期。

❷ 程序创建一个信号跟踪采样频率为10KHz的定时器中断。每次响应中断会根据最新的频率和相位跟踪数据测算出SPWM的占空比值，并调节PWM输出。

❸ Timer1的TimerA开启捕获模式、TimerB启动周期模式。两者结合配置共同承担检测计算信号频率的任务。

❹ Timer2A的TimerA开启捕获模式来计算信号相位偏移。

❺ PWM初始负责生成一个10KHz的PWM载波信号。通过响应TimerOA中断后调节该PWM波形的占空比来形成一个SPWM波。后续电路通过运放等电路合成跟踪信号。

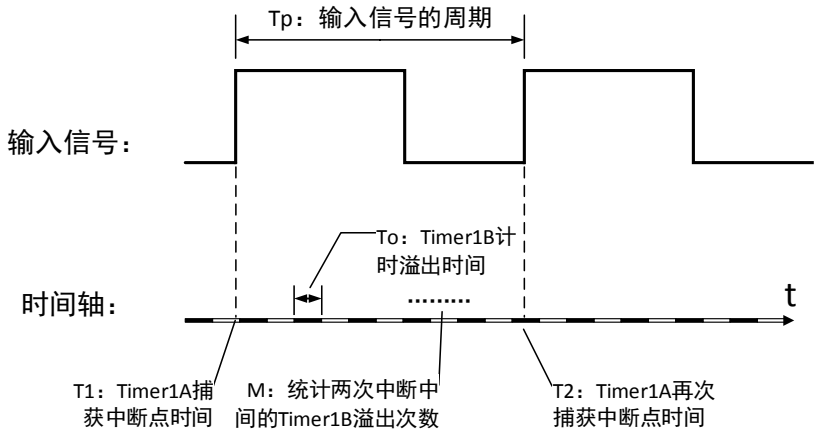
软件流程图如上图所示，频率相位的程序相对而言较为复杂。

1、 生成一个由 2048 点脉宽信息组成的正弦表，脉宽信息与正弦波幅值成一次函数关系。该表用于调整输出 PWM 信号的占空比值。

2、 定时器 Timer0 产生一个 10K 频率的定时周期响应，每次响应都会调整 PWM 的输出占空比值。

3、 定时器 Timer1 中的 Timer1A 和 Timer1B 同时启动，用于测量输入频率。其中 Timer1A 采用捕获模式响应输入信号的硬件中断。Timer2B 设定为溢出计时的工作模式。根据不同的输入频率计算输入信号的周期后换算得到信号频率；如下图所示，输入信号上升沿触发可以得到 T1，T2 两个中断点的定时器值。根据下图的构思可以得到输入信号的周期（Tp）：

$$Tp = T2 - T1 + To * M$$




信号频率测算示意图

最后根据系统频率信息就能够换算得到输入信号的精确频率信息。

## 关键代码分析

### 1、初始化 Timer0 定时器，生成 SPWM。

```
/*************************************************************
 * @brief    初始化Timer0定时器中断，以10K的频率响应定时中断，
 *         通过调节PWM信号的占空比输出一组SPWM信号
 * @param   null
 * @return  null
 *************************************************************/
void Init_Timer0_SPWM() {

    // Enable the peripherals used by this example.
    ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_TIMER0);

    // Enable processor interrupts.
    ROM_IntMasterEnable ();

    // Configure the two 32-bit periodic timers.
    ROM_TimerConfigure (TIMER0_BASE, TIMER_CFG_PERIODIC);
```

```
    // 8K Hz响应
    ROM_TimerLoadSet (TIMER0_BASE, TIMER_A,
           ROM_SysCtlClockGet() / SAMPLE_FREQUENCY);

    // Setup the interrupts for the timer timeouts.
    ROM_IntEnable (INT_TIMER0A);

    ROM_TimerIntEnable (TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // Enable the timers.
    ROM_TimerEnable (TIMER0_BASE, TIMER_A);

}
```

