中山大學 SUN YAT-SEN UNIVERSITY

电机拖动技术实验报告

院(系):智能工程学院　　学号：22354189　　姓名：张瑞程

日期：2024 年 12 月　　实验名称：第一次实验-DSP 基础外设实验

第一次实验-DSP 基础外设实验

目录

实验一：LED 灯控制实验

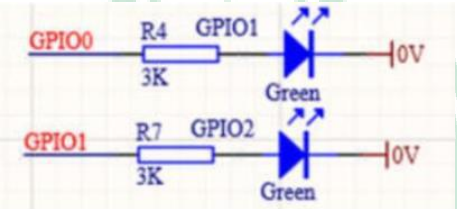
一、实验目的

学会使用 GPIO 控制 LED 灯状态；
学习配置寄存器并点亮 LED 灯的方法；
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

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 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；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：

```
17  void InitGpio(void)
18  {
19      asm(" EALLOW");                         // Enable EALLOW protected register access
20
21  //--- Group A pins
22      GpioCtrlRegs.GPACTRL.all  = 0x00000000;     // QUALPRD =1xSCLKOUT for  group A GPIO0-31
23      GpioCtrlRegs.GPAQSEL1.all = 0x00000000;     // No qualification for all group A GPIO 0-15
24      GpioCtrlRegs.GPAQSEL2.all = 0x00000000;     // No qualification for  group A GPIO 16-31
25      GpioCtrlRegs.GPADIR.all   = 0xFFFFFFFF;     // group A GPIO 0-31 AS OUTPUT
26      GpioCtrlRegs.GPAPUD.all   = 0x00000000;     //Pullups enabled GPIO31-20,GPIO0-15 disabled GPIO16-19
27
28
29      GpioCtrlRegs.GPAMUX1.bit.GPIO0  = 0;        // 0=GPIO   1=EPWM1A     2=rsvd       3=rsvd  d0??GPIO
30      GpioCtrlRegs.GPAMUX1.bit.GPIO1  = 0;        // 0=GPIO   1=EPWM1B     2=ECAP6      3=rsvd  d1??GPIO
```

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：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：

```
1. void delay(Uint16 t)
2. {
3.     Uint16 i;
4.     while(t--)
5.     {
6.         for(i=0; i<125; i++)
7.         {
8.             asm(" RPT #3 || NOP");
9.         }
10.    }
11. }
```

三、实验设备

（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO 配套电源。
（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

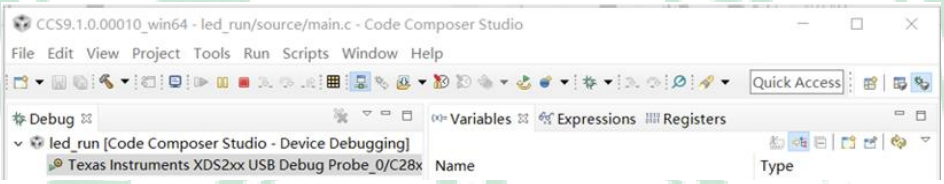
四、实验步骤

设备连接与测试：

（1）对实验设备进行硬件部分连接，连接好开发板的仿真器并上电，

（2）按照工程导入步骤导入光盘资料里的 led.run 工程，

（3）编译工程生成 LED.out 可执行程序，下载程序并运行。



可以观察到底板上的 2 个 LED（GPIO1，GPIO2）和核心板的 1 个 LED（RUN）都做闪烁点亮（见下图中标注）。

实验练习：

<1> 修改程序使得 LED 灯交替闪烁

void configtestledON(void) 里面的代码是点亮 LED

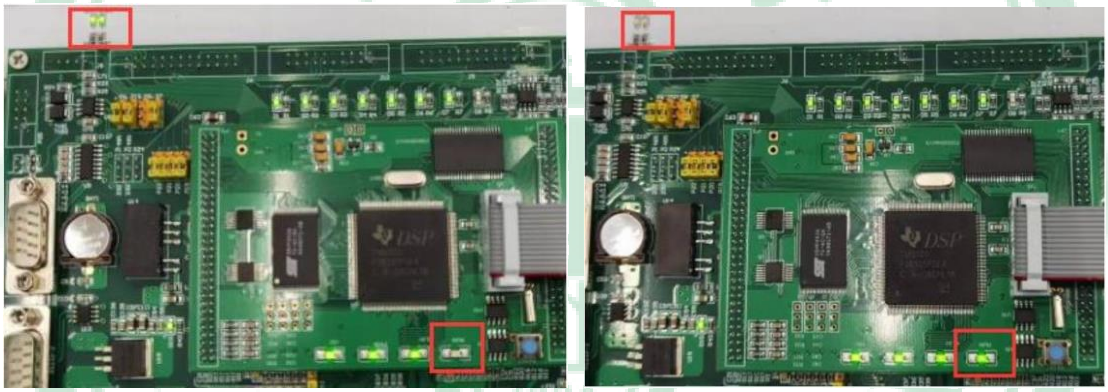
void configtestledOFF(void) 里面的代码是熄灭 LED

所以让 LED 灯交替闪烁只需要交换其中的代码即可。

代码修改如下：

```
1. void configtestledON(void)
2. {
3.     EALLOW;
4.     GpioDataRegs.GPASET.bit.GPIO6 = 1; //GPIO6 输出高电平，点亮 LED1
5.     GpioDataRegs.GPASET.bit.GPIO7 = 1; //GPIO7 输出高电平，点亮 LED2
6.     GpioDataRegs.GPASET.bit.GPIO23 = 1; //GPIO23 输出高电平，点亮 RUN
7.     EDIS;
8. }
9. void configtestledOFF(void)
10. {
11.     EALLOW;
12.     GpioDataRegs.GPACLEAR.bit.GPIO6 = 1; //GPIO6 输出低电平，熄灭 LED1
13.     GpioDataRegs.GPACLEAR.bit.GPIO7 = 1; //GPIO7 输出低电平，熄灭 LED2
14.     GpioDataRegs.GPACLEAR.bit.GPIO23 = 1; //GPIO23 输出低电平，熄灭 RUN
15.     EDIS;
16. }
```

