中山大學

SUN YAT-SEN UNIVERSITY

# 实 验 报 告

课程名称：电机与拖动技术

姓　　名：方桂安

学　　号：**20354027**

专业班级：**2020 级智能科学与技术**

任课教师：冯国栋

2022 年 12 月 8 日

# 实验一　DSP 基础外设实验

## LED 灯控制及寄存器配置实验

### 一、　实验目的

（1）学会使用 GPIO 控制 LED 灯状态；

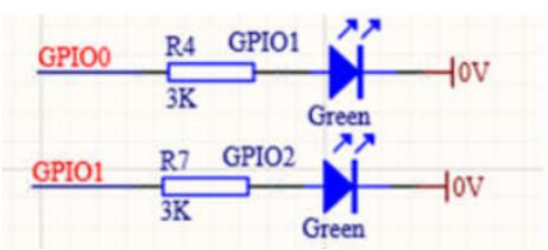
（2）学习配置寄存器并点亮 LED 灯的方法；

（3）熟悉 CCS 操作，学会使用工程，学习编译、下载和运行程序。

### 二、　实验原理

（1）GPIO 是通用输入/输出接口，在 TMS320F28335 处理器上有 88 根 GPIO 管脚，分为 A 组（32 根）、B 组（32 根）及 C 组（24 根）。芯片上的管脚资源是有限的，一 部分 GPIO 管脚与片内外设的外部管脚共用，即：一个 GPIO 口可以用作数字量电平的输 入/输出口，同时可能也是某个片内外设的外部管脚的一部分。所以，希望使用 GPIO 口 I/O 功能时，需要把相应的复用管脚切换到 GPIO 功能。下面将详细讲解如何找到相关的寄存器参数。

这里以底板 LED 为例进行说明。首先我们需要确定控制该 LED 灯的 GPIO 管脚号，在光盘资料中“实验箱底板硬件图\Super28335_BOX 控制板\”路径下打开原理图文件，找到需要点亮的 LED，如下图所示：



从上图中可以看出 GPIO1-GPIO2 对应底板的接口名字分别是“GPIO0-GPIO1”。连接到核心板上对应的接口名字是“GPIO6-GPIO7”

第 1 步：配置 GPIO 功能选择寄存器

对于 F28335 输入/输出引脚的操作，都是通过对寄存器的设置来实现的。例如，选择某个引脚功能是做外设引脚还是做通用数字 I/O 口，当引脚作为通用数字 I/O 口时，是做输入还是做输出，如何使其输出高电平或者低电平，如何使其引脚电平翻转，如何知道引脚上的电平是高或者是低，这些都是通过对 GPIO 寄存器的操作来实现的，每个 I/O 引脚都可以通过寄存器相应的位得到设置。在 F28335 中，功能选择控制寄存器是 GPxMUXn。

每个 GPIO 引脚都有多个功能，但是在同一时刻，通过功能选择控制寄存器只能选择一种功能。

若 GPxMUX.bit=0，管脚配置为数字 I/O 功能；

若 GPxMUX.bit=1，管脚配置为外设功能。

从光盘资料中可以查到 GPIO6-GPIO7 对应的 GPIO 功能选择配置寄存器为 GPAMUX1

第 2 步：配置 GPIO 方向控制寄存器

每个 I/O 口都对应一个方向控制寄存器，用来配置 GPIO 口的输入输出方向。复位 时，所有的 GPIO 配置为输入。在 TMS320F28335 处理器中，方向控制寄存器的名字是 “GPxDIR”。若 GPxDIR.bit=0，管脚配置为输入；若 GPxDIR.bit=1，管脚配置为输出。

从光盘资料中可以查到 GPIO6-GPIO7 对应的 GPIO 方向控制寄存器为 GPADIR。

第 3 步:配置 GPIO 数据寄存器

数据寄存器主要由数据寄存器 GPxDAT、置位寄存器 GPxSET、清除寄存器 GPxCLEAR 和状态翻转寄存器 GPxTOGGLE 等组成。 如果想要让通用数字 I/O 引脚输出高电平，又不影响其他引脚，则只需要对 GPxSET 寄存器的相应位写 1，对其写 0 无效。如果想要让通用数字 I/O 引脚输出低电平，又不影响其他引脚，则只需要对 GPxCLEAR 寄存器的相应位写 1，对其写 0 无效。

从光盘资料中可以查到 GPIO6-GPIO7 对应的 GPIO 置位寄存器为 GPASET，清除寄存器为 GPACLEAR。

学习了以上过程，可以修改相关寄存器的值来点亮 LED 灯。

（2）本实验的功能是底板的 2 个 LED 灯：GPIO1，GPIO2 和核心板的 1 个 LED：RUN 都做闪烁点亮。其中 RUN 对应的管脚为 GPIO23。由于管脚是有复用配置的，因此需要将管脚配置成对应的 GPIO 管脚和 output 输出模式，双击打开 led.Run 工程的 DSP2833x_Gpio.c 文件，可以实现管脚复用的相关配置，如图所示：

```
void InitGpio(void)
{
    asm(" EALLOW");                         // Enable EALLOW protected register access

//--- Group A pins
    GpioCtrlRegs.GPACTRL.all  = 0x00000000;     // QUALPRD =1xSCLKOUT for  group A GPIO0-31
    GpioCtrlRegs.GPAQSEL1.all = 0x00000000;     // No qualification for all group A GPIO 0-15
    GpioCtrlRegs.GPAQSEL2.all = 0x00000000;     // No qualification for  group A GPIO 16-31
    GpioCtrlRegs.GPADIR.all   = 0xFFFFFFFF;     // group A GPIO 0-31 AS OUTPUT
    GpioCtrlRegs.GPAPUD.all   = 0x00000000;     //Pullups enabled GPIO31-20,GPIO0-15 disabled GPIO16-19


  GpioCtrlRegs.GPAMUX1.bit.GPIO6  = 0;        // 0=GPIO   1=EPWM4A     2=EPWMSYNCI  3=EPWMSYNCO  d6由GPIO6控制
  GpioCtrlRegs.GPAMUX1.bit.GPIO7  = 0;        // 0=GPIO   1=EPWM4B     2=MCLKRA     3=ECAP2


  GpioCtrlRegs.GPAMUX2.bit.GPIO23 = 0;        // 0=GPIO  SYSRUN 1=EQEP1I     2=MFSXA      3=SCIRXDB
```