## 2、初始化 Timer1 定时器，测算输入信号频率

```
/*************************************************************
 * @brief    初始化Timer1定时器中的Timer1A和Timer1B中断，用于测算输入信号频率
 *           Timer1A工作在捕获模式；
 *           Timer1B工作在计数溢出模式；
 * @param   null
 * @return  null
 *************************************************************/
void Init_Timer1_Frequency() {
    // 启用Timer1模块
    ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_TIMER1);

    // 启用GPIO_M作为脉冲捕捉脚
    ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOB);

    // 配置GPIO脚为使用Timer4捕捉模式
    ROM_GPIOPinConfigure (GPIO_PB4_T1CCP0);
    ROM_GPIOPinTypeTimer (GPIO_PORTB_BASE, GPIO_PIN_4);

    // 为管脚配置弱上拉模式
    ROM_GPIOPadConfigSet (GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
           GPIO_PIN_TYPE_STD_WPU);
```

```
    // 配置使用Timer4的TimerA模块为边沿触发减计数模式
    ROM_TimerConfigure (TIMER1_BASE,
            TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_CAP_TIME |
TIMER_CFG_B_PERIODIC);

    // 使用下降沿触发
    ROM_TimerControlEvent (TIMER1_BASE, TIMER_BOTH, TIMER_EVENT_NEG_EDGE);

    // 设置计数范围为0x8FFF~0X8FFA
    ROM_TimerLoadSet (TIMER1_BASE, TIMER_A, 0xFFFF);
    ROM_TimerLoadSet (TIMER1_BASE, TIMER_B, 0xFFFF);

    // 注册中断处理函数以响应触发事件
    TimerIntRegister(TIMER1_BASE, TIMER_A, Int_Timer1A_Handler);
    TimerIntRegister(TIMER1_BASE, TIMER_B, Int_Timer1B_Handler);

    // 系统总中断开
    ROM_IntMasterEnable ();

    // 时钟中断允许，中断事件为Capture模式中边沿触发，计数到达预设值
    ROM_TimerIntEnable (TIMER1_BASE, TIMER_CAPA_EVENT |
TIMER_TIMB_TIMEOUT);

    // NVIC中允许定时器A模块中断
    ROM_IntEnable (INT_TIMER1A | INT_TIMER1B);

    // 启动捕捉模块
    ROM_TimerEnable (TIMER1_BASE, TIMER_BOTH);
}
```

### 3、初始化 Timer2 定时器，计算输入输出信号相位差：

```
/*************************************************************
 * @brief    初始化Timer2定时器中的Timer2A中断，用于测算输入信号和输出信号的时间差
 *         Timer2A工作在捕获模式；
 * @param   null
 * @return  null
 *************************************************************/
```

```
// 初始化相位跟踪
void Init_Timer2_Phase() {
    // 启用Timer4模块
    ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_TIMER2);

    // 启用GPIO_M作为脉冲捕捉脚
    ROM_SysCtlPeripheralEnable (SYSCTL_PERIPH_GPIOF);

    // 配置GPIO脚为使用Timer4捕捉模式
    ROM_GPIOPinConfigure (GPIO_PF4_T2CCP0);
    ROM_GPIOPinTypeTimer (GPIO_PORTF_BASE, GPIO_PIN_4);

    // 为管脚配置弱上拉模式
    ROM_GPIOPadConfigSet (GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
           GPIO_PIN_TYPE_STD_WPU);

    // 配置使用Timer4的TimerA模块为沿触发加计时模式
    ROM_TimerConfigure (TIMER2_BASE,
           TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_CAP_TIME);

    // 使用下降沿触发
    ROM_TimerControlEvent (TIMER2_BASE, TIMER_A, TIMER_EVENT_NEG_EDGE);

    // 设置计数范围为0~0x8FFF
    ROM_TimerLoadSet (TIMER2_BASE, TIMER_A, 0xFFFF);

    // 注册中断处理函数以响应触发事件
    TimerIntRegister(TIMER2_BASE, TIMER_A, Int_Timer2A_Handler);

    // 系统总中断开
    ROM_IntMasterEnable ();

    // 时钟中断允许，中断事件为Capture模式中边沿触发
    ROM_TimerIntEnable (TIMER2_BASE, TIMER_CAPA_EVENT);

    // NVIC中允许定时器A模块中断
    ROM_IntEnable (INT_TIMER2A);
```

```
    // 启动捕捉模块
    ROM_TimerEnable (TIMER2_BASE, TIMER_A);
}
```