重新运行程序，可以观察到 led 灯交替闪烁。



<2> 修改程序改变 LED 灯闪烁时间

通过对代码的调整，我们可以实现 LED 闪烁频率和闪烁间隔的控制。

对于调整闪烁频率，只需将 DELAY_US（）函数中的参数调大或调小：

```
1. while(1) // 主循环
2. {
3.     configtestledOFF(); // 熄灭 LED 灯，I/O 输出低电平
4.     DELAY_US(100000); // 延时    原来为：DELAY_US(500000);
5.     configtestledON(); // 点亮 LED 灯，I/O 输出高电平
6.     DELAY_US(100000); // 延时    原来为：DELAY_US(500000);
7. }
```

重新运行程序，可以观察到 led 灯闪烁频率加快。

对于调整闪烁间隔，只需在 LED1 与 LED2 添加延时函数新的 delay()函数，在点亮 LED 函数前的延时函数增加 10 倍时间：

```
1. void configtestledON(void) //点亮 LED 灯
2. {
3.     EALLOW;
4.     GpioDataRegs.GPASET.bit.GPIO6 = 1; //LED1
```

```
5.     delay(1000);
6.     GpioDataRegs.GPASET.bit.GPIO7 = 1; //LED2
7.     GpioDataRegs.GPASET.bit.GPIO23 = 1; //RUN
8.     EDIS;
9. }
```

重新运行程序，可以观察到 led 灯闪烁交替的频率明显变慢。

五、 实验结论与心得

本实验中，我们成功通过 GPIO 寄存器实现了对 LED 的点亮与熄灭控制。通过使用延时函数，我们能够控制 LED 的闪烁频率，并且通过调整延时时间来改变闪烁周期。通过交换 LED 点亮与熄灭的代码段，我们实现了 LED 的交替闪烁效果。此外，我们还学习了如何配置 GPxMUX、GPxDIR、GPxDAT 等寄存器，从而达到控制 LED 的目的。

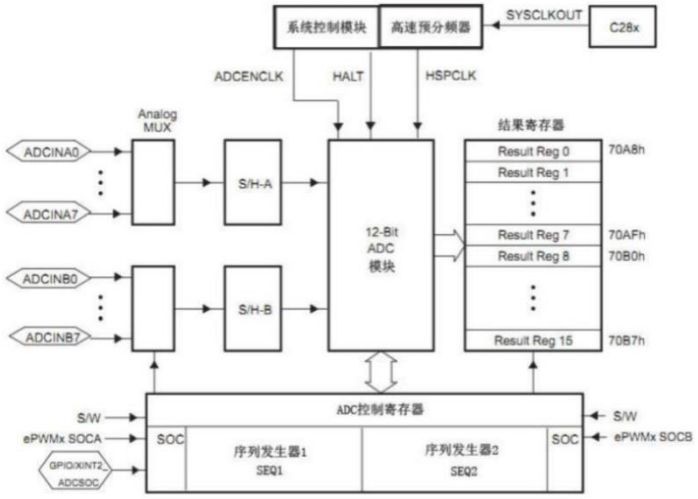
实验三：模数转换（A/D）测试实验

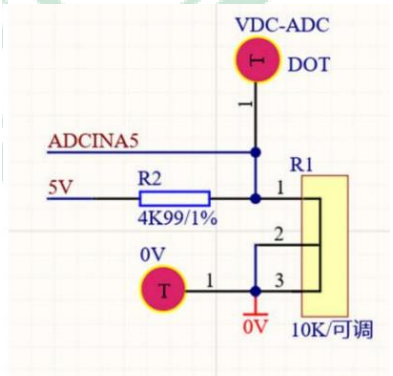
一、 实验目的

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。

二、 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 B，也就是图中的 S/HB。以下是相关原理图：





三、 实验设备

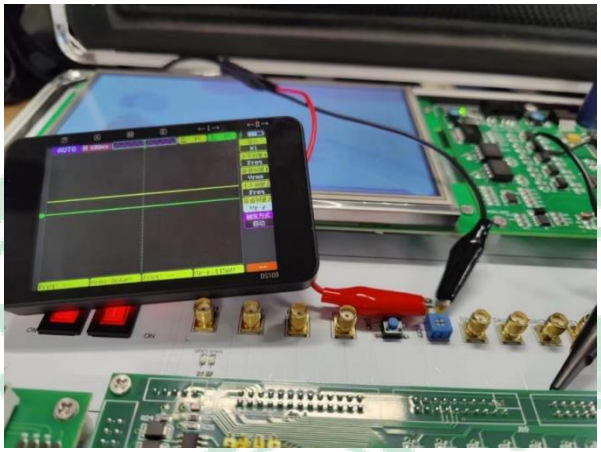
（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

