DSP课程实验指导书

C2000 Piccolo LaunchPad实验套件

（2019版）

第一章 实验平台介绍

一、TMS320F28027硬件资源简介

1. **高效率 32 位 CPU(TMS320F2802X)**

* 60MHZ 时钟频率，单周期指令16.67ns
* 16\*16 和 32\*32 乘法运算
* 16\*16 双乘法器
* 哈佛总线结构
* 原子操作
* 快速中断响应和处理
* 统一的存储器编程模式
* 高代码效率(C/C++和汇编)

1. **低设备和系统成本**

* 单一 3.3V 供电、无电源排序要求
* 上电复位和掉电复位
* 低功耗

1. **时钟系统**

* 2 路内部零管脚锁相环
* 片上晶体振荡器/外部时钟输入
* 看门狗时钟模块
* 时钟丢失检测电路

1. **22个可编程，带输入滤波的多路复用GPIO引脚**
2. **外设中断扩展PIE模块，支持所有外设中断**
3. **3个32位CPU定时器**
4. **每个ePWM模块具有16位独立定时器**
5. **片上存储器**

* Flash（16位32k）
* SARAM（16位 6k）
* OTP（16位 1k）
* BOOTROM

1. **128位安全密钥**

* 保护存储器模块的安全
* 防止固件的逆向操作

1. **通信接口**

* 一路UART模块
* 一路SPI模块
* 一路IIC模块

1. **增强的控制外设**

* 两组共8路增强型脉宽调制器(ePWM)
* 高精度 PWM(HRPWM)
* 增强型捕获模块(eCAP)
* 13路12位模拟数字转换器（ADC），转换时间216.67ns
* 片上温度传感器

二、TMS320F28027引脚图

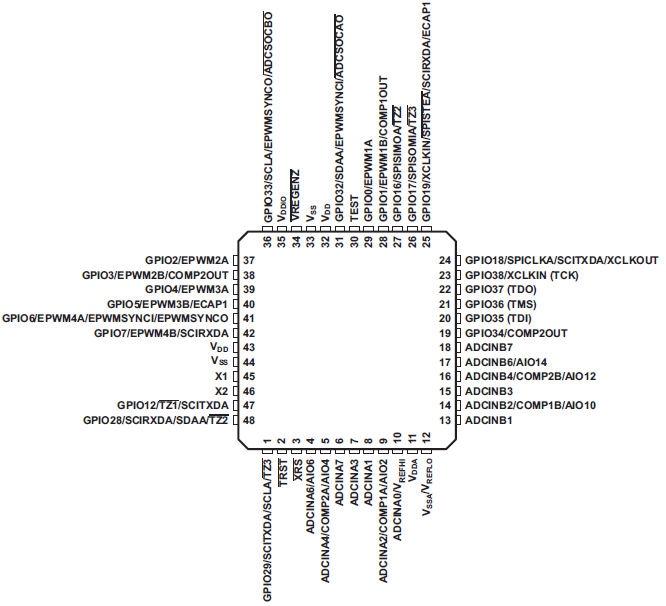


图1.1 芯片引脚图(俯视)

三、实验套件介绍

实验套件由底板和最小系统板组成。最小系统板如下图所示：

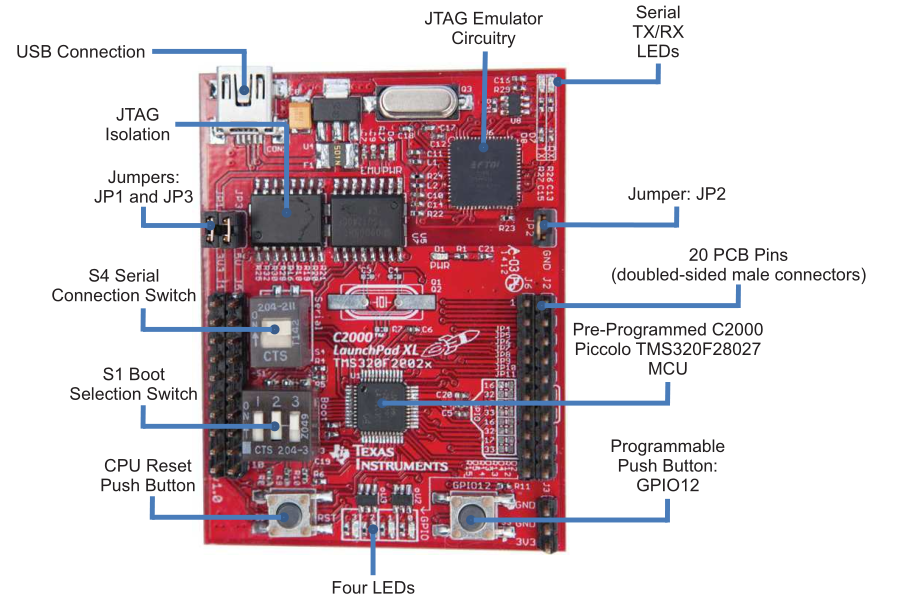


图1.2 最小系统板俯视图

最小系统板通过两侧的4行排针引出片上外设，排针对应引脚功能如下表所示：

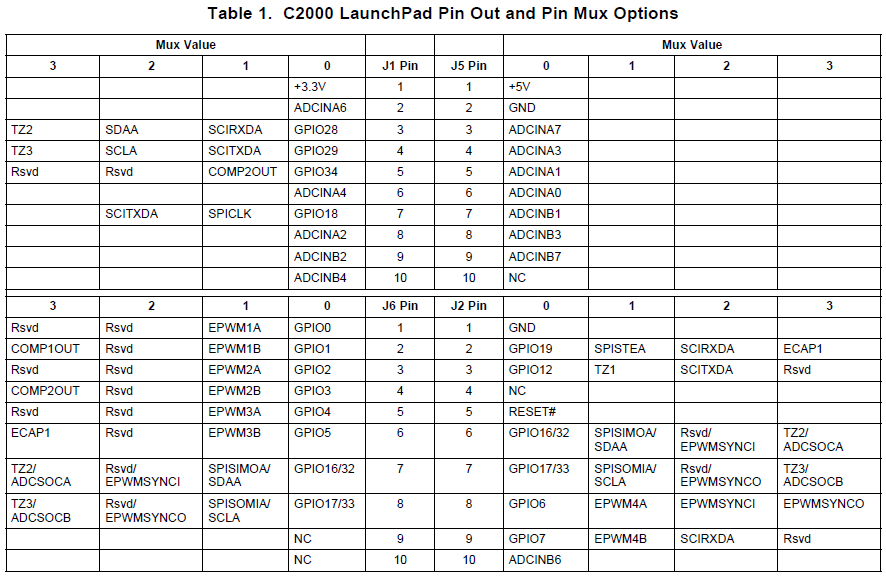


图1.3 芯片引脚功能含义说明

最小系统板通过一根miniUSB线可以连接PC机。miniUSB线缆有两个作用：对系统板供电和连接SCI串口。其中SCI串口连接的就是位于J1 排针上面的3、4号脚，即对应片上的GPIO28、29。