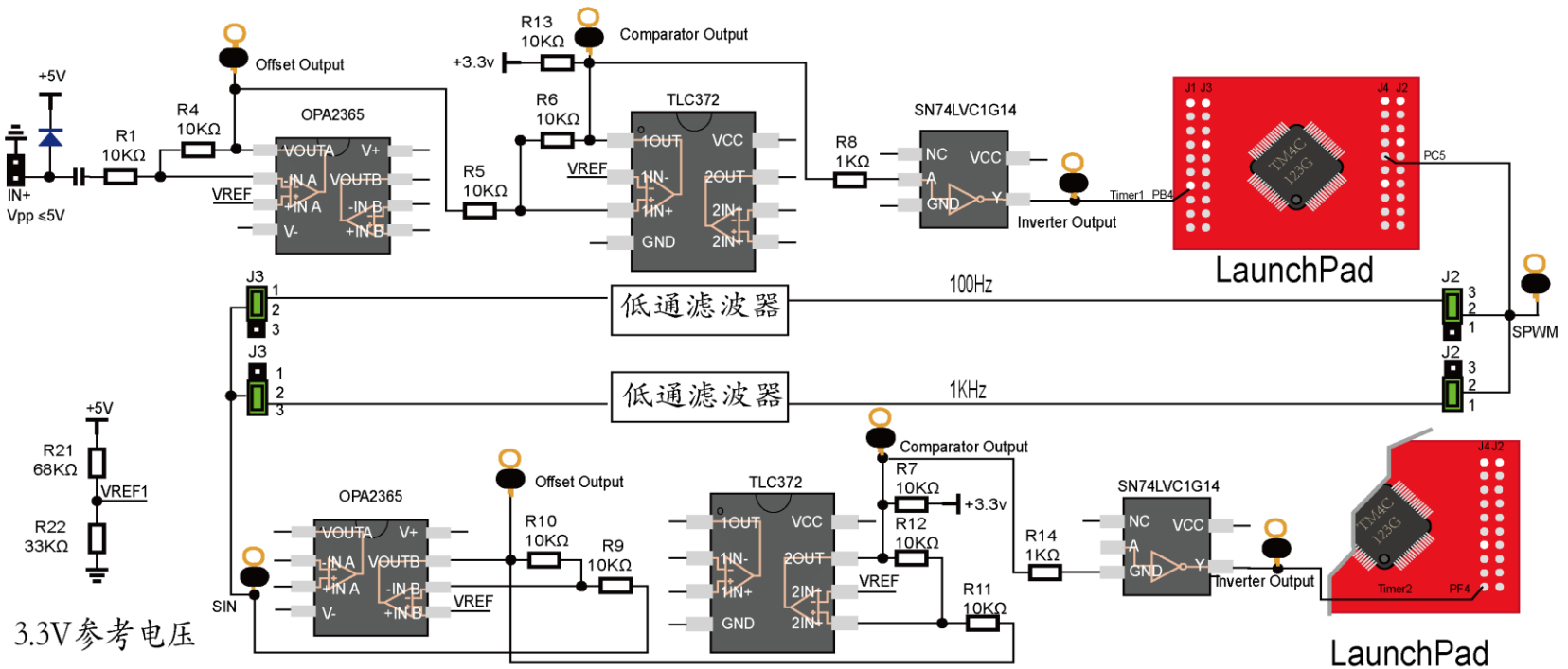
## 4、频率计算方法：

```
// 统计频率信号
void Int_Timer1A_Handler(void) {
    //  最后一次Timer的计数值
    uint32_t CapTimer_Fre = 0;

    // 清除中断标志位
    ROM_TimerIntClear (TIMER1_BASE, ROM_TimerIntStatus (TIMER1_BASE,
false));

    ROM_TimerEnable (TIMER1_BASE, TIMER_A);

    Sample_Index = 0;

    // 获取当前Timer1的计数值
    CapTimer_Fre = TimerValueGet(TIMER1_BASE, TIMER_B);

    // 当两次中断之间的时间差超过了一个Timer1B的计数周期时，需要补加一个Timer1B的计数
值再相减
    if (Fre_TimerOutCount >= 1)
        fre_temp_tick = CapTimer_Fre_Ori
              + ((Fre_TimerOutCount - 1) << 16 )//* TIMER_TOTAL_COUNTNUM)
              + (TIMER_TOTAL_COUNTNUM - CapTimer_Fre);
    else    // 若两次中断的响应在一个周期内，则直接用前一次的计数器值减去后一次的计数器
值
        fre_temp_tick = CapTimer_Fre_Ori - CapTimer_Fre;

    // 统计多个时间差值，统计取平均值
    fre[Fre_Frequece_Index++] = (TIVA_MAIN_FREQUENCY /
fre_temp_tick);//Frequency_detect(fre_temp_tick, Fre_TimerOutCount);
//将计算所得频率放入频率数组中

    if (Fre_Frequece_Index >= FREQUENCY_AVARAGE_NUM)
```

```
    {
        //取平均处理
        Fre_Cur_Frequency = average_float(fre, FREQUENCY_AVARAGE_NUM,
FREQUENCY_CUT_NUM);

        //计算不同频率对应步进值
        Phase_step_N = Fre_Cur_Frequency * SIN_TABLE_N / SAMPLE_FREQUENCY;

        //统计值存放队列清空
        Fre_Frequece_Index = 0;
    }
    //中断处理结束，将当前计数值转化为上次中断计数值用于下次计算差值
    CapTimer_Fre_Ori = CapTimer_Fre;

    Fre_TimerOutCount = 0;
}
```