四、 实验步骤

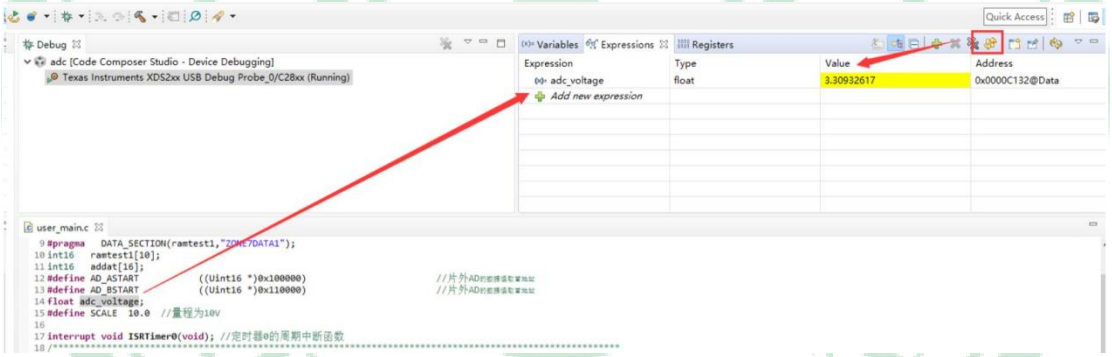
（1）首先开启设备连接 CCS，打开 adc 工程并测试是否连接成功；

（2）编译工程生成 ADC.out 可执行程序，将程序烧录到设别中并运行。

（3）将万用表调至直流档（由于没有万用表，这里我们利用示波器进行代替），用正负极分别连接底板上“VDC-ADC”和“0V”。设备连接状态如图（要注意两个电爪不要挨在一起，否则可能造成短路）：



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



这里实际测量数值的过程中使用的是示波器。对比两个示数，在误差可接受范围内相等。

多次用十字螺丝刀旋转可调电阻，重复实验以减少偶然性，结果依旧稳定地保持一致。

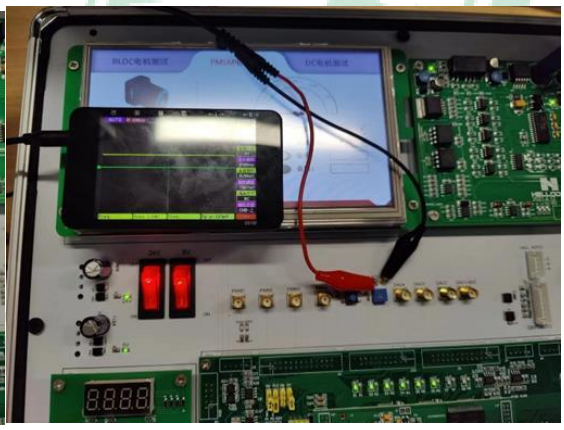
实验练习：

通过实验一的程序修改，实现当 A/D 转换的电压超过特定阈值时点亮 LED 灯。进一步的改进可以是，在 A/D 采样电压处于某一区间内时使 LED 灯闪烁，而当电压超过更高阈值时则完全点亮 LED 灯。

为了满足报告中提出的要求，根据采样电压的不同值来控制 LED 灯以不同的模式点亮。具体来说，当 A/D 采样电压大于 2V 时，LED 灯应完全点亮；而当 A/D 采样电压在 0V 到 1V 之间时，LED 灯应以闪烁模式工作。这需要我们在代码的主函数部分进行相应的修改，以下是主函数部分的代码修改示例：

```
// 主循环
for(;;) {
    // 当 adc_voltage 的值在 1.0 和 2.0 之间时
    while(adc_voltage > 1.0 && adc_voltage < 2.0) {
        ConfigtestledOFF(); // 关闭测试 LED
        DELAY_US(500000);    // 延时 500000 微秒（500 毫秒）
        ConfigtestledON();   // 打开测试 LED
        DELAY_US(500000);    // 延时 500000 微秒（500 毫秒）
    }
    // 当 adc_voltage 的值大于 2.0 时
    while(adc_voltage > 2.0) {
        ConfigtestledON();   // 打开测试 LED
    }
    ConfigtestledOFF(); // 关闭测试 LED
}
```

此主函数中的循环无退出条件，因此只要程序开始执行以后，就会不断进行判断。当电压在 1~2V 之间的时候 LED 灯会进行闪烁；当电压在 2V 以上的时候 LED 灯会常亮；当两个条件都不满足的时候，LED 灯会保持在熄灭状态。
我们分别调节电压 1.2 到 2.2，观察到 led 明暗闪烁状态如下图所示，实验成功。

五、 实验结论与心得

在本次实验中，我们深入探究了 F28335 微控制器的 A/D 模块，该模块具备 12 位的高分辨率和 16 个通道，这些通道被分为 A 组和 B 组，它们各自使用独立的采样保持器。我们成功地对 F28335 的 A/D 转换功能进行了测试，发现 ADC 转换的结果与示波器的测量值完全一致，从而确认了该模块的正常运作。此外，通过调整 A/D 采样电压，我们实现了 LED 灯的点亮和闪烁，这一过程加深了我们对

A/D 转换与输出控制应用的理解。

在修改代码过程中，最初，我们尝试将实验一中的四行代码直接粘贴到主函数中，结果遇到了函数无法识别的错误提示。随后，我们将函数定义连同所需的库包一并复制过来，却遇到了包无法找到的错误。最终，我们将必要的头文件复制到主函数所在的文件夹内，程序得以顺利运行。这次经历让我深刻体会到，编写代码是一个不断尝试、错误修正和调试的过程，每一步都朝着成功迈进。

