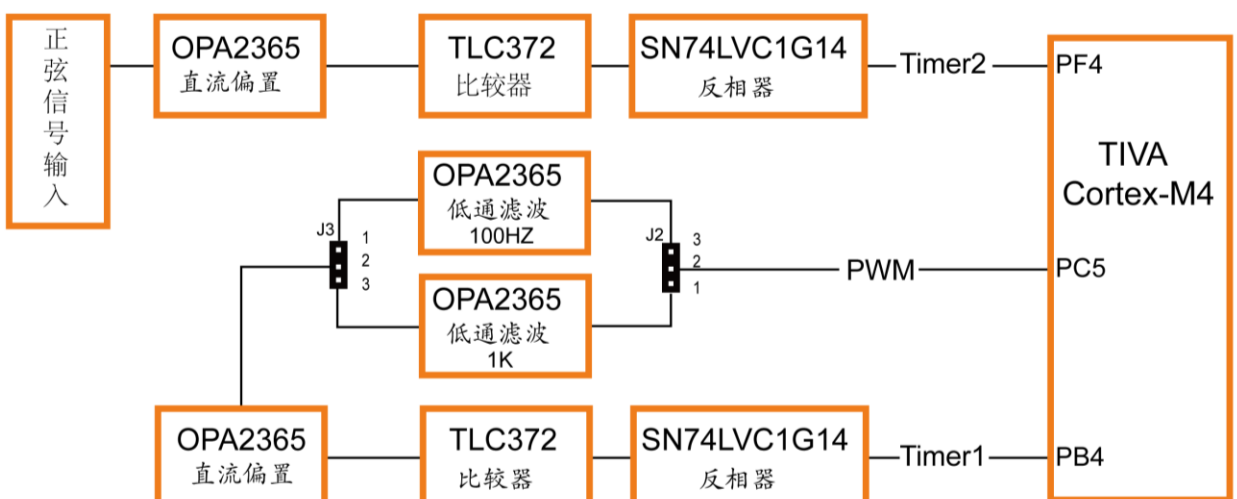
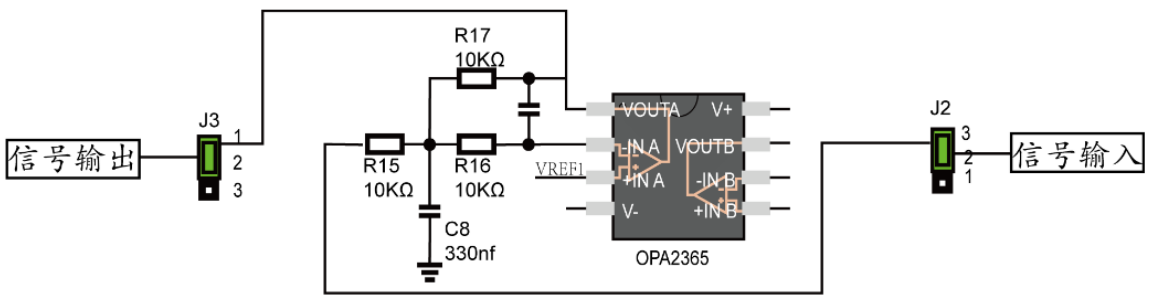
# 第十章、频率与相位跟踪模块