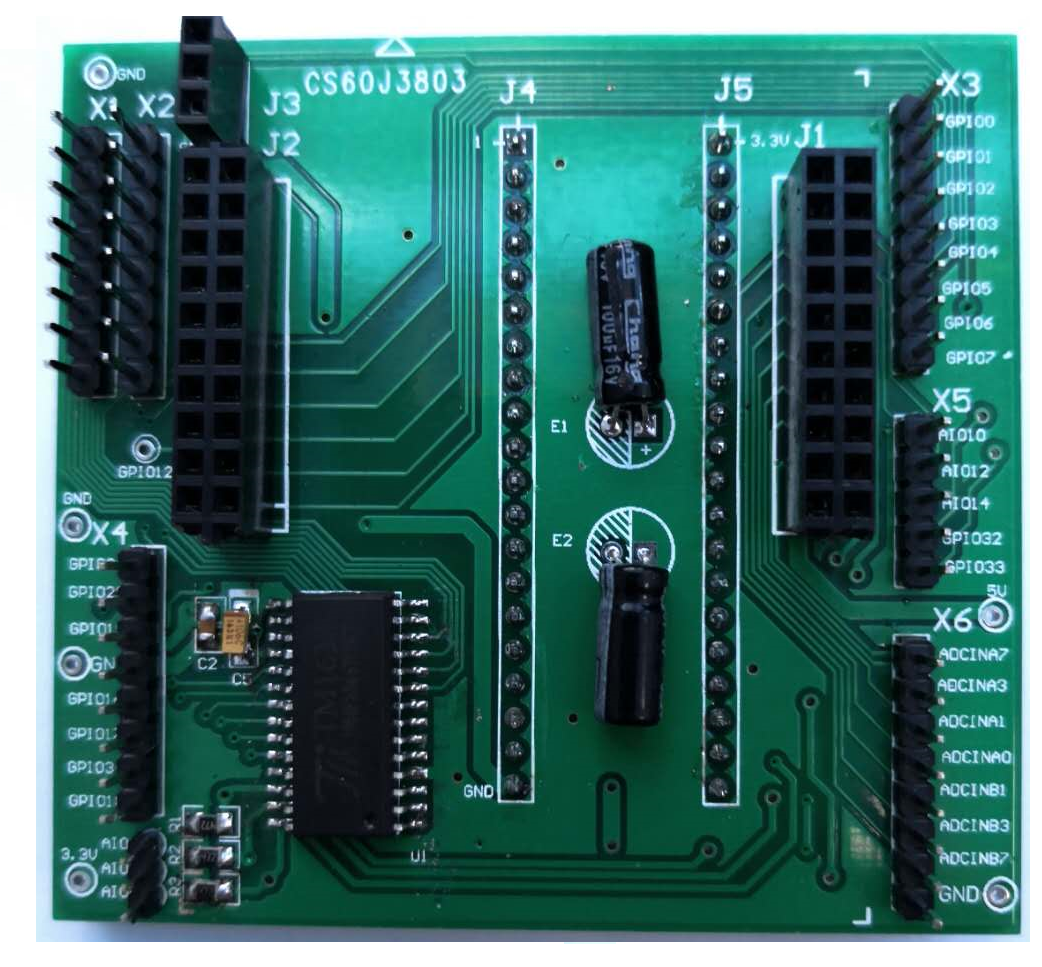
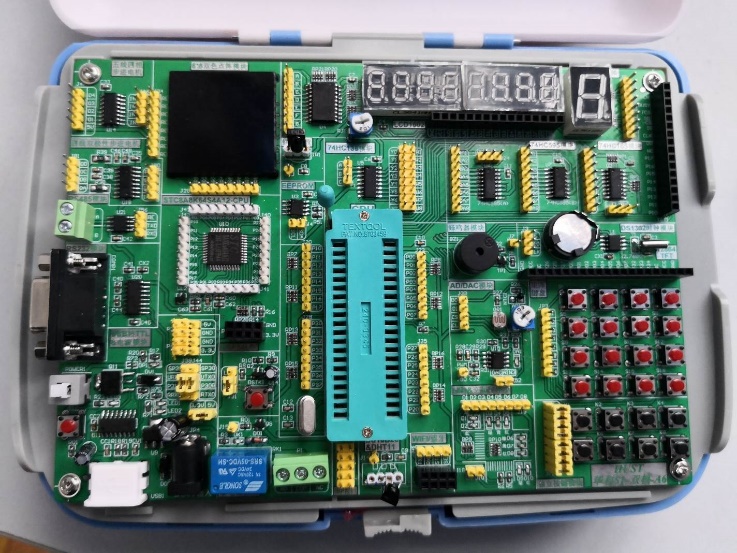
若要使用SCI串口时，需要将板上S4拨码开关打开，如下图所示：