实验四：数模转换（D/A）测试实验

一、 实验目的

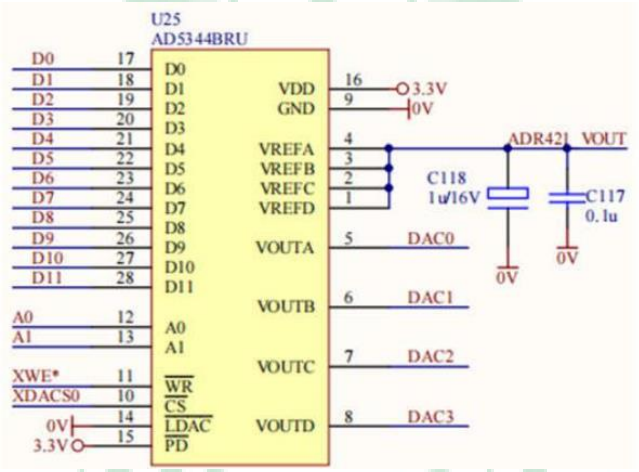
（1）测试 F28335 的 D/A 转换功能；

（2）了解 F28335 内部的 D/A 转换的工作原理。

二、 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。

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、 实验步骤

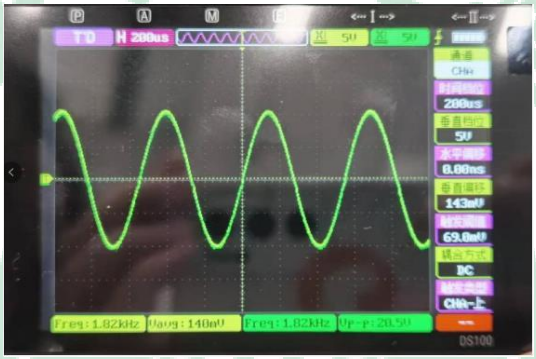
（1）参考基于 CCS 仿真调试和程序下载的实验步骤，开启设备连接 CCS，打开 CCS。

（2）对实验设备进行硬件部分连接，连接好开发板的仿真器并上电,打开示波器（长按按钮 P）,系统连接状态如下图所示：

（3）按照工程导入步骤导入光盘资料里的 dac 工程

（4）编译工程生成 dac.out 的可执行程序

（5）下载程序,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，观察波形程序运行后的波形图如下：



重复上述步骤连接 DAC2,DAC3,DAC4 等接口，都能观察到相似的正弦波形。

实验练习：

〈1〉修改程序输出不用的波形，方波，三角波等，进一步改进其幅值与频率，观察并记录实验现象。

*提示：数组 sincal1= sin(TWOPAI*k*0.005);保存的是正弦波。修改其他波形，就是改变 sincal1 数字里面的值。如果输出余弦波就修改位如下：*

for(k=0;k<200;k++)

*{ sincal1[k]= cos(TWOPAI*k*0.005); }*

我们首先观波形输出位置的代码，可以发现，第一个 K 循环用于生成样板波形，然后通过主循环将其赋到输出中，同时在输出中对值进行限幅。因此，要想修改输出的波形，只需要修改 K 循环中的波形生成函数即可。

```
1. // 复位 AD7606
2. for(k=0; k<200; k++) {
3.     sincal1[k] = sin(TWOPAI * k * 0.005);
4. }
5.
6. // 工作初始化
7. for(;;) {
```

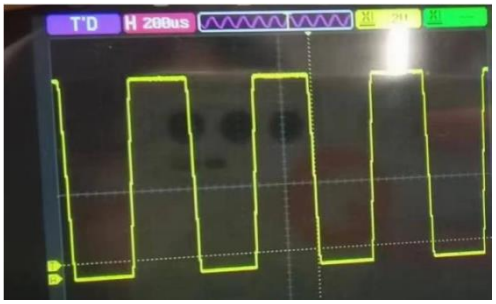
```
    for(i=0; i<200; i++) {
        DA0 = sincal1[i];
        if(DA0 >= 0.9999) DA0 = 0.9999;
        if(DA0 <= -0.9999) DA0 = -0.9999;
        *DA_ADD0 = DA0 * 2048 + 2048;

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

        DA2 = sincal1[i];
    }
}
```

对于**方波**的修改，我们通过设置两个标志变量（j 和 c），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。如果需要改变赋值，修改 0/1 两种结果即可；改变频率，修改 c 的判断条件即可。

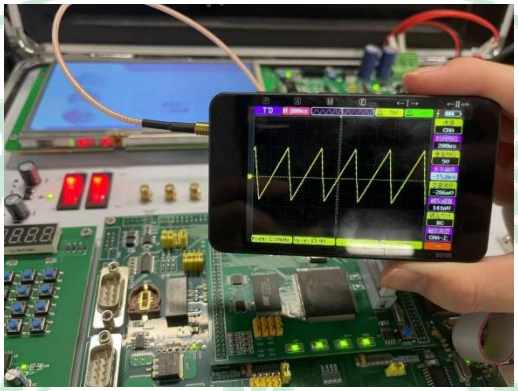
```
int16 j = 0;
int16 c = 0;
for(k = 0; k < 200; k++) {
    if(j == 0) {
        sincal1[k] = 1;
        c++;
        if(c == 100) {
            j = 1;
            c = 0;
        }
    } else {
        sincal1[k] = 0;
        c++;
        if(c == 100) {
            j = 0;
            c = 0;
       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



对于**三角波**的修改，可以将 sincall 数组前 100 设置为递增函数，以第 100 为对称轴，轴对称后 100 的递减函数，代码如下：