# 频率相位跟踪模块介绍

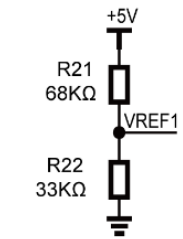
## 实验简介

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。

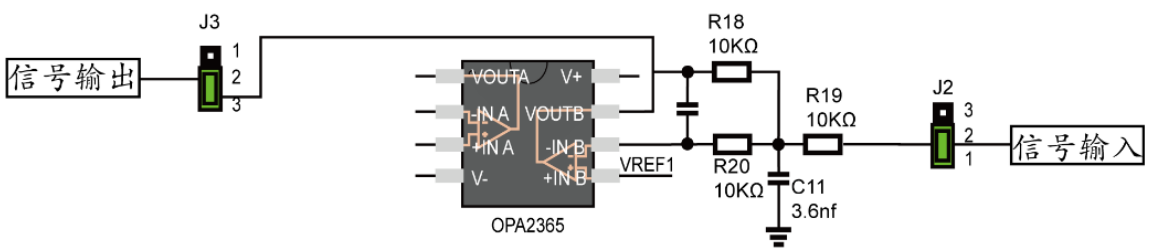





截止频率为100Hz的低通滤波器




3.3V参考电压




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。

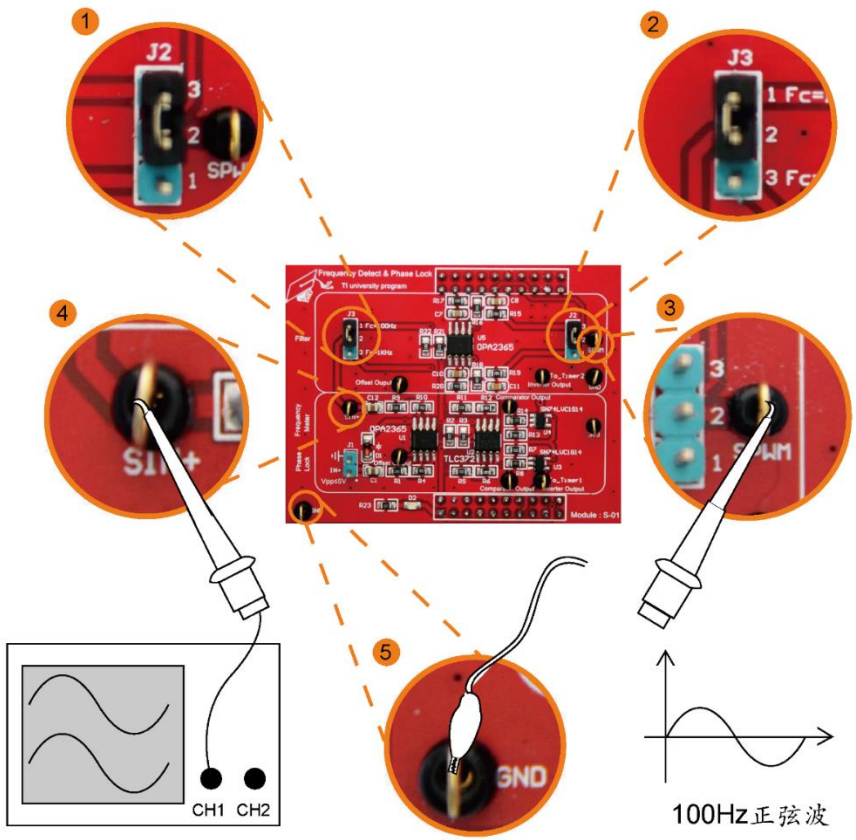
4、仪器连接：示波器表笔连接到图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

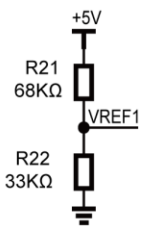
5、信号输入：将信号发生器的表笔连接到图 3 SPWM处，分别输入频率f<100Hz，100Hz<f<1KHz，f>1KHz正弦波，信号发生器同样要注意接地如图 5 。

