

Cortex-M4原理与实践实验报告

题目：PWM呼吸灯实验

**作者姓名 林宇航**

**指导教师**  **郭方洪**

**专业班级 自动化1901**

**学 院 信息工程学院**

**提交日期** 2021年11月22日

**一、实验目的**

1. 了解 PWM 基本概念。

2. 学习 TM4C129x Series Cortex-M4 的 PWM 工作原理与方式。

3. 学习 PWM 相关库函数的使用。

4. 学习对 PWM 模块输出周期和占空比的设置。

**二、实验设备**

1.计算机一台，操作系统为WindosXP或Windos7，装有CCSv6.0软件。

2.EK-TM4C1294XL实验开发板一块。

3.USB连接线一条。

**三、实验原理**

**1.PWM简介**

脉冲宽度调制（PWM，Pulse-Width Modulation)，使用了微处理器的数字输出实现对模拟电路的控制，这种技术被广泛应用于测量，通信，控制，变换等各种领域中。

在PWM中，电压或电流源是以一种通或断的重复脉冲序列形式被加到模拟负载上去的。通的时候即是直流供电被加到负载上，断的时候即是供电被断开。在给定的任何时刻，满幅值的直流供电要么完全有，要么完全无，只有0和1两种状态，所以PWM信号是数字的。这也是PWM信号的优点之一，信号保持为数字形式，可以将噪声降到最小，只有在噪声强到足以将逻辑1改变为逻辑О或将逻辑О改变为逻辑1时，也才能对数字信号产生影响。

**2.PWM基本概念**

占空比(Duty Cycle)在电信领域中有如下含义:在一串理想的脉冲周期序列中(如方波)，正脉冲的持续时间与脉冲总周期的比值。占空比是指高电平在一个周期之内所占的时间比率。

死区时间一般是指控制不到的时间域。在PWM 中，死区时间是在输出时，为了使H桥或半H桥的上下管不会因为开关速度问题发生同时导通而设置的一个保护时段。通常也指PWM 的响应时间。死区时间设置的大，PWM模块工作会更加可靠，但是会导致输出波形失真并且降低输出效率。如果死区时间设置的小，输出波形要好一些，但是会降低可靠性。所以死区时间的设置一般是在保证安全的条件下，越小越好。

**3.TM4C129x Series Cortex-M4的PWM工作原理与方式**

TM4C129x Series Cortex-M4包含一个PWM模块，提供了4个PWM 发生器模块和一个控制模块，一共有8路PWM信号输出。每个PWM发生器模块输出2路PWM信号，可以共享时钟和频率，也可以通过编程控制为独立工作，或者为一对带死区的互补信号。PWM发生模块的输出信号，pwmA'和 pwmB'，在被传送到控制器引脚前由输出控制模块管理，输出为MnPWM0，MnPWM1或者为MnPWM2，MnPWM3等等。

每个PWM发生器模块的主要特性:

（1）4个故障状态处理输入，能迅速提供低延迟关断，防止损坏控制的电机;

（2）一个16位计数器，两个PWM比较器;

（3）死区发生器;

令丰富的中断机制和ADC触发;

灵活可控的输出控制模块,多种输出方式;

安全可靠的错误检测保护功能。

PWM模块整体结构和PWM信号发生器整体结构分别如下图3-1，3-2所示。

图示, 示意图

描述已自动生成

图示, 示意图

描述已自动生成

**4.实验用到的主要库函数简介**

void PWMClockSet(uint32\_t ui32Base,uint32\_t ui32Config)函数用于配置PWM时钟，实现分频。ui32Base为 PWM模块的基地址; ui32Config，为配置PWM时钟的参数。

void PWMGenConfigure(uint32\_t ui32Base, uint32\_t ui32Gen,uint32\_t ui32Config)函数用于配置PWM 发生器。ui32Base为 PWM模块的基地址; ui32Gen为需要配置的PWM 发生器; ui32Config为配置PWM 发生器工作方式的参数。

void PWMGenPeriodSet (uint32\_t ui32Base, uint32\_t ui32Gen,uint32\_t ui32Period)函数用于配置PWM发生器的周期。ui32Base为PWM模块的基地址;ui32Gen为需要配置的PWM发生器; ui32Period为配置PWM 发生器的周期参数。