```
for(k = 0; k < 200; k++) {
    if(k < 100) {
        sincal1[k] = k;
    } else {
        sincal1[k] = -k + 200;
    }
    // sincal1[k] = sin(TWOPAI * k * 0.005); // 这行代码被注释掉了
}
```

运行后结果结果如下，可以成功观察到三角波形。



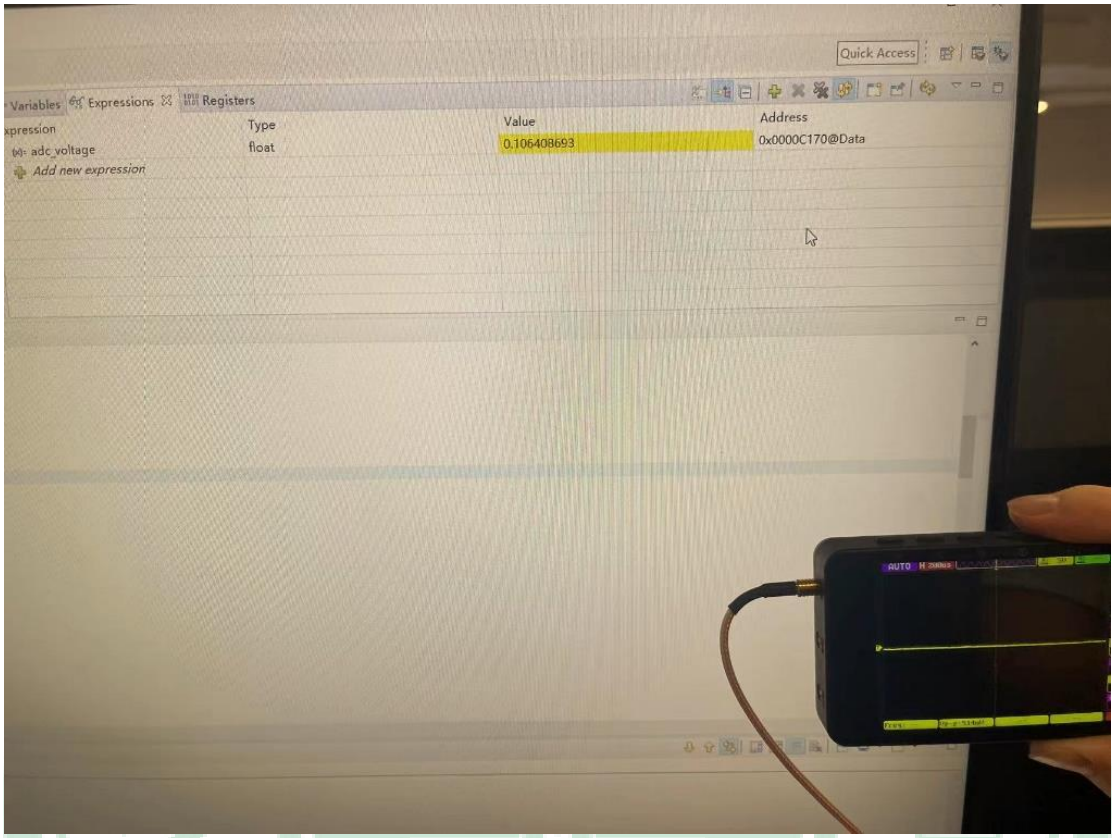
<2>结合 A/D 实验（用 dac_adc 程序），当 A/D 电压大于某一阈值输出波形，否则不输出波形，利用 dac_adc 程序实现，观察并记录实验现象。

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。我们通过控制 adc_votage 来控制波形的输出功能，当 A/D 电压大于某一阈值输出波形，否则不输出波形。代码如下：