然后双击打开 LED.Run 工程的 main.c 文件，定义 LED 灯的闪烁功能，如图所示：

```
void configtestledON(void) //点亮LED灯
{
   EALLOW;

   GpioDataRegs.GPASET.bit.GPIO6=1; //LED1
   GpioDataRegs.GPASET.bit.GPIO7=1; //LED2
   GpioDataRegs.GPASET.bit.GPIO23=1;//RUN

   EDIS;
}

void configtestledOFF(void)
{
   EALLOW;

   GpioDataRegs.GPACLEAR.bit.GPIO6=1;
   GpioDataRegs.GPACLEAR.bit.GPIO7=1;
   GpioDataRegs.GPACLEAR.bit.GPIO23=1;

   EDIS;
}
```

其中，EALLOW：仿真读取使能位，执行该汇编函数，去保护，解锁寄存器，完成管脚复用 的配置。EDIS：添加寄存器保护机制，无法对系统寄存器进行修改配置。

再加入以下的延时程序之后便可以实现 LED 灯闪烁，如下图所示：

```
void delay (Uint16 t)
{ Uint16 i;
  while(t--)
  {
     for(i=0;i<125;i++)
     asm(" RPT #3 || NOP");
  }
```

## 三、 实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。

（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

## 四、 实验过程

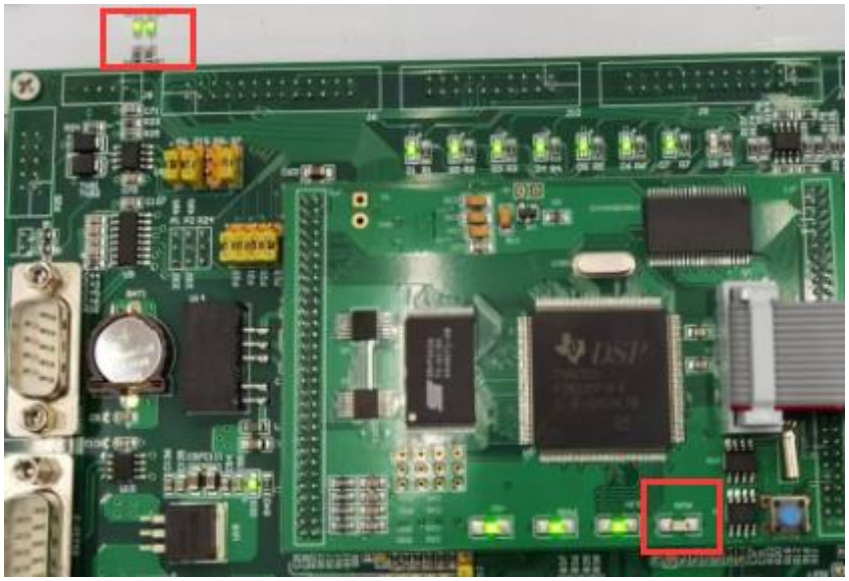
首先开启设备连接 CCS，打开 led.run 工程并测试是否连接成功；

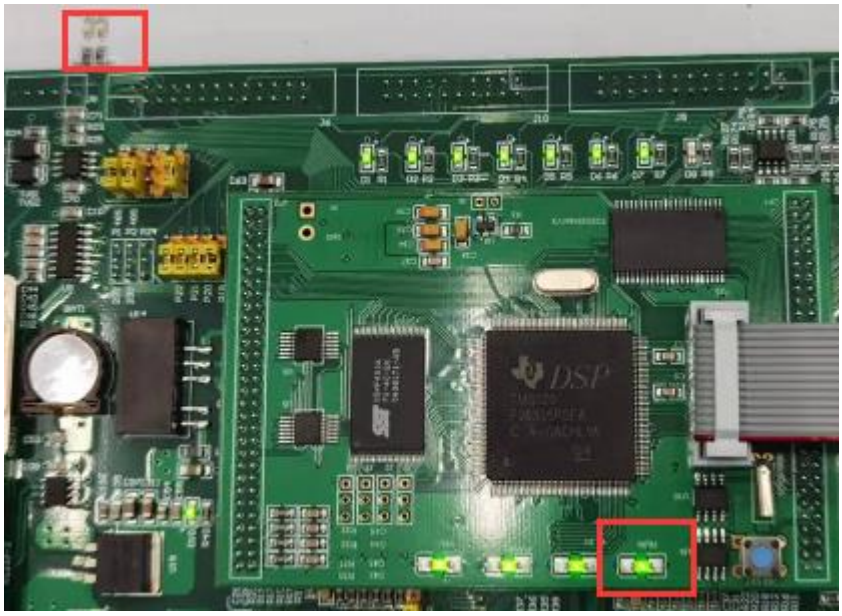
编译工程生成 LED.out 可执行程序，下载程序并运行。

图中红框所示的 led 闪烁，运行成功。

```
33
34    while(1)
35    {
36
37      configtestledOFF(); //熄灭LED灯，I/O输出低电平
38      DELAY_US(5000000);    //延时
39
40      configtestledON(); //点亮LED灯，I/O输出高电平
41      DELAY_US(500000);  //延时
42
43    }
44 }
45
46
47 void configtestledON(void) //点亮LED灯
48 {
49    EALLOW;
50
51    GpioDataRegs.GPASET.bit.GPIO6=1; //LED1
52    delay(10000);
53    GpioDataRegs.GPASET.bit.GPIO7=1; //LED2
54    GpioDataRegs.GPASET.bit.GPIO23=1;//RUN
55
56    EDIS;
57 }
```