5、在 Timer1B 的中断响应中，统计出输入信号两次上升沿捕获中断之间 Timer1B 计数器的计数溢出次数：

```
//统计 Timer1B计数溢出的次数
void Int_Timer1B_Handler(void) {
    // 清除中断标志位
    ROM_TimerIntClear (TIMER1_BASE, ROM_TimerIntStatus (TIMER1_BASE,
true));
    ROM_TimerEnable (TIMER1_BASE, TIMER_B);
    // 统计 Timer1B计数溢出的次数
    Fre_TimerOutCount++;
}
```


+5V
IN+
Vpp ≤5V
R1
10KΩ
R4
10KΩ
Offset Output
OPA2365
VOUTA
V+
-IN A
VOUTB
+IN A
-IN B
V-
+IN B
VREF
R5
10KΩ
R13
10KΩ
+3.3v
Comparator Output
R6
10KΩ
TLC372
1OUT
VCC
1IN-
2OUT
1IN+
2IN-
GND
2IN+
VREF
R8
1KΩ
SN74LVC1G14
NC
VCC
A
GND
Y
Inverter Output
Timer1 PB4
J1 J3
J4 J2
PC5
LaunchPad
J3
1
2
3
低通滤波器
100Hz
J2
SPWM
1KHz
+5V
R21
68KΩ
VREF1
R22
33KΩ
SIN
3.3V参考电压
R10
10KΩ
R9
10KΩ
R7
10KΩ
R12
10KΩ
R11
10KΩ
R14
1KΩ
Timer2
PF4
J4 J2