```
// 主循环
for(;;) {
    while(adc_voltage > 2.0) {
        for(i = 0; i < 200; i++) {
            DA0 = sincali[i];
            if(DA0 >= 0.9999) DA0 = 0.9999;
            if(DA0 <= -0.9999) DA0 = -0.9999;
            *DA_ADD0 = DA0 * 2048 + 2048;
            DA1 = sincali[i];
            if(DA1 >= 0.9999) DA1 = 0.9999;
            if(DA1 <= -0.9999) DA1 = -0.9999;
        }
    }
}
```

具体来讲，就是在主循环外面再套上一个 while 循环，使得只有 adc_votage>2.0 的时候才可以输出波形，不然就都是没有波形的。

结果展示：（当电压小于 2.0V 的时候没有输出）



五、 实验结论与心得

在本次实验中，我们深入探究并验证了 F28335 开发板的数字到模拟（D/A）转换功能。同时，我们也对 AD5344 模块的工作原理有了更加深刻的理解。通过精心编写和调整代码，我们成功地生成了多种波形，包括方波和三角波等，并通过示波器对这些波形进行了直观的观察和验证。我们通过设定数值范围的限制，结合模拟到数字（A/D）转换实验，实现了基于特定阈值的波形输出控制。

通过本次实验，我们不仅掌握了 D/A 转换的基本概念和操作技能，还学会了如何运用这些知识来解决实际问题。此外，实验过程中的调试和优化也让我们更加熟悉了开发板和模块的性能特点，为未来的项目开发打下了坚实的基础。

实验五：PWM 实验

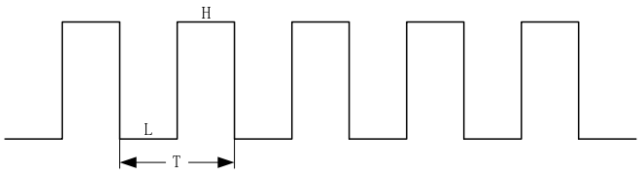
一、实验目的

（1）学会配置 PWM 对应引脚。

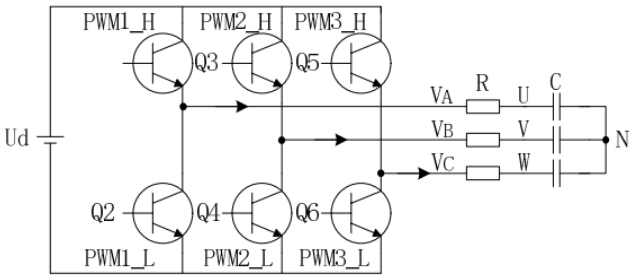
（2） 对应的引脚可以输出波形。

二、实验原理

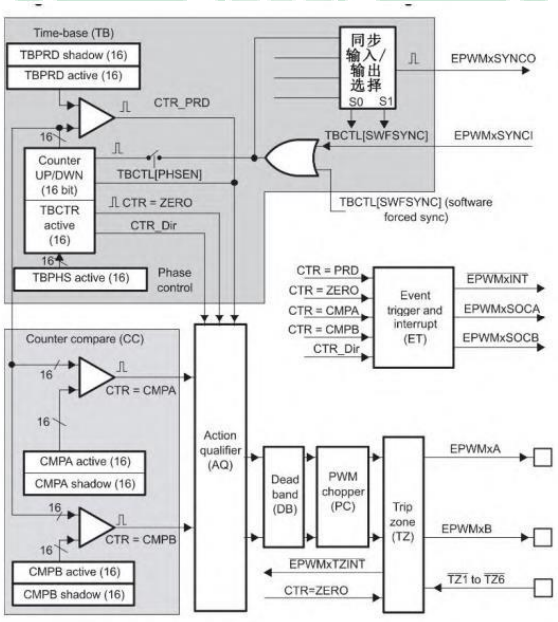
PWM 是 Pulse Width Modulation 的缩写，即脉宽调制，通俗点讲就是宽度可调节的方波脉冲，如下图所示。



在实际应用中，PWM 用于驱动开关器件，比如 MOSFET、IGBT 或者 IPM 模块，PWM 输 出的高低电平刚好可以控制开关器件的导通或者关断，从而实现通过改变输出方波的占空比来改变等效的输出电压。下图是电力电子中最为常见和实用的一种拓扑结构，通过六路 PWM 来控制六个开关管，可以将直流电压 Ud 逆变成对称的三相交流电 U、V、W。



TMS320F28335 有 6 个 ePWM 模块，ePWM1、ePWM2、ePWM3、ePWM4、ePWM5 和 ePWM6。每个 ePWM 模块都有相同的内部逻辑电路，因此在功能上这 6 个 ePWM 模块都是相同的。
下图为 ePWM 模块内部结构框图。

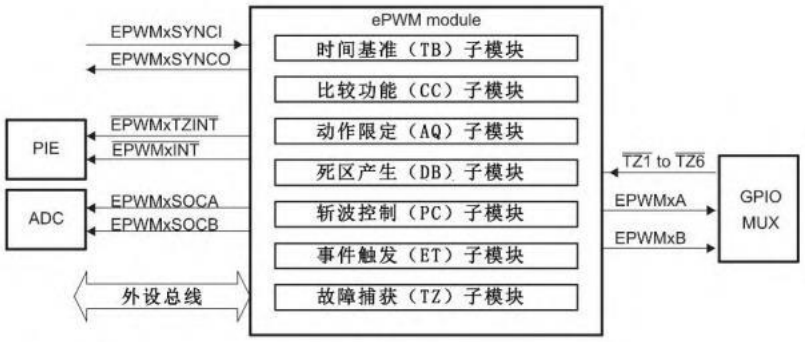


从上图可以看出，ePWM 模块内部包含有 7 个子模块，分别是时间基准子模块 TB、 比较功能子模块 CC，动作限定子模块 AQ、死区控制子模块 DB、斩波控制子模块 PC、事件触发子模块 ET 和故障捕获子模块 TZ，正是由这几个子模

块的配合，才可以方便地得到所需的 PWM 波形。 每个 ePWM 模块都具有以下功能：

- 可以输出两路 PWM，EPWMxA 和 EPWMxB；
- 两路 PWM 可以独立输出，也可以互补输出；
- 具有相位控制功能，可以超前或者滞后于其他 ePWM 模块；
- 具有死区控制功能，可以分别对上升沿和下降沿进行延时控制；
- 具有故障保护功能，通过对触发条件的设置，当故障发生时，可自动将 PWM 输出引脚设置为低电平、高电平或高阻状态；
- 具有高频斩波功能，高频斩波信号对 PWM 进行斩波控制，用于高频变换器的门极驱动；
- 所有事件都可以触发中断，也都可以产生内部 ADC 转换的启动脉冲 SOC。

下图为 ePWM 模块所包含的子模块以及相关的信号。



从上图可以看出：

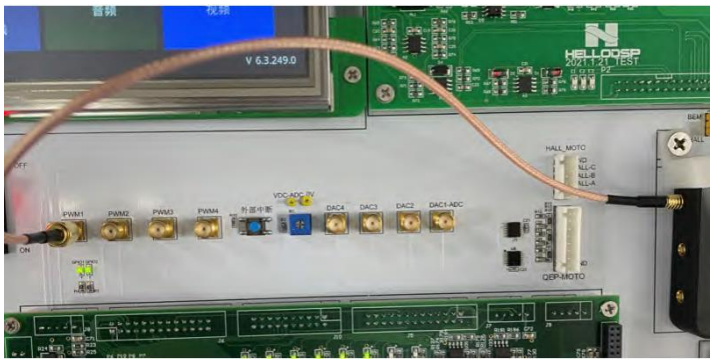
- PWM 输出信号 EPWMxA 和 EPWMxB，这两个信号通过 GPIO 口输出，从而产生 PWM 波；
- 故障触发信号 TZ1~TZ6。这些故障触发信号是用来通知 ePWM 模块，外部电路出现了故障，需要立即停机，从而实现保护功能。每个 ePWM 模块都可以使用或屏蔽掉故障触发信号。TZ1~TZ6 可以配置成同步输入模式，并从相应的 GPIO 口输入。
- 时钟基准同步信号输入 EPWMxSYNCI 及输出 EPWMxSYNCO，同步信号可以将所有的 ePWM 模块连接成一个整体，当然，每个 ePWM 模块都可以通过设置，使用或者忽略同步信号。
- ADC 启动信号 EPWMxSOCA 和 EPWMxSOCB。

三、 实验设备

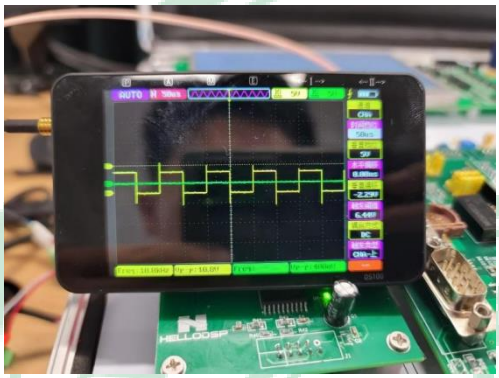
（1）硬件：BOX28335 实验平台，仿真器 HDSP-XDS200ISO，相应的配套电源。
（2）软件：安装了 Windows7/10 和 CCS 软件的 PC。

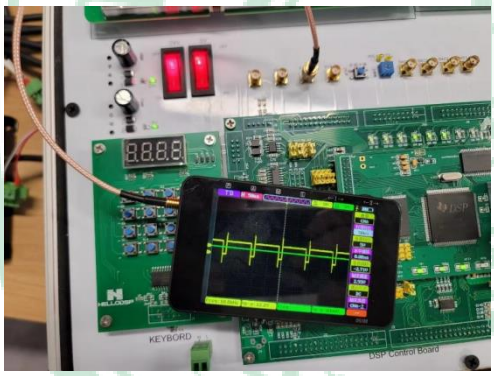
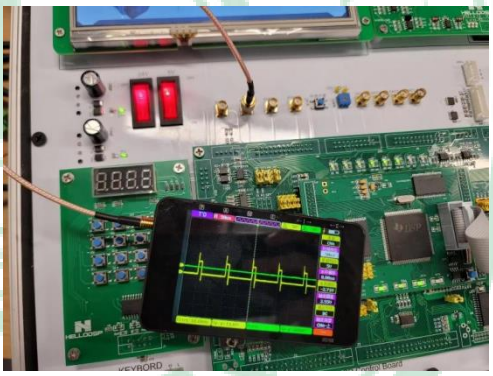
四、实验步骤

基于 CCS 仿真调试和程序下载的实验步骤，开启设备连接 CCS，打开 CCS。
（1）对实验设备进行硬件部分连接，连接好开发板的仿真器并上电，打开示波器（长按按钮 P）
（2）按照工程导入步骤导入光盘资料里的 pwm 工程，
（3）编译工程生成 pwm.out 的可执行程序，
（4）下载程序，将示波器的 CHA 通道与底板上 PWM1 接口用 SMA 线相连接，按键 A 重置示波器，运行程序，观察波形，连接方式如下图所示：



（5）重复操作上述步骤分别连接 PWM2，PWM3，PWM4 等接口。
连接 PWM1，PWM2，PWM3，PWM4 四个通道都能在示波器上观察到波形。
按顺序四路通道输出如下：

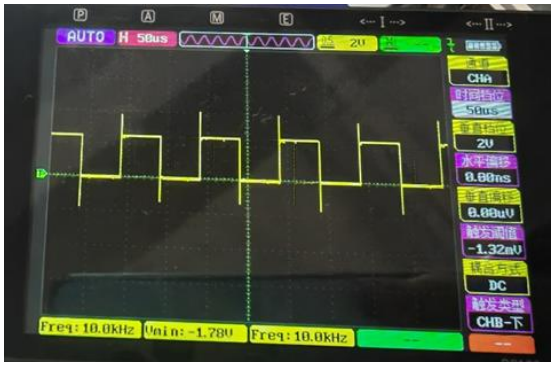


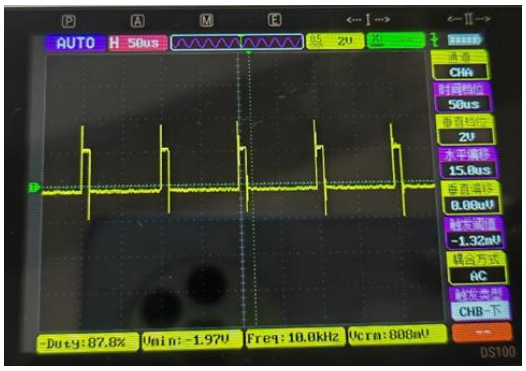
实验练习

〈1〉尝试调节 PWM 方波的占空比，通道 1 调节调节 EPwm1Regs.CMPA.half.CMPA 的值，其他通道以此类推，观察并记录实验现象。

我们更改了 1—6 通道{ EPwmnRegs.CMPA.half.CMPA = SP / 8; } SP 的比值，观察到更改前后占空比发生对应的改变。



SP / 8



SP / 32

<2> 结合 A/D 实验，当 A/D 电压改变时改变 PWM 方波占空比，参考 P114 。或者用 adc_pwm 程序

我们参考实验手册 114 页提供的指引对项目代码进行修改。首先，在主函数中对代码进行修改，读取 adc_votage 的值，将其处理后赋值给 sp