图1.4 S4拨码开关细节图

将系统板连接电脑时要将S1拨码的3号拨码打开。否则连接会失败。

下图是实验套件图，由转接板和目标板组成，使用时将系统板排针插在转接板上，然后将转接板插在目标板对应的座子并锁紧。转接板和目标板之间根据实验的需要连接相应的排线。具体见图1.6~图1.7为转接板到最小系统板和实验目标板的接口原理图，图1.8~图1.10为本课程所要求实验内容在目标板上的原理图。

a)转接板 b)目标板

图1.5 实验套件

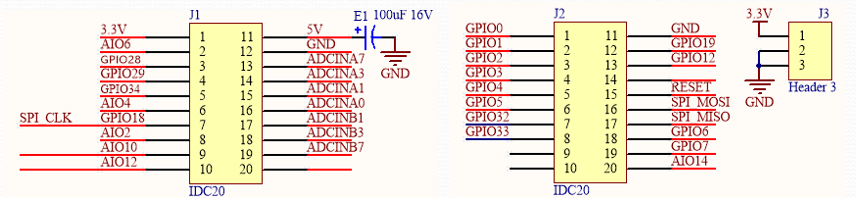


图1.6 转接板到最小系统板接口

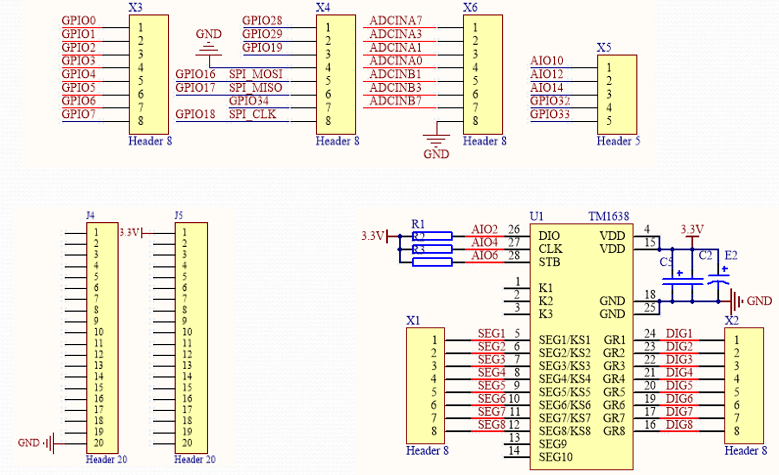


图1.7 转接板到目标板接口

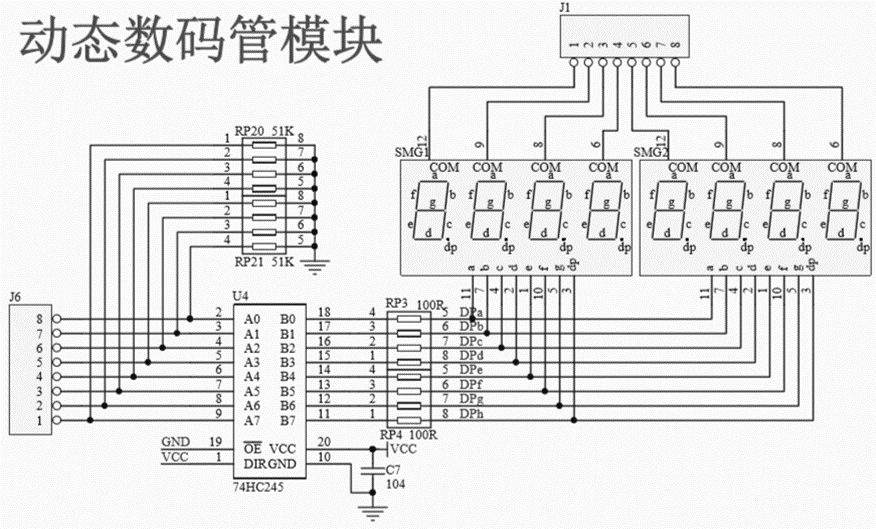


图1.8 目标板动态扫描LED电路

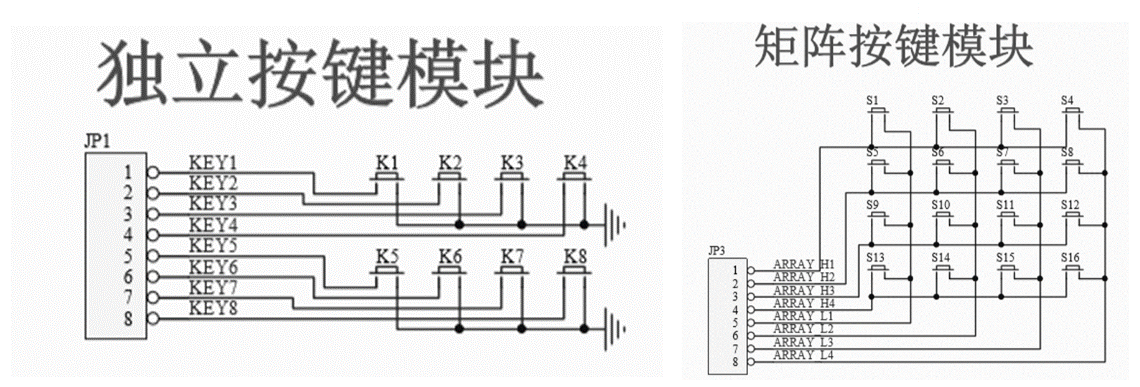


图1.9 目标板按键电路

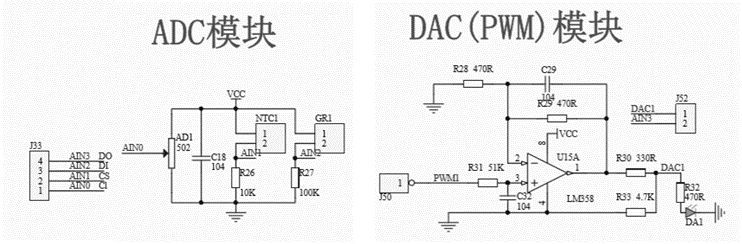


图1.10 目标板上ADC和DAC电路

实验目标板有丰富的模块，有兴趣的同学可开展多种实验研究。图1.11为实验目标板完整的图纸。

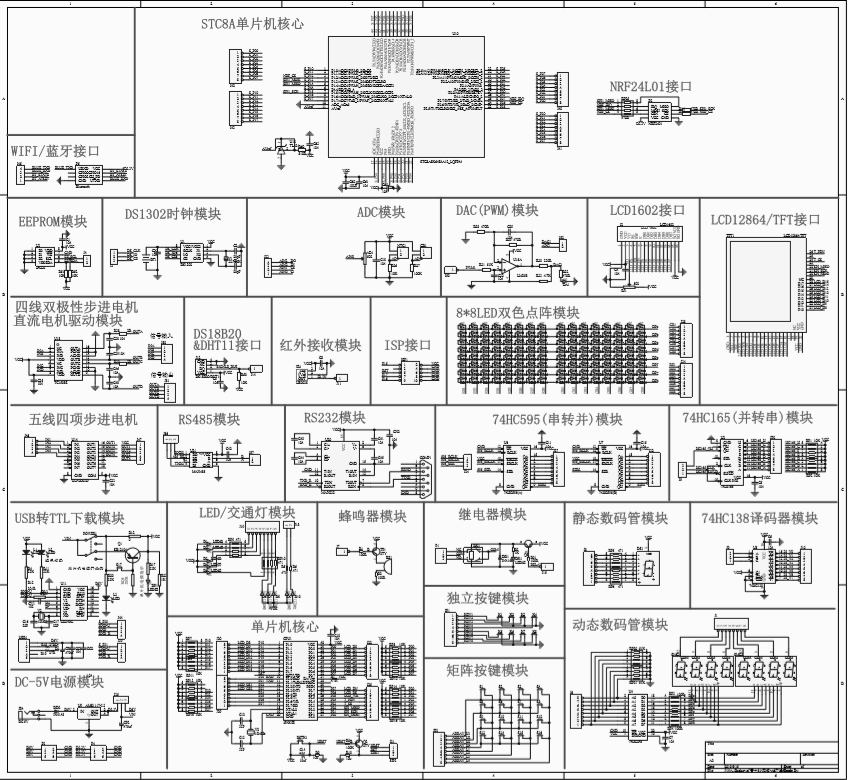


图1.11 实验目标板原理图

第二章 实验编译环境简介

**在使用CCS之前，请在TI官网上下载controlSUITE文件夹，并阅读提供的参考文献与datasheet。**

**CCS各版本软件下载地址：**

[**http://processors.wiki.ti.com/index.php/Download\_CCS**](http://processors.wiki.ti.com/index.php/Download_CCS)

（建议用CCSv5及以上版本）

一、新建一个工作目录

CCSv5启动时，会出现一个如图2.1的工作平台搜索（Workspace Launcher）界面，若先前建立了多个工作平台可通过点击下拉按钮进行选择；若需要建立一个新的工作目录可通过浏览按钮进行。这里在 f28027 的v129版本目录下建立了一个 Myproject目录，之后有关f28027的项目都放在这个目录中。