# 频率相位跟踪实验

1、 理解原理图以后编写 Launchpad 代码，代码可参考网上资源。然后烧写代码。

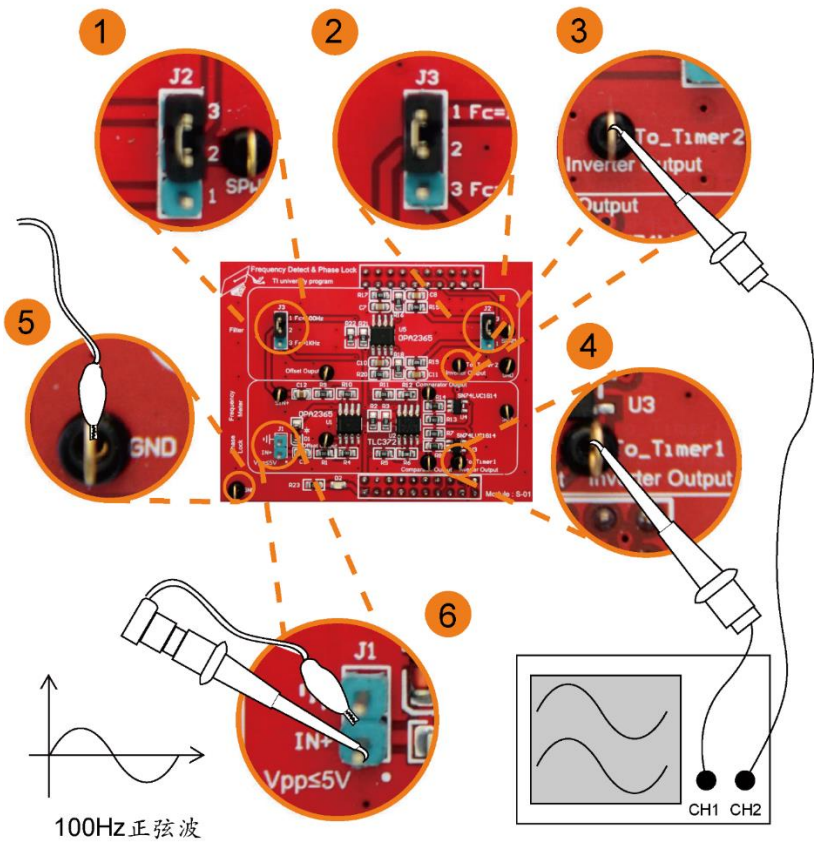
2、 在母板上 TIVA、液晶、MDAC 模块连接完成，准备实验。

3、 跳帽的连接：如图 1 和图 2 。在频率相位模块上用跳帽将 J2 的 3、2 和 J3 的 1、2 连接。

4、仪器连接：示波器两个表笔分别连接到图 3 所示的测试点。和图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

5、信号输入：将信号发生器的表笔连接到图 6 J1 的 IN+ 5 输入频率 100Hz 的正弦波，信号发生器同样要注意接地如图 。

6、打开 TIVA 开关，可以观察到 LED 点亮，在液晶上能看到信号发生器输入的正弦波的频率。同时在示波器上能看到两路信号。一路是 TIVA 经滤波迟滞比较以后的方波。另一路是信号发生器，经过迟滞比较以后的正弦波。观察两路信号比较异同，分析其产生的原理。



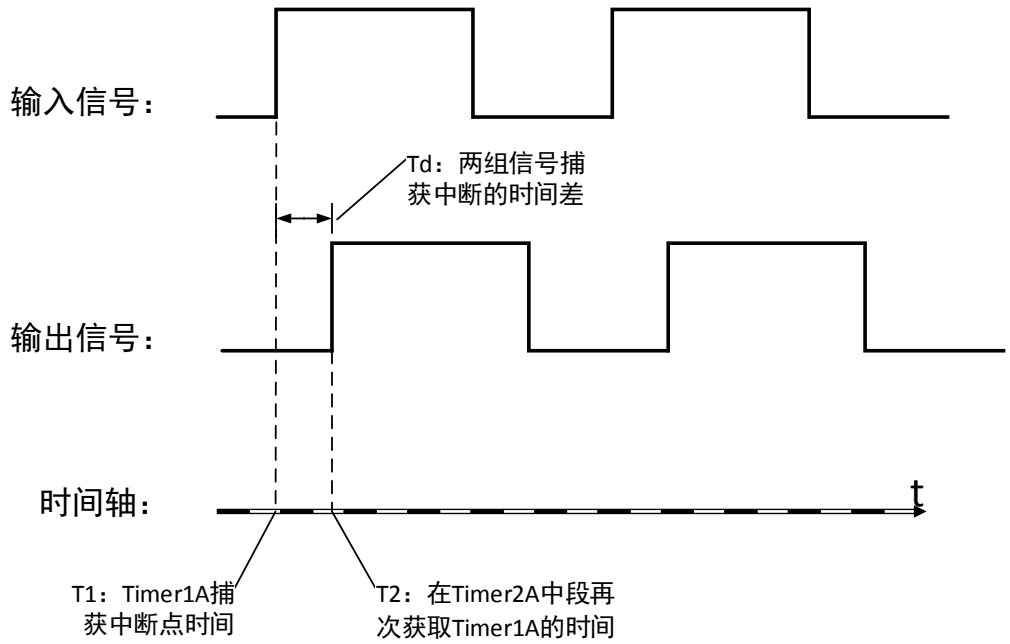

**注意**

连接仪表及跳线时断开电源。另外信号发生器的输入信号改为1KHz的时候，只需要在步骤2用跳帽将J2的2、1和J3的2、3连接即可。

## 软件流程图及关键代码分析

本实验和 SPWM 波的生成与正弦波发生实验一致，所以软件流程也是相同的。

定时器 Timer2 采用和 Timer1A 同样的捕获模式设置采样输出信号的中断，用于测量输入输出信号相位差。输入信号的中断还是在 Timer1A 中响应，输出信号的中断在 Timer2A 上升沿触发时可以获取 Timer1A 定时器时间。通过以上一步同样的方式可以得到两个信号的相位差值。如下图所示：