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。

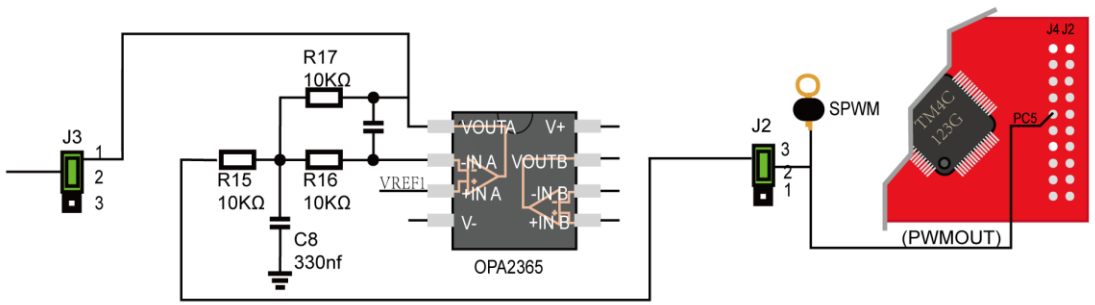
**注意**

连接仪表及跳线时断开电源。



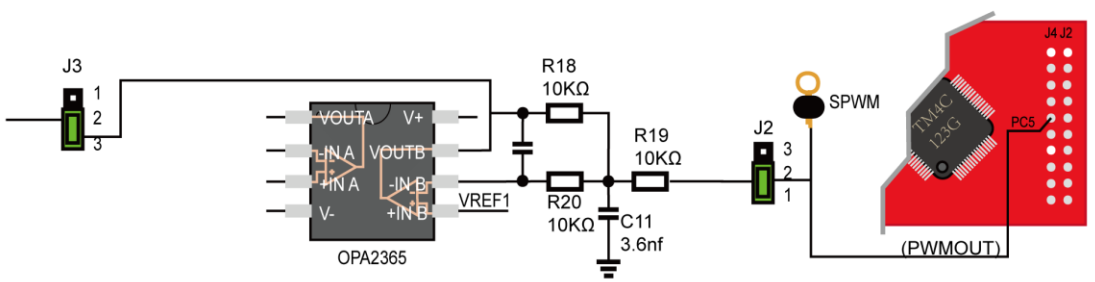



3.3V参考电压




频率为100Hz的正弦波

LlaunchPad




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J3的1、2和J2的2、3连接。

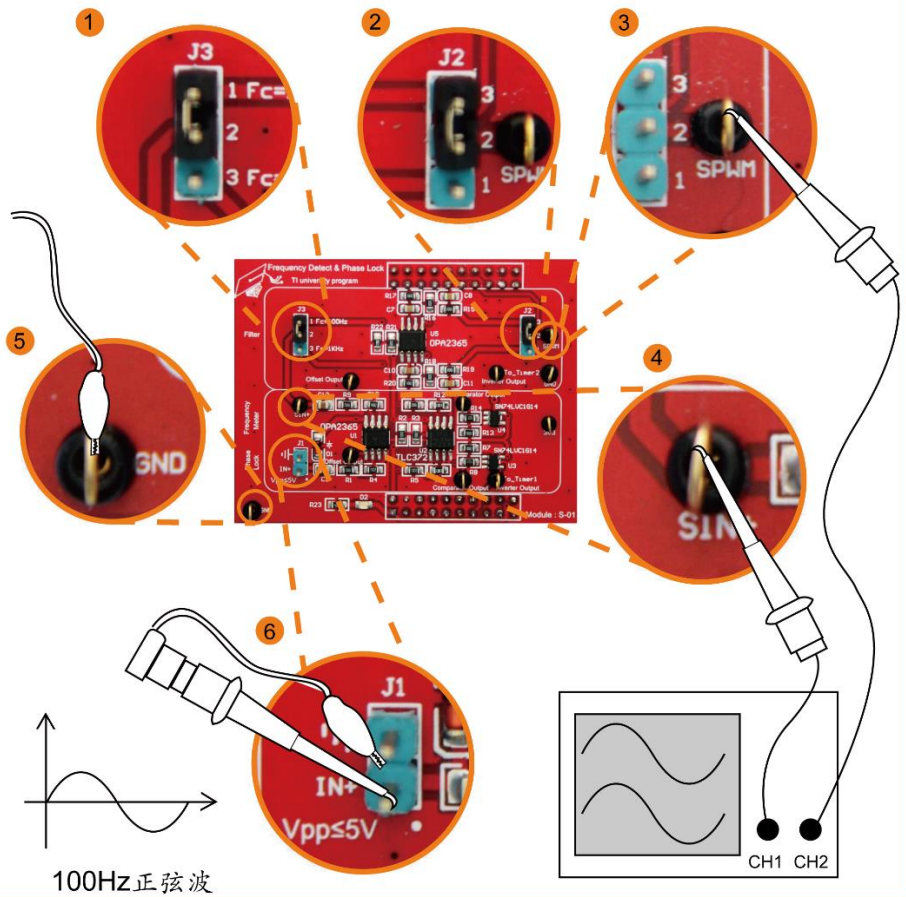
4、仪器连接：示波器两个表笔分别连接到图 3 所示的测试点。和图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

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 。

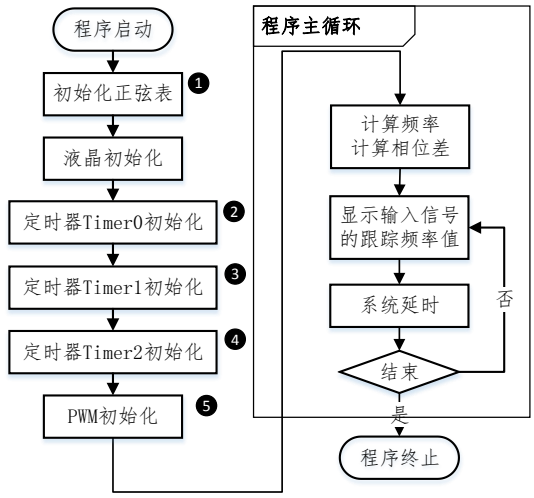
6、打开TIVA开关，可以观察到LED点亮，在液晶上能看到信号发生器输入的正弦波的频率。同时在示波器上能看到两路信号。一路是TIVA输出的占空比变化的方波。另一路是经过滤波以后的正弦波。