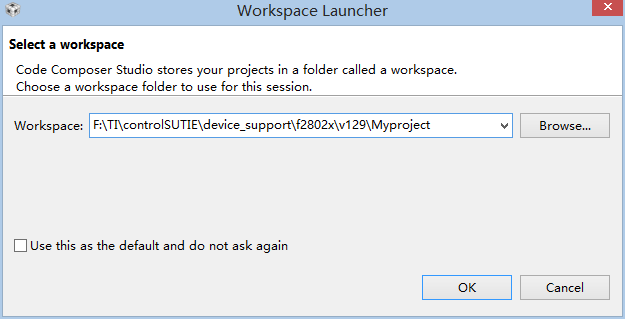


图2.1 工作平台搜索界面

二、构建项目

每当建立一个新的工作平台（或目录）之后，点击OK，便出现TI资源管理器界面。这里介绍几个常用的选项

* New Project 在上级目录下建立一个新项目
* Examples 打开已加载的示例
* Import Project 导入已有的项目
* Getting Started 打开在线视屏介绍

点击New Project ，出现如图2.2的CCS新项目构建平台1（New CCS Project），如图所示键入或选择相应参数：

1. Project name此框输入要建立的项目的名称，键入TimeLedBlink
2. Advanced settings输出类型选择，通过下拉菜单选择可执行文件Executable（默认值）
3. target左框为设定系统内的类型选择，默认为<select or type filter text>，这里选择2802x Piccolo右框为芯片选择，这里选择TMS320F28027

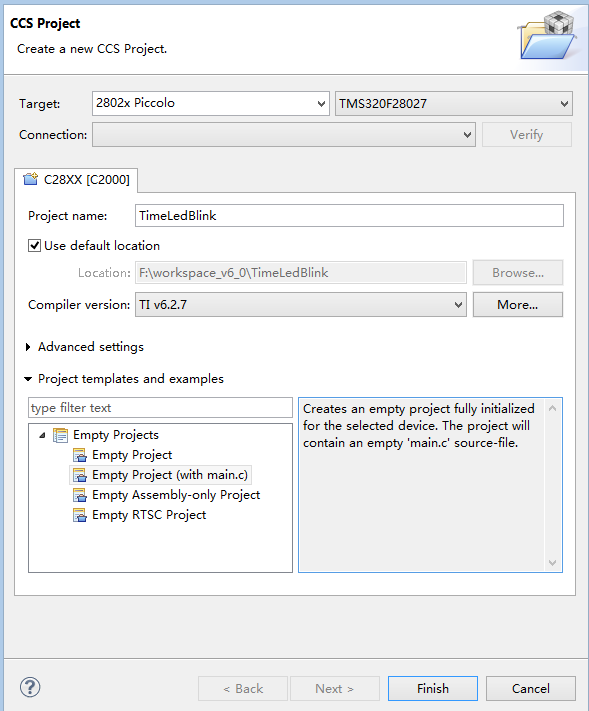


图2.2 CCS新项目构建平台

Advanced settings中参数设置 (可选缺省值)

1. Compiler version编译器版本选择，采用默认值
2. Linker command file链接器命令文件，选择28027\_RAM\_lnk.cmd
3. Runtime support library运行时需支持的库文件，可选择 rts2800\_ml.lib

此时，在Myproject目录中新添了一个TimeLedBlink项目，打开这个项目，可见编译器对这个新建项目的基本配置，参见图2.3。



图2.3 新建TimeLedBlink项目目录

新建TimeLedBlink项目以v129版本的timed\_led\_blink项目为原型，文件采用timed\_led\_blink项目的架构。图2.4 为timed\_led\_blink项目的目录，将该目录中的Example\_2802xLEDBlink.c文件复制到TimeLedBlink项目中，改名为TimeLedBlink.c，并删除main.c文件，之后形成的目录如图2.5所示。