void PWMPulseWidthSet(uint32\_t ui32Base, uint32\_t ui32PWMOut,

uint32\_t ui32Width)函数用于配置指定PWM发生模块的脉冲宽度。ui32Base为PWM模块的基地址; ui32PWMOut为PWM输出引脚; ui32Width为脉冲的正频部分。

void PWMOutputState(uint32\_t ui32Base,uint32\_t ui32PWMOutBits,bool bEnable)函数用于使能或者禁止相应的PWM输出。ui32Base为PWM模块的基地址; ui32PWMOutBits 通过逻辑与选择对应的PWM引脚;bEnable为 true，则使能对应PWM引脚，否则，禁止相应引脚。

void PWMGenEnable(uint32\_t ui32Base, uint32\_t ui32Gen)函数用于使能相应PWM 发生器模块。ui32Base为 PWM模块的基地址; ui32Gen为需要使能的PWM 发生器模块。

上述函数及暂未用到的其他PWM 相关函数详细使用方法请参阅driverlib 库函数介绍sW-TM4C-DRL-UG-2.1.0.12573 .pdf。

PWM呼吸灯实验通过适当控制PWM的频率、占空比，延时与适当的占空比步进值flip来调节LED灯的亮度，明暗变化，产生呼吸效果。

**5.PWM频率与占空比计算**

PWMValue=PWMDividedCIk/PWMFrequency-1

这里PWMValue为填入PWMOLOAD寄存器的值;PWMDividedCk为分频后的PWM模块的时钟频率;PWMFrequency为所需要的PWM频率。在减计数模式下，将两频率之比减一后的值填入PWMOLOAD寄存器。

**四、实验流程图**

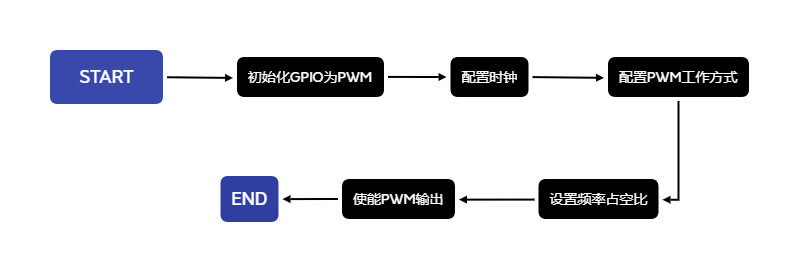


图 1 简化PWM初始化流程

**五、实验代码、注释**

#include <stdbool.h>

#include <stdint.h>

#include "inc/hw\_memmap.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h"

#include "driverlib/pwm.h"

#include "driverlib/sysctl.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

uint32\_t g\_ui32SysClock;

void main(void)

{

volatile uint32\_t ui32Loop=0,PWMPulseWidth=50;//设置初始占空比

volatile int32\_t flip=4;//亮度变化速度初始值

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |SYSCTL\_CFG\_VCO\_480), 5000000);//5MHz系统时钟

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_PWM0);//PWM0模块使能

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);//GPIOF模块使能

//设置PF1、PF2、PF3引脚复用功能为PWM

GPIOPinConfigure(GPIO\_PF1\_M0PWM1);

GPIOPinConfigure(GPIO\_PF2\_M0PWM2);

GPIOPinConfigure(GPIO\_PF3\_M0PWM3);

//配置相应引脚用于PWM功能

GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

GPIOPinTypePWM(GPIO\_PORTF\_BASE, GPIO\_PIN\_3);

//配置两个PWM发生器为减计数、立即更新方式

PWMGenConfigure(PWM0\_BASE, PWM\_GEN\_0, PWM\_GEN\_MODE\_DOWN |PWM\_GEN\_MODE\_NO\_SYNC);

PWMGenConfigure(PWM0\_BASE, PWM\_GEN\_1, PWM\_GEN\_MODE\_DOWN |PWM\_GEN\_MODE\_NO\_SYNC);

//配置两个PWM发生器工作频率为5MHz/4/10000=125Hz

PWMGenPeriodSet(PWM0\_BASE, PWM\_GEN\_0, 10000);