在LED1与LED2添加延时函数，在点亮LED函数前的延时函数增加10倍时间：





成功实现：

（1）修改程序使得 LED 灯交替闪烁

（2）修改程序改变 LED 灯闪烁时间

## 定时器/计数器控制实验

### 五、 实验目的

（1）实现定时器的功能；

（2）熟悉 F28335 定时器的基本结构；

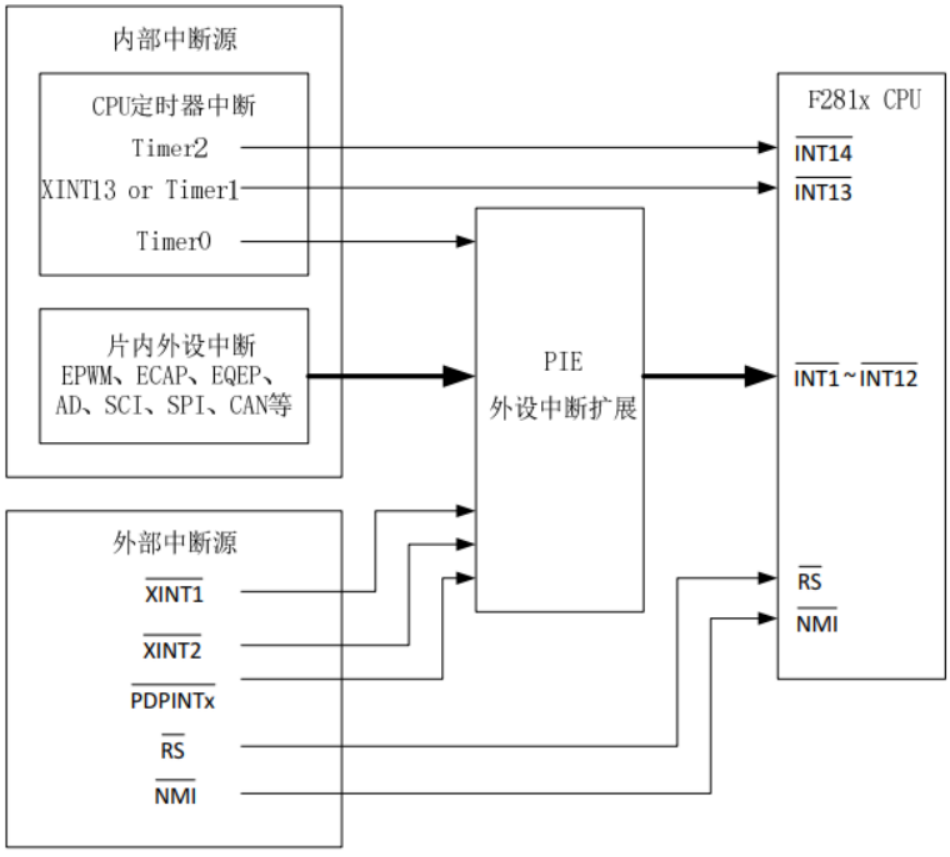
（3）掌握定时器的控制方法；

（4）掌握使用定时器中断方式控制程序的流程。

## 六、 实验原理

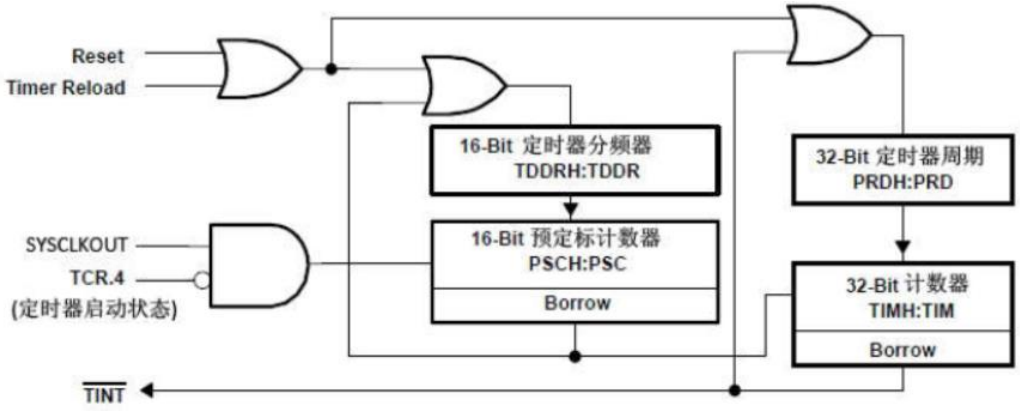
一般定时计数器有两个功能，一是定时：时钟源一般来源于 DSP 内部，当然也可以 选择来自于外部。二是计数：我们可以利用它的功能来计算外部脉冲在一段时间内到来 的次数，所以叫做计数器（记录外部脉冲的次数）。

F28335 的 CPU TIMER 定时器，一共有 3 个，分别是：CPU-Timer 0、CPU-Timer 1、CPU-Timer2。其中，CPU-Timer2 是预留给 DSP/BIOS 操作系统的。如果用户没有用到操作系统，CPU-Timer2 是当做普通的 CPU 定 时器使用。这三个定时器的中断信号，分别是 TINT0、TINT1 和 TINT2，分别对应于中断 向量 INT1、INT13 和 INT14。对应示意图如下：



可以看出，TIN0 的中断信号，是有经过 PIE，而另外两个 TINT1 和 TIN2 中断信号是没有经过 PIE 的。

其中，定时器的工作原理也有对应的结构图，如下图所示：




在 SYSCLKOUT 时钟信号的驱动下，预分配计数器（PSCH：PSC）的值大于 0 时，每 出现一个脉冲，预分配计数器的值，减 1。当预分配计数器的值，减到 0 的时候，自动 将（TDDRH：TDDR）中的值，装载到预分配计数器中，同时产生一个脉冲输出。 在这个脉冲的作用下，32 位计数器（TIMH：TIM）中的值，也会减 1，当减到 0 的时 候，自动将 32 位周期寄存器（PRDH：PRD）中的值，装载到 32 位计数器（TIMH： TIM）中，同时产生一个中断信号 TINT。

双击打开 timer0 工程目录下的 TIMER0.c 文件，此实验使用 CPU 定时器 0 的周期中断来控制 LED 灯的闪烁。如下图所示：

```
// Disable CPU interrupts
   DINT; 禁用CPU中断

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
   InitPieCtrl();初始化控制寄存器

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;初始化PIE控制寄存器；禁止所有PIE中断，清除所有标志位
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
   InitPieVectTable(); 使能PIE中断向量表
```