图2.4 timed\_led\_blink项目目录

项目目录修改后，图1.4 CCS项目编辑平台也作如图2.6所示的相应修改。编译器会将目录中的源文件、头文件或其他相关文件加载到CCS编辑平台相应的项目中，这样省去了链接的麻烦，可直接编译下载。对单一的项目而言这种方法还是可行的，倘若要构建很多项目，把许多可共享的相同的头文件和源文件塞入每一个项目中，显然这种做法不便对文件进行管理，同时太多且低效地占用了存储空间。TI提倡的做法是：设置链接变量通过编译器将所需的头文件及共享源文件及库文件实现编译器层面的链接。

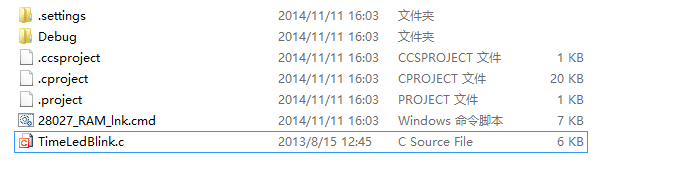


图2.5 修改过的TimeLedBlink项目目录

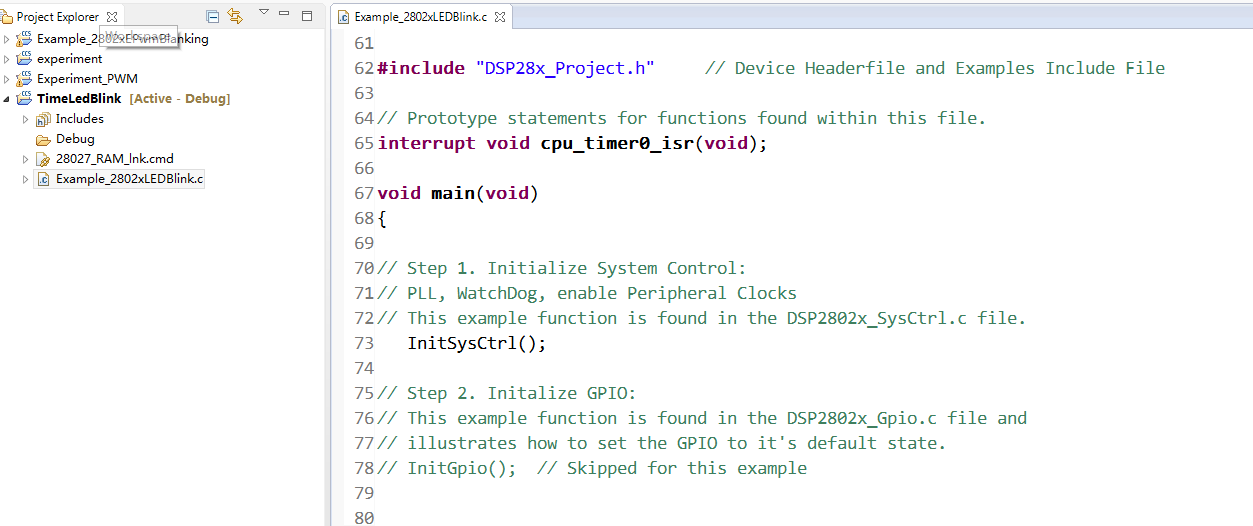


图2.6 项目目录修改后CCS编辑平台

注：新建工程具体步骤请参考,由于CCS引用的C文件.asm文件、cmd文件较多，在新建过程中需要花费较多时间。

三、导入已有的项目

由于新建的TimeLedBlink项目以timed\_led\_blink项目为蓝本，因此，先导入timed\_led\_blink项目，以便参照。

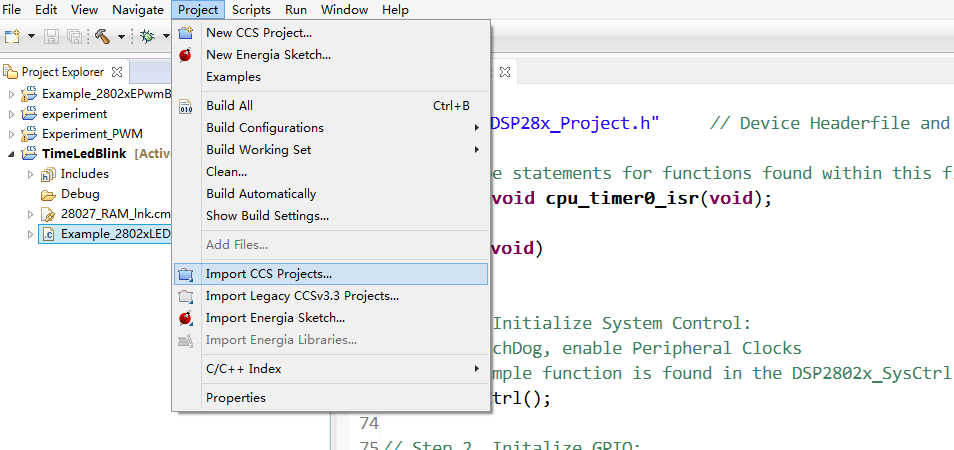


图2.7 导入已有的项目方法

* + 1. 进入“导入CCS Eclipse项目”界面

如图2.7，点击Project -> Import Existing CCS Eclipes Project，导入已有的CCS Eclipes项目，出现图2.8的界面；