**注意**

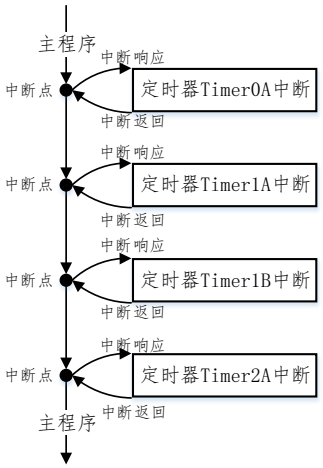
连接仪表及跳线时断开电源。另外信号发生器的输入信号改为1KHz的时候，只需要在步骤2用跳帽将J2的2、1和J3的2、3连接即可。



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

1. 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。
2. 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。
3. 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。
4. Timer2A的TimerA开启捕获模式来计算信号相位偏移。
5. PWM初始负责生成一个10KHz的PWM载波信号。通过响应Timer0A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。

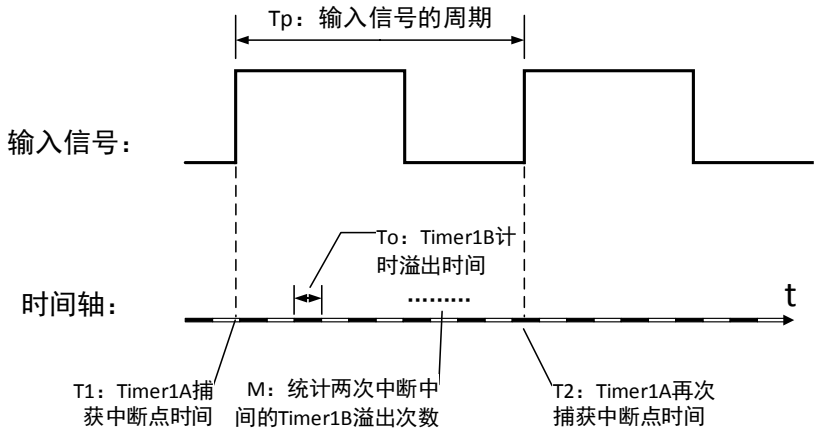
软件流程图如上图所示，频率相位的程序相对而言较为复杂。

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);
}
```

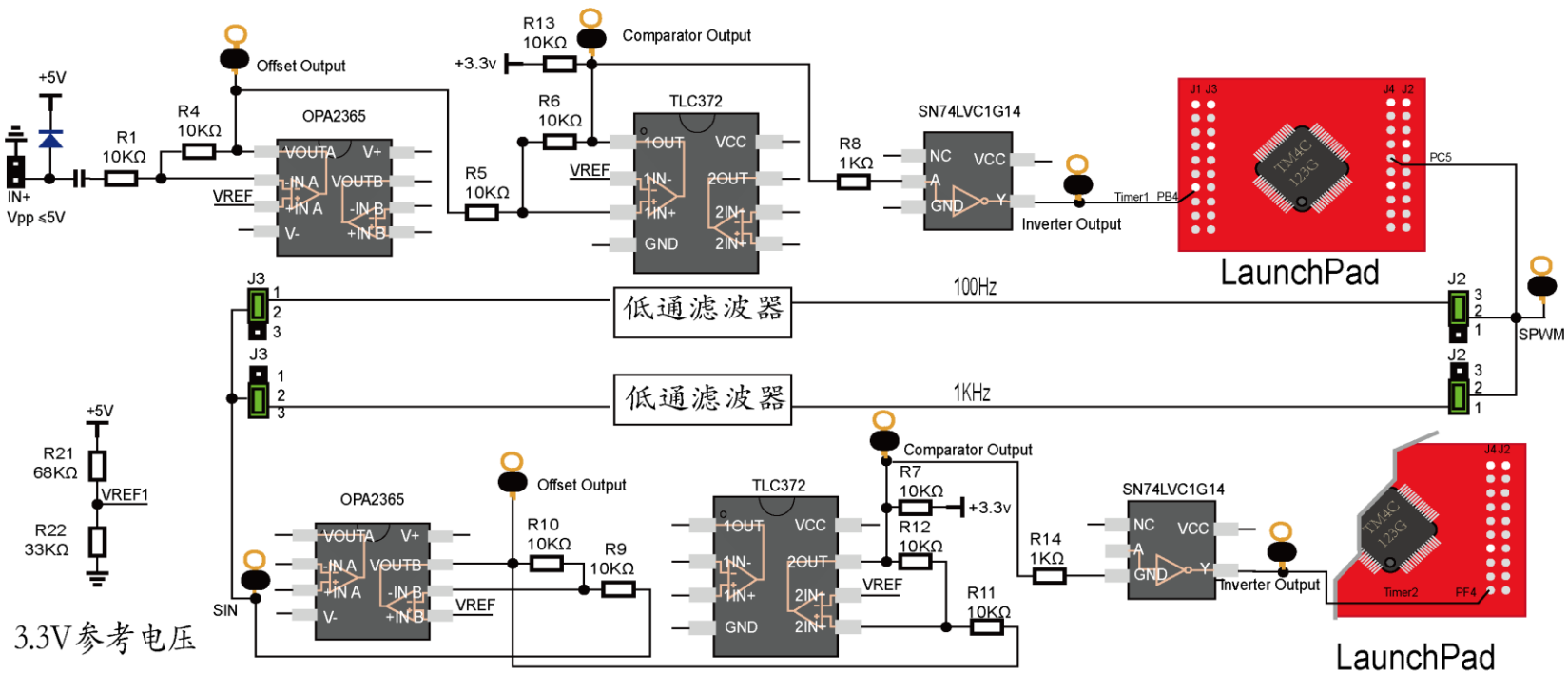
4、频率计算方法：