将自定义的中断函数地址赋给 CPU 定时器 0 的中断向量，如下图所示：

```
EALLOW;  // This is needed to write to EALLOW protected registers
PieVectTable.TINT0 = &ISRTimer0;
EDIS;    // This is needed to disable write to EALLOW protected registers
```

本实验只需初始化 Cpu Timers，如下图所示：

```
InitCpuTimers();   // For this example, only initialize the Cpu Timers
```

配置 CPU 定时器 0 的周期为 1，定时器时钟频率为 150MHz 如下图所示：

```
ConfigCpuTimer(&CpuTimer0, 150, 1000000);
StartCpuTimer0();
```

使能 CPU 定时器 0 的 CPU 中断，如下图所示：

```
// Enable CPU int1 which is connected to CPU-Timer 0
   IER |= M_INT1;
```

然后使能 CPU 定时器 0 周期中断对应的 PIE 中断，位于第一组第 7 个，所以置 1，如下图所示：

```
// Enable TINT0 in the PIE: Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;   使能PIE
// Enable global Interrupts and higher priority real-time debug events:
    EINT;   // Enable Global interrupt INTM
    ERTM;   // Enable Global realtime interrupt DBGM

    EALLOW;
    GpioCtrlRegs.GPAMUX2.bit.GPIO23=0;  //run灯控制引脚
    GpioCtrlRegs.GPADIR.bit.GPIO23=1;
    EDIS;
    GpioDataRegs.GPADAT.bit.GPIO23=0; //D1灯的控制引脚
                    RUN灯的初始状态为熄灭
    for(; ;)
    {

    }

}
interrupt void ISRTimer0(void)
{

    GpioDataRegs.GPADAT.bit.GPIO23=~    GpioDataRegs.GPADAT.bit.GPIO23;


   // Acknowledge this interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    CpuTimer0Regs.TCR.bit.TIF=1;     确认此中断能够响应同组其他中断
    CpuTimer0Regs.TCR.bit.TRB=1;

}
```

## 七、 实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。

（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

## 八、 实验过程

首先开启设备连接 CCS，打开 timer.run 工程并测试是否连接成功；

编译工程生成 TIMER.out 可执行程序，下载程序并运行。


图中红框所示的 RUN 灯闪烁，运行成功。

```
// Configure CPU-Timer 0, 1, and 2 to interrupt every second:
// 150MHz CPU Freq, 1 second Period (in uSeconds)

   ConfigCpuTimer(&CpuTimer0, 150, 100000);
   StartCpuTimer0();

// Enable CPU int1 which is connected to CPU-Timer 0
   IER |= M_INT1;

// Enable TINT0 in the PIE: Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
```