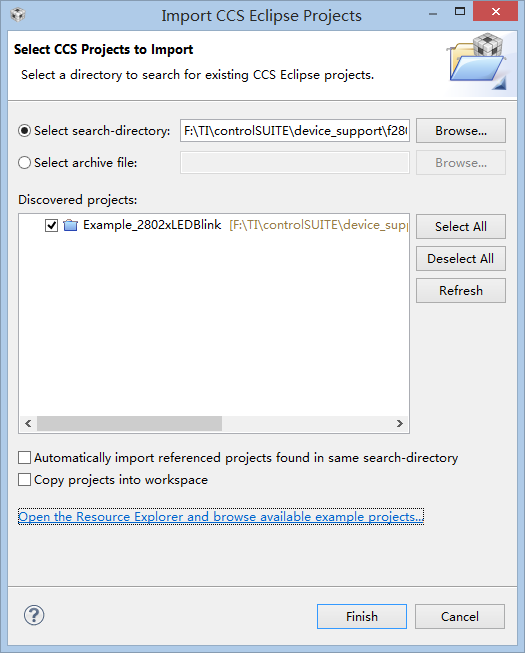


图2.8 选择已有的项目界面

* + 1. 导入已有的CCS Eclipse 项目

通过浏览按钮选择所需的timed\_led\_blink项目，再按按钮，之后，在项目资源管理器窗口会出现刚导入的项目，点击激活Example\_2802xLEDBlink项目，并双击该项目中的Example\_2802xLEDBlink.c文件，将该文件加载到编辑视窗，如图2.9所示。

以Eclipse为架构的CCSv5平台其项目资源管理器（Project Explorer）可涵盖TI现有的数字信号控制器（DSC）及其他的嵌入式系统（embedded system），比如MSP430和C2000的项目可以并存在这个窗口。在众多项目中被点击激活的项目称之为焦点项目，其尾部用[Active-Debug]标志。

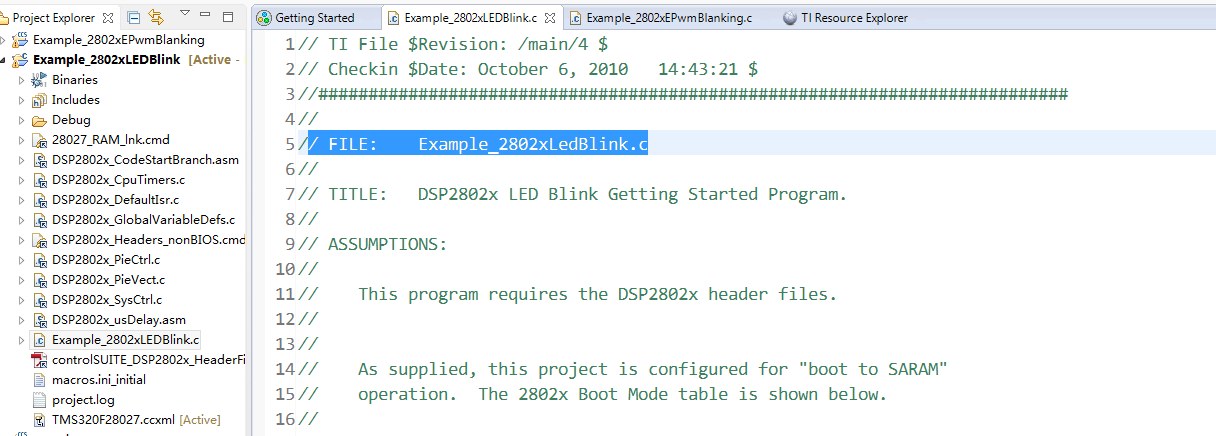


图2.9 打开激活项目的主文件

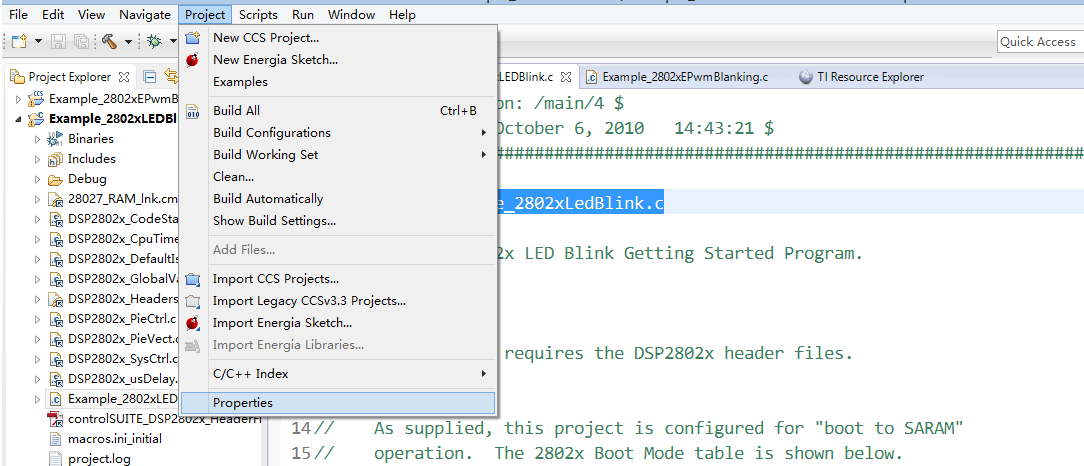


图2.10 激活项目的属性设置

* + 1. 项目属性设置

之后，如图2.10，点击 Project -> Properties，出现图2.11当前项目的属性设置界面，该图的左面是一个文字过滤器（type filter text），作用是搜索显示与输入的文字匹配的条目。刚打开时为默认状态，阴影条 General为属性的常规设置，如图2.11所示，可设置的条目为：