```
while(1) {
    ADC_Ctr1(); // ADC 采样
    adc_voltage = ((float) Get_A_Adc(5)) * 0.00030518;
    sp = (SP * (adc_voltage / 3.5));
    DELAY_US(1800);
    StartEPwm();
}
```

然后在 StartEPwm()函数中，使用 EPwm1Regs 表达式调整方波信号的占空比。代码如下：

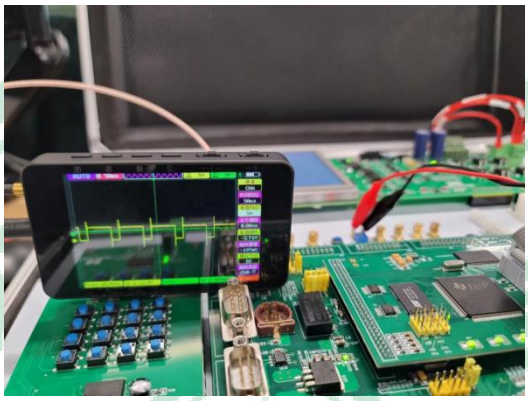
```
void StartEPwm()
{
    EALLOW; // 允许对受保护的寄存器进行写操作
    // 同步时钟到定时器
    SysCtrlRegs.PCLKCRe.bit.TBCLKSYNC = 0;
    // 重置定时器状态和计数器
    EPwm1Regs.TBSTS.all = 0;
    EPwm1Regs.TBPHS.half.TBPHS = 0; // 设置计数器的初始相位
    EPwm1Regs.TBCTR = 0; // 重置计数器的值
    // 配置比较控制寄存器，这里设置为 0x5E，具体含义取决于硬件手册
    EPwm1Regs.CMPCTL.all = 0x5E;
    // 设置比较 A 的值为 sp，比较 B 的值为 0，sp 可能是之前计算得到的值
    EPwm1Regs.CMPA.half.CMPA = sp;
    EPwm1Regs.CMPB = 0;
    // 配置动作定标寄存器 A，这里设置为 0x6E，具体含义取决于硬件手册
    EPwm1Regs.AQCTLA.all = 0x6E;
    // 配置动作定标寄存器 B，这里设置为 0x0，表示没有动作
    EPwm1Regs.AQCTLB.all = 0;
    // 强制加载阴影寄存器，这里设置为 0x0，表示不强制加载
    EPwm1Regs.AQSFRC.all = 0;
    EPwm1Regs.AQCSFRC.all = 0x9;
```