修改 CPU 定时器 0 的周期，图中光标所选的参数以 us 为单位，缩小 10 倍，周期变为 0.1s。


肉眼可见 RUN 灯闪烁频率增大，每 0.1 秒一次。

## 模数转换（A/D）测试实验

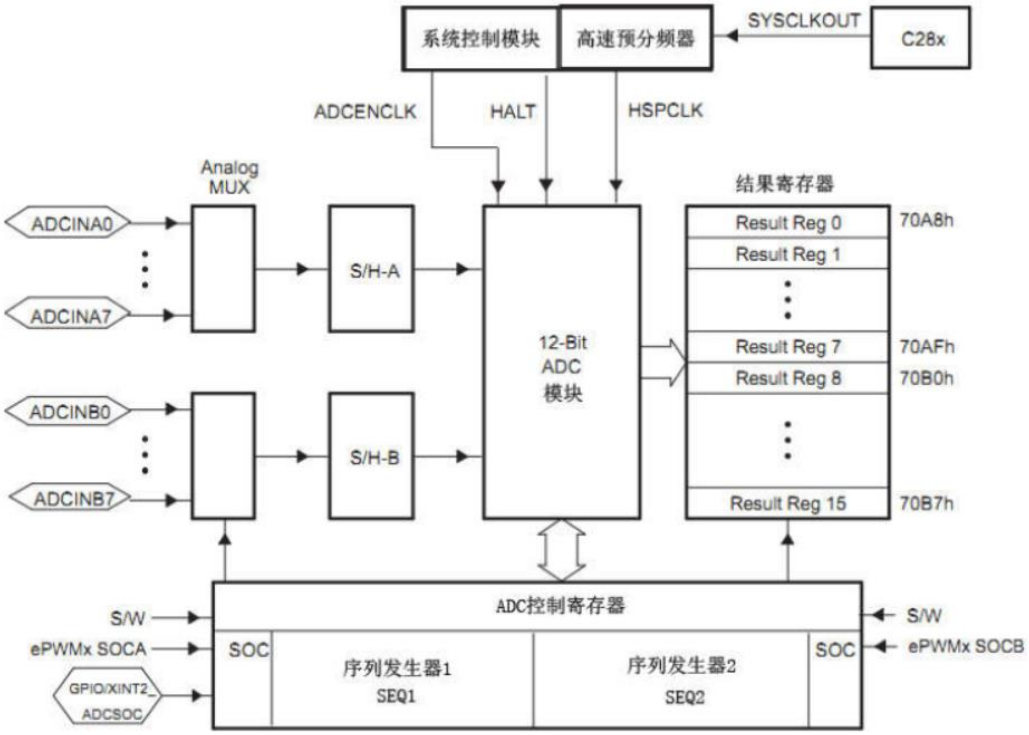
### 九、 实验目的

（1）测试 F28335 的 A/D 转换功能；

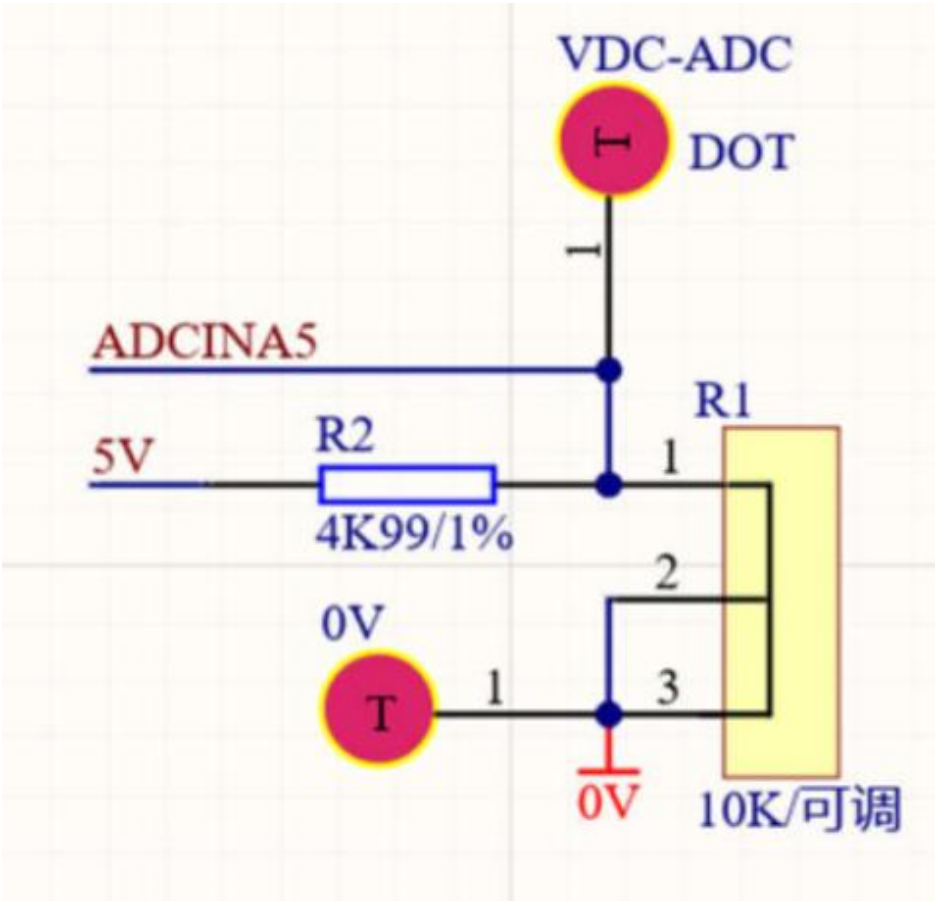
（2）了解 F28335 内部的 A/D 转换的工作原理。

### 十、 实验原理

F28335 内部的 ADC 模块是一个 12 位分辨率的、具有流水线结构的模数转换器，其结构框图如下图所示。从图中可以看到，F28335 的 ADC 模块一共具有 16 个采样通道，分成了两组，一组为 ADCINA0~ADCINA7，另一组为 ADCINB0~ADCINB7。A 组的通道使用采样保持器 A，也就是图中的 S/H-A，B 组的通道使用采样保持器 B，也就是图中的 S/H-B。


系统控制模块
高速预分频器
SYSCLKOUT
C28x
ADCENCLK
HALT
HSPCLK
Analog MUX
结果寄存器
ADCINA0
ADCINA7
S/H-A
12-Bit ADC 模块
Result Reg 0
Result Reg 1
Result Reg 7
Result Reg 8
Result Reg 15
70A8h
70AFh
70B0h
70B7h
ADCINB0
ADCINB7
S/H-B
ADC控制寄存器
S/W
ePWMx SOCA
GPIO/XINT2_ ADCSOC
SOC
序列发生器1 SEQ1
序列发生器2 SEQ2
SOC
S/W
ePWMx SOCB

以下是相关原理图：


VDC-ADC
DOT
ADCINA5
5V
R2
4K99/1%
R1
1
2
3
0V
T
0V
10K/可调

## 十一、 实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。

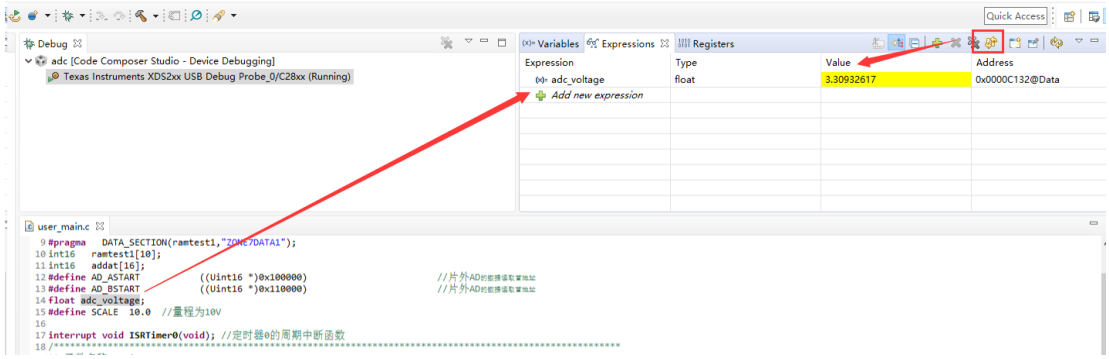
（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

## 十二、 实验过程

首先开启设备连接 CCS，打开 adc.run 工程并测试是否连接成功；

编译工程生成 ADC.out 可执行程序，下载程序并运行。

在主函数 user_main.c 中将 adc_voltage 变量添加至“Expressions”观察窗口，点击刷新按钮，观察电压示数。




由于我们没有万用表，这里使用的是示波器


对比示数，可以看出 CCS 中显示的 3.30V 与示波器中的 3.34V 在误差允许范围内

大小一致。

Variables | Expressions | Registers

| Expression | Type | Value | Address |
|---|---|---|---|
| adc_voltage | float | 1.36474609 | 0x0000C132@Data |
| Add new expression | | | |