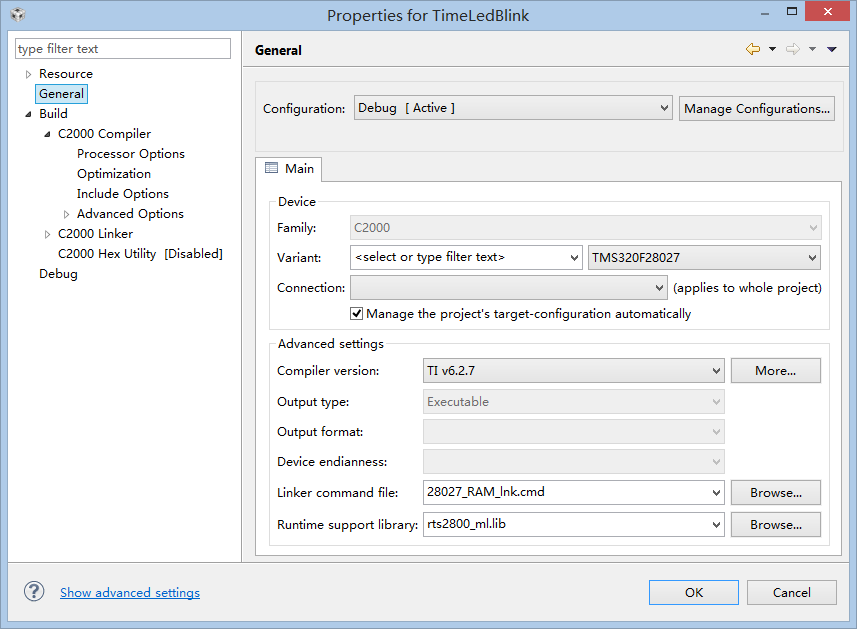


图2.11 当前项目的属性设置

1. Variant

该条目左框可采用默认值，不必修改，编译器会自动找到对应变量，右框通过下拉菜单选择“TMS320F28027”；

1. connection

选择连接的仿真器，通过下拉菜单选择

“Texas Instruments XDS100v1 USB Emulator”，使用XDS100v1仿真机

1. Compiler version

选择编译器版本。该框与下面的有效编译器推荐版本联合使用，根据推荐选择编译器版本；

1. Effective compiler version

推荐有效的编译器版本，以便上面选择合适的编译器版本；

1. Linker command file

选择链接器命令文件，这里选择28027\_RAM\_Ink.cmd

1. Runtime support library

选择运行时支持的库文件，这里选择<automatic>或rts2800\_ml.lib。之后，点击ok，就可以对导入的项目进行调试。

四、CCS常用按钮

4.1 编译界面常用按钮

图2.9为构建一个新的项目或者导入一个已经存在的项目时的编辑界面，此时，图标处在激活状态（底色为白色）。

表2.1 为CCSv5编辑状态下的常用按钮。

表2.1 CCSv5编译状态下的常用按钮

|  |  |  |
| --- | --- | --- |
| 图标 | 名称 | 说明 |
|  | Build ‘Debug’ | 根据实际使用的情况，该按钮对自建项目有效，其下拉菜单有两个可选项：  Debug 允许对自建项目进行编译；  Release 将自建项目释放成配置前的状态，此时编译将不能通过。 |
|  | Debug | 编译调试按钮，该按钮在项目激活时有效，其下拉菜单有历次编译记录，可激活选定的项目并进行编译。 |
|  | Search | 搜索按钮，其下拉菜单有3种搜索方式：  File Search 文件搜索；  c\c++ Search c\c++ 搜索；  Git Search Git 搜索 |
|  | Back to | 记录通过项目管理器（Project Explorer）导入的项目（或文件），按该钮时，将根据后进先出的顺序激活对应的项目 |
|  | LastEdit Location | 记录通过按钮依次导入的项目或文件，按该钮时，将根据后进先出的顺序激活对应的项目。当最后一个被激活后，该按钮关闭 |
|  | CCS Debug | 切换到当前的调试界面 |
|  | CCS Edit | 切换到当前的编辑界面 |

4.2 调试界面常用按钮

在编译状态下，按调试按钮，当编译链接全部通过后，可进入图2.12 所示的timed\_led\_blink项目调试界面。上部椭圆形红框内为调试状态下的常用按钮，下部椭圆形红框标注的阴影条左端有一个箭头表示程序将从这一行开始运行。