```
// 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
VREF
R5
10KΩ
R13
10KΩ
+3.3v
Comparator Output
R6
10KΩ
TLC372
R8
1KΩ
SN74LVC1G14
Inverter Output
Timer1 PB4
LaunchPad
PC5
J3
低通滤波器
100Hz
J2
SPWM
1KHz
R21
68KΩ
VREF1
R22
33KΩ
3.3V参考电压
SIN
R10
10KΩ
R9
10KΩ
R7
10KΩ
R12
10KΩ
R11
10KΩ
R14
1KΩ
Timer2 PF4

# 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J3的1、2和J2的2、3连接。

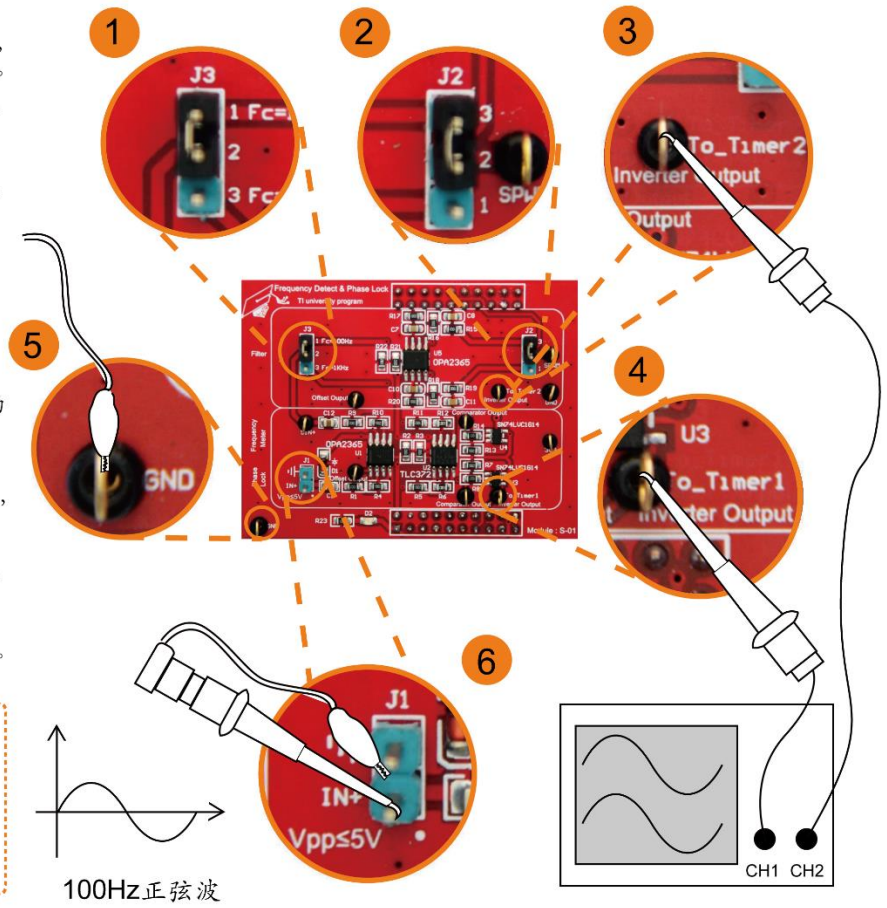
4、仪器连接：示波器两个表笔分别连接到图 3 所示的测试点。和图 4 所示的测试点。注意示波器不要忘了图 5 的接地。