用十字螺丝刀旋转可调电阻，重复实验以减少偶然性，结果依旧保持一致。

```
128
129         adc_voltage=addat[5]*SCALE/32768.0;
130         Uint16 i = 0;
131     if (adc_voltage>1){
132         for(i=0;i<10;i++){
133             configtestledON(); //点亮LED灯，I/O输出高电平
134         }
135     }
136     else if (adc_voltage>0&&adc_voltage<1){
137         for(i=0;i<10;i++){
138             configtestledOFF(); //熄灭LED灯，I/O输出低电平
139                 DELAY_US(500000);    //延时
140
141                 configtestledON(); //点亮LED灯，I/O输出高电平
142                 DELAY_US(500000);  //延时
143                }
144     }
145
```

结合实验一修改程序实现，当 A/D 采样电压大于 1V 时完全点亮 LED 灯；在 A/D 采样电压在 0V~1V 之间时 LED 闪烁。
分别调节电阻直至 CCS 显示电压为 0.5V 与 1.5V，可以观察到 LED 按照我们设置的逻辑明暗切换。

## 数模转换（D/A）测试实验

### 十三、 实验目的

（1）测试 F28335 的 D/A 转换功能；

（2）了解 F28335 内部的 D/A 转换的工作原理。

### 十四、 实验原理

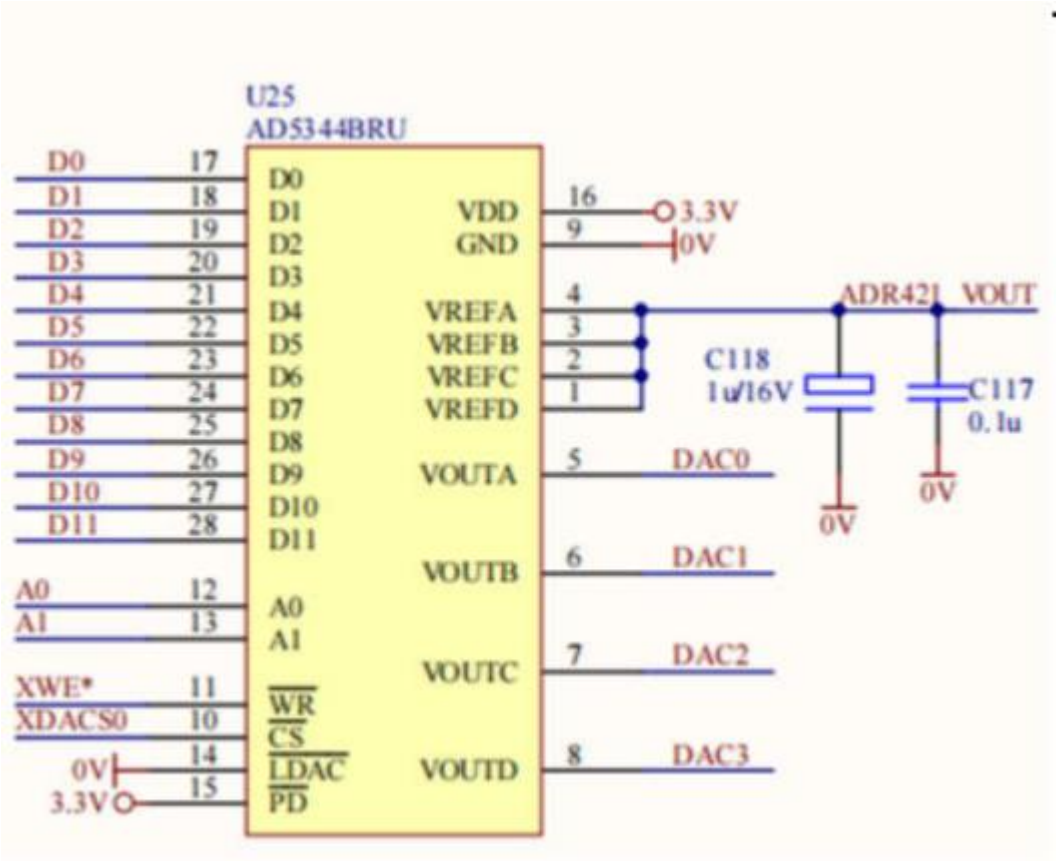
（1）AD5344 模块特性

AD5344 是 12 位的数模转换器，在 2.5V 到 5.5V 的电源条件下工作，只需要 500μA，并具有降电模式，进一步将电流降低到 80nA。AD5344 包含了一个芯片上输出缓冲区，可以驱动输出到两个层轨，AD5344 具有一个并行接口。

（2）数模转换工作原理

D/A 转换器将输入的二进制数字量转换成模拟量，以电压或

电流的形式输出。D/A 转 换器基本上由 4 个部分组成，即权电阻网络、运算放大器、基准电源和模拟开关。它是 把连续的模拟信号转变为离散的数字信号的器件。一般用低通滤波即可以实现。数模转换器工作原理就是数字信号先进行解码，即把数字码转换成与之对应的电平，形成阶梯状信号，然后进行低通滤波。