PWMGenPeriodSet(PWM0\_BASE, PWM\_GEN\_1, 10000);

PWMOutputState(PWM0\_BASE, PWM\_OUT\_1\_BIT | PWM\_OUT\_2\_BIT | PWM\_OUT\_3\_BIT, true);//使能PWM输出信号

//使能两个PWM发生器模块

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_0);

PWMGenEnable(PWM0\_BASE, PWM\_GEN\_1);

while(1)

{

//通过延时一段时间，使LED保持对应的亮度

for(ui32Loop=0;ui32Loop<1000;ui32Loop++)

{

;

}

//设置占空比以调节LED灯亮度

PWMPulseWidthSet(PWM0\_BASE, PWM\_OUT\_1,PWMPulseWidth);

PWMPulseWidthSet(PWM0\_BASE, PWM\_OUT\_2,PWMPulseWidth);

PWMPulseWidthSet(PWM0\_BASE, PWM\_OUT\_3,PWMPulseWidth);

PWMPulseWidth += flip;

if(PWMPulseWidth >= 9950)//决定呼吸灯最大亮度

{

flip = -4;//设置亮度为递减

}

else if(PWMPulseWidth <= 50)//决定呼吸灯最小亮度

{

flip = 4;//设置亮度为递增

}

}

}

**实验现象：**

LED0、LED1、LED2实现呼吸灯效果



**四、思考题**

**1. 当设置PWM的频率过高或过慢时，LED闪烁将如何变化？**

通过调用PWMGenPeriodSet（）函数来调整PWM的工作频率，从而来调整LED灯的闪烁。

当设置的PWM频率过高时，灯变化的速度会变快，然后会有一段常亮时间，若高到一定程度，LED灯会保持常亮。

当设置的PWM频率过低时，灯变化的速度会变慢，同时最大亮度也会减小（这里当我设置频率较低时，程序出现了警告，所以并没有看到LED灯常暗的情况）。

**2. flip过大或过小，LED闪烁将如何变化？**

程序中的PWMPulseWidth值正比于PWM的占空比，程序每执行一次，通过fiip值来调整PWMPulseWidth值，从而改变PWM的占空比使得PWMPulseWidth值在50到9950之间交替变换，来实现LED灯的明暗变换，以实现呼吸效果。

Flip值过大：闪烁频率越来越大，直至常亮（就是50和9950两个占空比切换，视觉上就是常亮）。

Flip值过小：闪烁频率越来越小，若减小到零，则LED的亮度保持不变。

**3. 另一个实验程序给出了直接配置 PWM 相关寄存器来控制 PWM 输出，实现 LED 呼吸灯，试比较直接配置寄存器和调用库函数的优缺点。**

**直接配置寄存器：**

**优点：**

程序运行占用资源少，直接对寄存器操作。

**缺点：**

代码长度以及每个参数的配置都需要自己设定，较为麻烦。

**调用库函数：**

**优点：**

1.用API函数，理解函数使用的功能以及参数，简洁明了。

2.用函数封装对寄存器的操作，代码的易读性提高，操作库函数更方便。

3、库对单片机底层寄存器配置要求不是十分高。不要求使用者对驱动层十分了解。

**缺点：**

1.执行速度要比直接操作寄存器慢。

2.每个API函数的功能需要自己查阅手册。

**五、实验体会与心得**

1）LED通过PWM来开关，当频率达到人眼CFF值（60Hz）的时候，看上去就是一个常亮的灯，当然，频率不能盲目得高，高于LED开关频率

2）当占空比小于一定值，产生的等效电流不足以点亮LED灯，所以到达点亮的阈值即可切换占空比自加，同理，当占空比大于一定值，等效电流已经达到LED亮度的最大值，所以达到LED电流最大值的时候即可切换占空比自减。

3）LED呼吸的呼吸频率可以通过修改每次自加或自减的步长（本程序中为flip），flip绝对值越大呼吸越快，反之越慢。

总得来说通过本次实验，我学会了利用PWM模块输出不同占空比的脉冲，来实现呼吸灯的程序。我通过修改不同的参数，对整个程序有了深刻且透彻的理解。但利用PWM模块实现呼吸灯，只能算是一个入门的程序，更加高级的应用，还需要更深入的学习。