```
22. }
```

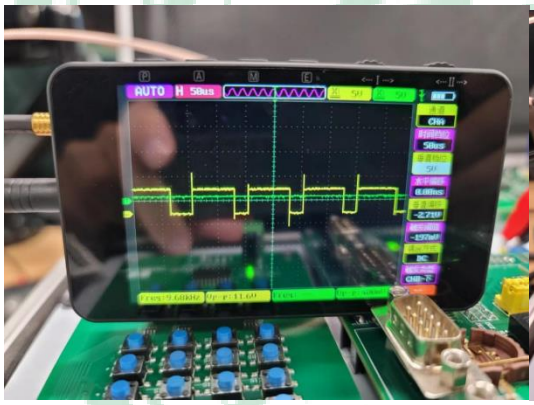
同时还要对各个 PWM 输出都做以调整。

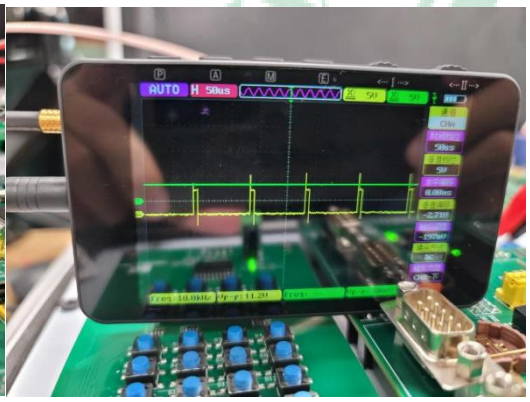
```
1. EPwm2Regs.CMPA.half.CMPA = sp;
1. EPwm3Regs.CMPA.half.CMPA = sp;
1. EPwm4Regs.CMPA.half.CMPA = sp;
```

实验的时候，用示波器监控两路输出，一个是 PWM 口输出的方波信号，观察占空比；另一个是用电压线监控 adc_votage 的值；同时还要调整 ADC 处旋钮的值。系统连接状态如图：



观察两者之间变动的联系，结果如下：





可以显著地观察到，随着指示 adc_votage 值的绿线上升（adc_votage 上升），方波的占空比显著逐渐增加，且到了最后的极端情况下变成了一条直线。成功实现了当 A/D 电压改变时改变 PWM 方波占空比的要求。

五、 实验结论与心得

通过本次 PWM 实验，我成功掌握了 TMS320F28335 开发板的 PWM 配置方法，并深入理解了脉宽调制的工作原理。实验中，通过调整 ADC 采样值影响 PWM 占空比，实现了电压到 PWM 信号的转换。观察示波器上的波形变化，我直观地感受到了数字信号控制模拟信号的过程。此外，实验中的故障保护和高频斩波等功能，让我对 ePWM 模块的全面性有了更深的认识。

实验心得

在完成这一系列实验后，我深刻体会到了理论与实践相结合的重要性。通过亲手操作实验设备，我对 DSP 基础外设有了更加深入的了解，特别是在 GPIO 控制、模数转换（A/D）、数模转换（D/A）以及 PWM 波形生成等方面获得了宝贵的实践经验。

实验中，我学会了如何配置和操作 GPIO 引脚来控制 LED 灯的状态，这不仅加深了我对数字电路的理解，也锻炼了我的动手能力。在模数转换实验中，我通过实际操作了解到了 F28335 微控制器的 A/D 模块的高分辨率和多通道特性，以及如何通过代码控制来实现电压测量和 LED 灯的点亮与闪烁，这让我对微控制器的应用有了更加直观的认识。数模转换实验让我对 D/A 转换器的工作原理有了更加深刻的理解，通过编写和调整代码，我成功地生成了多种波形，并通过示波器观察到了这些波形的实际效果。在 PWM 实验中，我学会了如何配置 PWM 引脚并输出波形，通过调整占空比，我能够控制电机的转速和方向，这一过程让我对电机控制有了更加直观的感受。

总的来说，这些实验不仅增强了我的理论知识，也提高了我的实践技能。我相信这些经验将在我未来的学习和工作中发挥重要作用。