## 十五、 实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。

（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

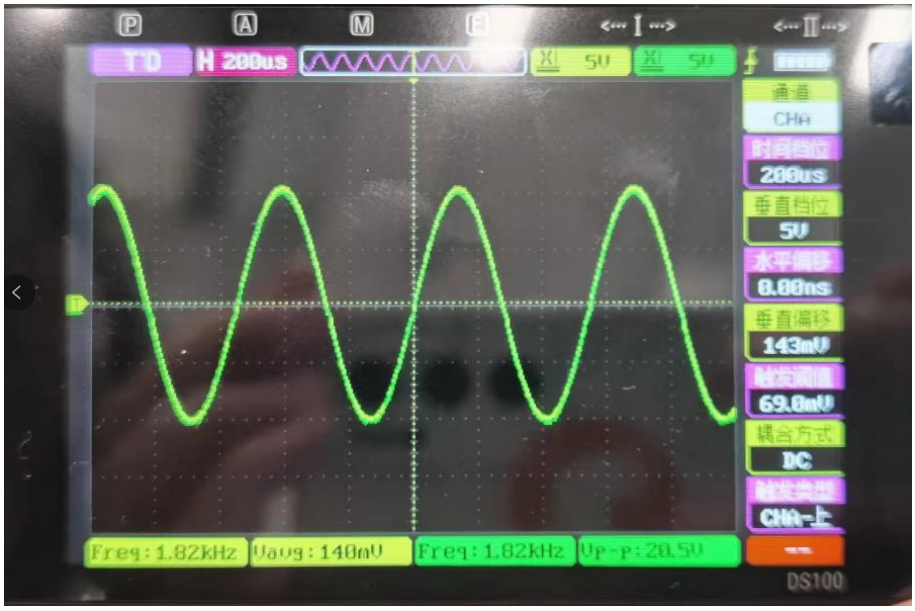
## 十六、 实验过程

首先开启设备连接 CCS，打开 dac.run 工程并测试是否连接成功；

编译工程生成 DAC.out 可执行程序，下载程序并运行。

将示波器的 CHA 通道与底板上 DAC-ADC 接口用 SMA 线

相连接，按键 A 重置示波器，运行程序,观察波形：

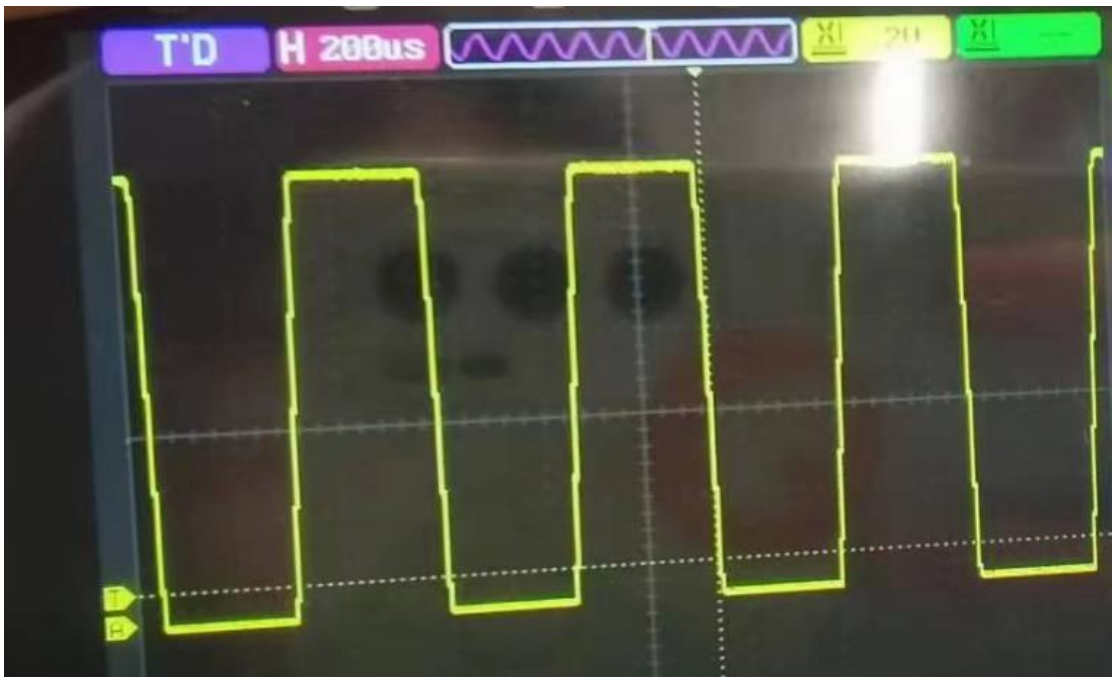



重复上述步骤连接 DAC2,DAC3,DAC4 等接口，都能看到正弦波形：


阅读代码，我们猜测注释掉的这行代码产生的正弦数组就是波形形状的关键。

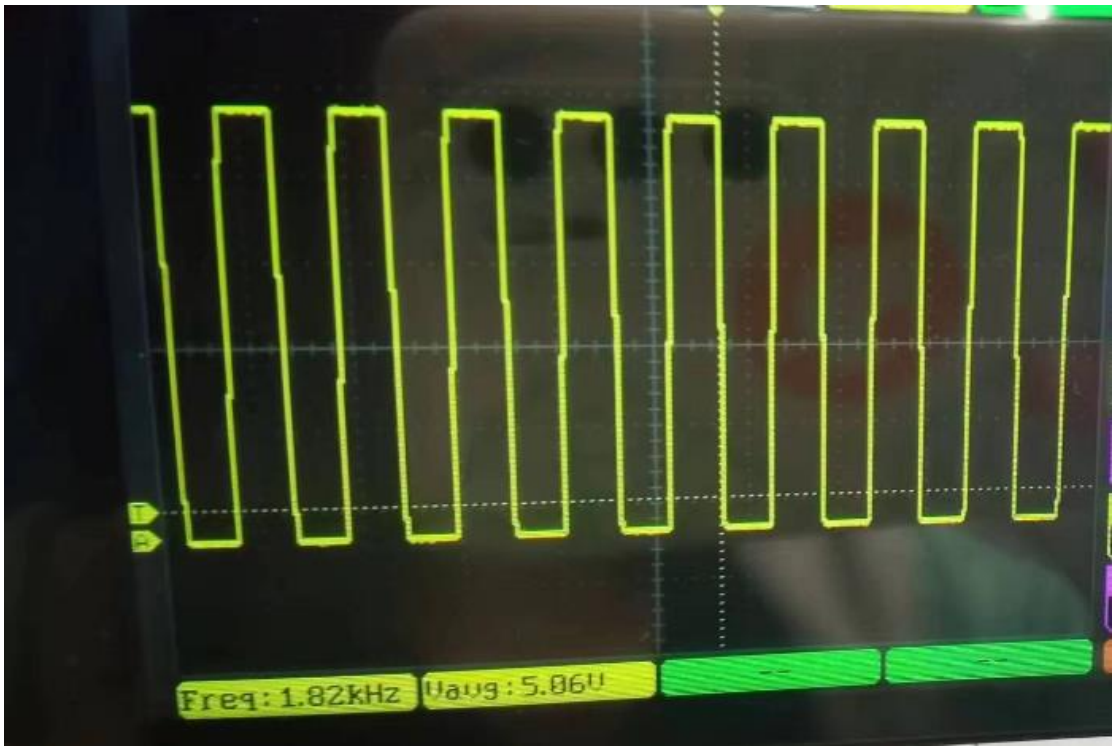
故我们设置了变量 j 作为 flag，变量 c 用来计数，将原本长度 200 的 sincal1 数组赋值为 0,1 两种结果，通过这种方式实现了方波信号。

```
int16 j=0;
int16 c=0;
for(k=0;k<200;k++)
 {
 //sincal1[k]= sin(TWOPAI*k*0.005);
 if (j==0){
     sincal1[k]=1;
     c++;
     if(c==100){
         j=1;
         c=0;
     }
 }
 else{
     sincal1[k]=0;
              c++;
              if(c==100){
                  j=0;
                  c=0;
              }
 }
```