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方波。观察两路信号比较异同，分析其产生的原理。

**注意**

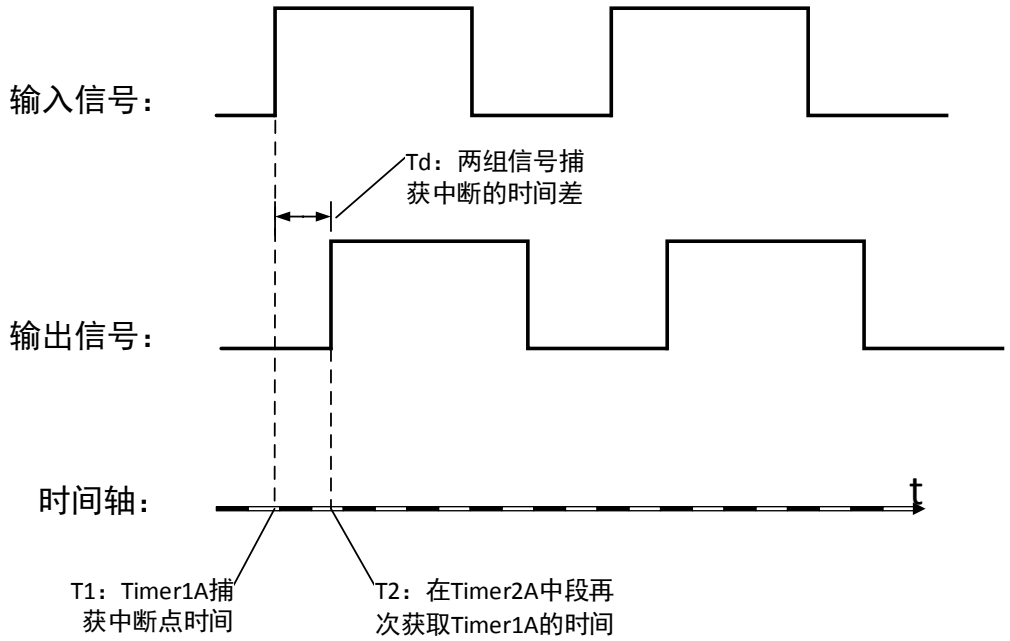
连接仪表及跳线时断开电源。另外信号发生器的输入信号改为1KHz的时候，只需要在步骤2用跳帽将J2的2、1和J3的2、3连接即可。



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。

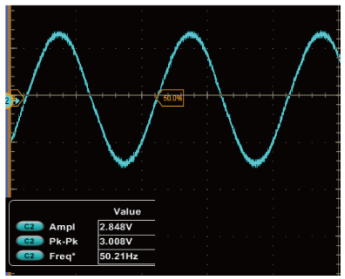
```
/*****************************************************************
 *名称：Delay_time_cal(void)
 *功能：计算相位差
 *入口参数：无
 *出口参数：temp_delay_phase
```

```
 ***********************************************************************/
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;
}
```

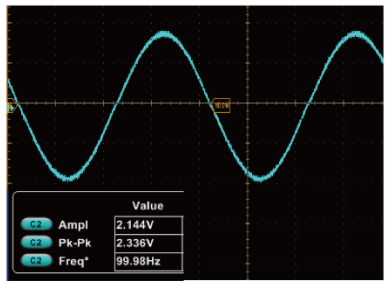
## 低通滤波测试图

如图 1 为截止频率为100HZ的低通滤波器在频率为50HZ下的波形，能观察到信号幅值没有衰减。图 2 是在频率为100HZ下的波形，能观察到信号幅值大概衰减到幅值的0.7倍。

1




2




## 相位跟踪测试图