相位跟踪示意图

# 关键代码分析

## 1、在 Timer2 中计算输入输出两路信号的相位差值：

```
// 统计相位跟踪差值
void Int_Timer2A_Handler(void) {
    // 清除中断标志位
    ROM_TimerIntClear (TIMER2_BASE, ROM_TimerIntStatus
(TIMER2_BASE, false));

    // 因为减计数会自动停止，所以需要重新启用计数模块
    ROM_TimerEnable (TIMER2_BASE, TIMER_A);

    //获取中断响应时的Timer1A计数值
    uint32_t CapTimer1A = ROM_TimerValueGet
(TIMER1_BASE, TIMER_B);

    // 计算两路信号计数差值
    if (Fre_TimerOutCount >= 1)
        Phase_tick = CapTimer_Fre_Ori
              + ((Fre_TimerOutCount - 1) << 16) // *
0xFFFF
              + (TIMER_TOTAL_COUNTNUM - CapTimer1A);
    else
        Phase_tick = CapTimer_Fre_Ori - CapTimer1A;

    //通过PID算法计算相位差，将相位差放入相位差数组中
    pha[Pha_Phase_Index++] =
Delay_time_calculate(Phase_tick, Fre_Cur_Frequency);
    // 取相位差平均值
    if (Pha_Phase_Index >= PHASE_AVARAGE_NUM)
    {
        Delay_phase = average_float(pha,
PHASE_AVARAGE_NUM, PHASE_CUT_NUM);// average_Phase();
        Pha_Phase_Index = 0;
    }
```

```
}
```

## 2、相位差的计算：

相位差在计算过程中会使用到 PID 算法调节。PID 算法，按偏差的比例（P）、积分（I）和微分（D）进行控制的 PID 控制器（亦称 PID 调节器）是应用最为广泛的一种自动控制器。它具有原理简单，易于实现，适用面广，控制参数相互独立，参数的选定比较简单等优点；而且在理论上可以证明，对于过程控制的典型对象——“一阶滞后+纯滞后”与“二阶滞后+纯滞后”的控制对象，PID 控制器是一种最优控制。PID 调节规律是连续系统动态品质校正的一种有效方法，它的参数整定方式简便，结构改变灵活（PI、PD、…）。在两路信号的相位跟踪同步过程控制中通过 PID 算法调节输出信号的相位值不断的趋近与输入信号的相位达到跟踪的效果。

```
/*******************************************************
**************
 *名称：Delay_time_cal(void)
 *功能：计算相位差
 *入口参数：无
 *出口参数：temp_delay_phase

*******************************************************
************/
long int Delay_DETA = 0;
float kp = 0.5, ki = 0.05, kd = 0.01, ui0 = 0.0, ui =
0, u = 0;
float e = 0, e1 = 0;
float Delay_time_calculate(int Phase_tick, float
frequency)            //计算相位差
{
    volatile float Cal_delay_time = 0.0;
    volatile float temp_delay_phase = 0.0;

    Cal_delay_time = IQ_div_f_i(Phase_tick ,
TIVA_MAIN_FREQUENCY) ;
```

```
    if (Cal_delay_time > 0.5 / frequency)
        Cal_delay_time -= 1.0 / frequency;
    if (Cal_delay_time < -0.5 / frequency)
        Cal_delay_time += 1.0 / frequency;

    temp_delay_phase = Cal_delay_time * frequency *
SIN_TABLE_N;

    //位置式PID调节器
    e = temp_delay_phase;
    ui = ui0 + ki * e;
    u = kp * e + kd * (e - e1) + ui;
    ui0 = ui;
    e1 = e;
    temp_delay_phase = u;

    return temp_delay_phase;
}
```