对此，如果想改变赋值，修改 0/1 两种结果即可；想改变频率，修改 c 的判断条

件即可。
效果如下：




三角波的无穷傅里叶级数展开为：

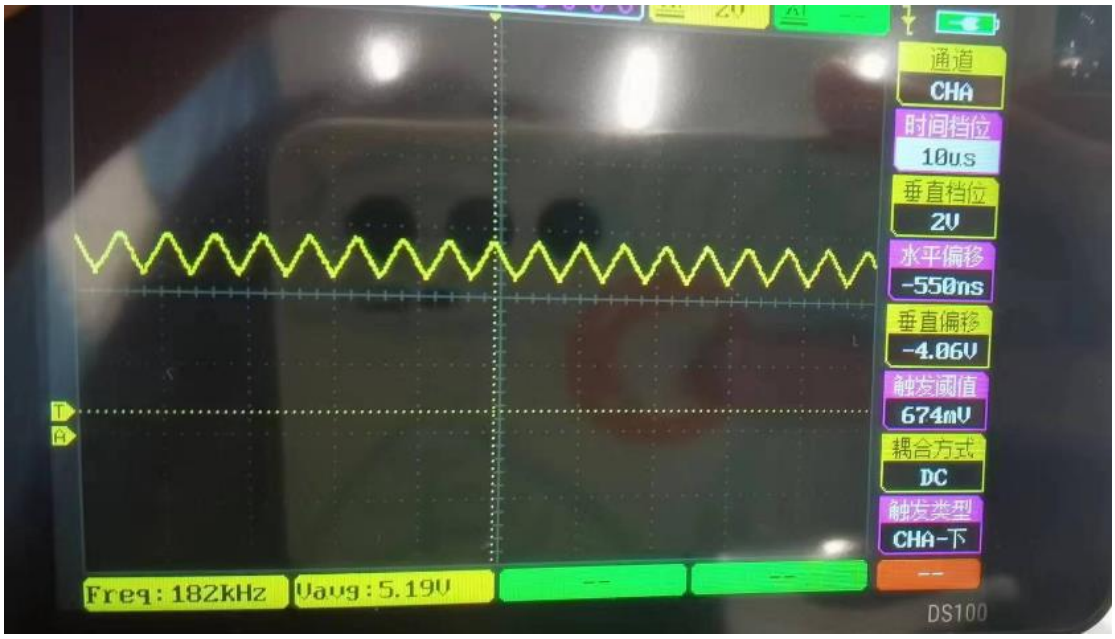
$$x_{\text{triangle}}(t) = \frac{8}{\pi^2}\sum_{k=0}^{\infty}(-1)^k\frac{\sin((2k+1)t)}{(2k+1)^2}$$
$$= \frac{8}{\pi^2}\left(\sin(t) - \frac{1}{9}\sin(3t) + \frac{1}{25}\sin(5t) - \cdots\right)$$

但这样的实现太过复杂，我认为可以将 sincall 数组前 100 设置为一次递增函数，以第 100 为对称轴，轴对称后 100 的一次递减函数，代码如下：

```
//--------------------------------外设初始化
--------------------------------
        InitPeripherals();
////--------------------------------复位
AD7606----------------------------------
    for(k=0;k<200;k++)
    {

    //sincal1[k]= sin(TWOPAI*k*0.005);
    if (k<100){
     sincall[k] = k;
    }
    else{
     sincall[k] = -k+200;
    }
    }
```

示波器结果如下，可以看出波形变为了三角波。




原代码中在向 DAC 通道写值之前，首先检查该值是否在-0.9999 到 0.9999 的范围之外。如果是的话，它将被钳制在这个范围内最近的值。

我们计划结合 A/D 实验，当 A/D 电压大于某一阈值输出波形，否则不输出波形。故将原来代码中超出范围内的值全部置零，结果如下：

```
for(i=0;i<200;i++)
          {
       DA0=sincal1[i];
      if(DA0>=0.9999)       DA0=0;
      if(DA0<=-0.9999)    DA0=0;
      *DA_ADD0 = DA0 * 2048+2048;

       DA1= sincal1[i];
       if(DA1>=0.9999)   DA1=0;
       if(DA1<=-0.9999) DA1=0;
       *DA_ADD1 = DA1 * 2048+2048;

       DA2= sincal1[i];
       if(DA2>=0.9999)   DA2=0;
       if(DA2<=-0.9999) DA2=0;
       *DA_ADD2 = DA2 * 2048+2048;

        DA3= sincal1[i];
        if(DA3>=0.9999) DA3=0;
            if(DA3<=-0.9999)     DA3=0;
          *DA_ADD3 = DA3 * 2048+2048;
                }
```

## 十七、 实验心得与感受

1．熟悉了 CCS 的操作流程，包括如何使用工程、编译、下载和运行程序。

2．掌握了使用工具箱和 CCS 调试软件的基本方法，为进一步学习 DSP 外设与电机实验提供了基础。

3．了解了使用 GPIO 控制 LED 灯状态的方法，掌握了定时器的控制方法。

4．了解 F28335 内部的 A/D 转换和 D/A 转换的工作原理。

5．通过 4 个实验，我们的理论知识得以在实践中应用，深刻感受到认真严谨的态度和团队合作精神的重要性。任何一个实验都是需要大家共同努力去完成，任何一个细节出了问题都会导致最终的失败，所以就要求我们去负责的对待每个环节，每一个步骤。只要把握好整个过程，就会顺利的到达预期的目的。