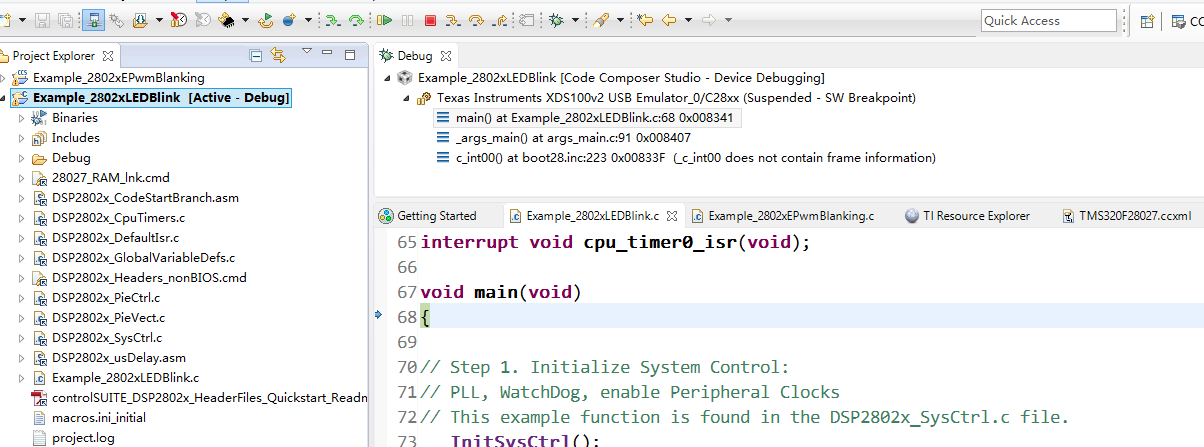


图2.12 timed\_led\_blink项目的调试界面

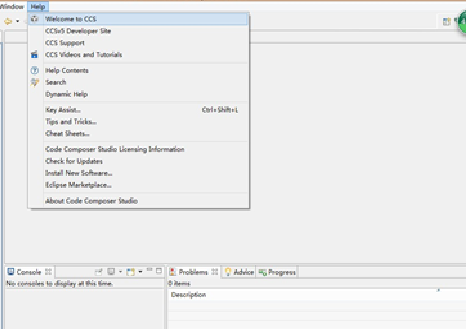
表2.2为调试界面的常用按钮。

表2.2 CCSv5调试界面的常用按钮

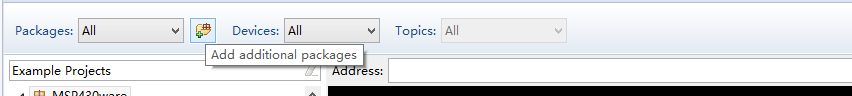
|  |  |  |
| --- | --- | --- |
| 图标 | 名称 | 说明 |
|  | Resume | 运行按钮。 |
|  | Suspend | 暂停按钮。 |
|  | Terminate | 结束按钮 |
|  | Step Into | 步入按钮 |
|  | Step Over | 步出按钮 |
|  | Assembly Step Into | 汇编步入按钮 |
|  | Assembly Step Over | 汇编步出按钮 |
|  | Step Return | 单步返回按钮 |
|  | Reset CPU or Rese Emulatort | 复位CPU或者复位仿真机按钮 |
|  | Restart | 重新开始 |
|  | Refresh | 刷新按钮 |
|  | Real Time Mode | 实时模式按钮 |
|  | Continuous Reflash | 连续刷新按钮 |
|  | Load | 下载按钮 |
|  | Show logical structrue | 逻辑结构显示按钮 |
|  | Collapse all | 全部折叠 |
|  | Open new View | 打开新的观察窗 |
|  | Pin to debug Context |  |
|  | Open Perspective | 打开透视窗 |
|  | CCS edit Perspective | CCS 编辑透视窗 |
|  | disconnect hardware | 断开硬件 |

五、导入controlSUITE

点击 help –> welcome to CCS



点击添加文件夹，选择controlSUITE的路径并添加，即可在example projects中直接导入例程，并有步骤提示。



快捷键说明：选中函数并按F3可以直接进入函数

第三章 实验内容

实验板可调试功能包含了DSP芯片的时钟定时器、中断、GPIO、ADC、SCI、ePWM和eCAP。

根据所学的内容，完成以下实验：

一、定时器和GPIO实验

1.1 实验要求

1. 通过定时器实现延时，编写跑马灯程序；
2. 对主板上的按键GPIO12进行采样，添加软件去抖功能；
3. 实现按键对跑马灯动作的选择，至少两种跑马灯动作；
4. 学习TM1638驱动程序，完成数码管的显示功能。

1.2 实验目的

1. 掌握GPIO端口的配置和操作方法；
2. 掌握定时器的配置和操作方法；
3. 掌握定时器中断的配置；
4. 学会使用GPIO实现TM1638的驱动，实现数码管的显示功能。

1.3 实验说明

定时器控制LED灯闪烁和数码管的显示。

点亮LED灯的实验称得上是开发板中的“Hello World！”，一般初学者都会首先尝试此实验。在这个实验中，可以分别采用查询和中断的方式来获知定时器计数器的溢出。然后操纵GPIO控制LED亮或是不亮。自己设计跑马灯的程序，以定时器实现延时，练习定时器和GPIO的使用。

LED的控制是通过GPIO输出实现的，按键的采样则是通过GPIO输入实现的。通过读取GPIO12端口的状态，判断按键是否按下。机械按键存在抖动的问题，因此添加软件去抖程序实现更稳定的采样。

底板上有LED数码管显示模块及其驱动电路，数码管的驱动是通过TM1638实现的，TM1638是LED驱动控制专用电路，能实现LED驱动、键盘扫描等功能。其控制输入引脚只有三个，因此可以大大节约控制芯片的GPIO引脚的占用。通过一块TM1638数码管驱动芯片，可以实现8位数码管的驱动。TM1638的驱动程序是通过操作GPIO引脚实现的，TM1638的底层驱动程序已由例程给出，可参考例程学会TM1638的驱动方法。LED数码管的显示是对GPIO的巩固与加强，通过对TM1638的驱动，加深对GPIO功能的理解。

注：四个LED的GPIO端口分别为GPIO0、GPIO1、GPIO2、GPIO3。

考察的知识包含：时钟的配置、定时器配置、中断配置、GPIO配置。可参看实验板的配套程序(Clock电子钟的设计)

二、PWM和eCAP实验

2.1 实验要求

1. 输出带死区的互补PWM波形；
2. 通过eCAP单元捕获其中一路PWM波形并获取其周期和占空比。

2.2 实验目的

1. 掌握PWM的周期、死区电平和死区时间的设定方法；
2. 掌握PWM计数周期中断和比较值匹配中断的配置方法。
3. 掌握eCAP单元的操作方法及其中断的配置；
4. 掌握eCAP单元时间测量和占空比测量的方法。

2.3 实验说明

输出PWM波形，并通过eCAP捕获PWM脉冲。

配置EPWMxA和EPWMxB输出带死区的互补PWM波形，输出端口在转接板上的X3端子上（排针端子），具体芯片引脚参见第一章中的引脚定义。

在输出PWM的基础上，将其中一路PWM输出引脚接到eCAP捕获引脚上，通过捕获单元获取PWM的周期和占空比信息。

三、ADC实验

3.1 实验要求

1. 通过ADC读取滑动变阻器(目标板上AD1)中间抽头上的电压值，并将结果在数码管上显示。
2. 通过ADC读取目标板上NTC1两端的电压差值，并将结果在数码管上显示。

3.2 实验目的

1. 掌握ADC的配置和操作方法；
2. 掌握使用PWM触发AD采样的方法。

3.3 实验说明

本实验在前面实验的基础上，添加了ADC的使用以及ADC中断的配置。通过PWM触发AD采样（也可尝试其他触发AD采样的方法，多方面了解和掌握ADC的操作），并将采样结果换算成实际的电压结果，在数码管上显示出来。

数码管上显示AD采样结果的时候，需要将数据的每一位输出到数码管的相应的位置上，还涉及到小数点位置的显示，因此，建议编写数码管上层驱动程序，实现浮点数的显示功能。

本实验中，ADC的转换触发采用PWM控制，复习了上一实验中对PWM的配置。

四、综合实验

4.1 实验要求

1. 通过ADC获取目标板上滑动变阻器AD1中间抽头的电压，并实时显示在上排数码管上；从目标板的J33的第一脚引到DSP的模拟输入端。
2. 将（1）中得到的采样结果转换成PWM的占空比输出到目标板上J50，将PWM信号低通滤波得到与PWM占空比成正比关系的电压，相当于DAC。目标板的J52的第一脚为DAC的输出电压值。
3. 通过另一路ADC对（2）中DAC得到的电压进行采样，并实时显示在下排数码管上。可以J52上的DAC1信号直接接到DSP的模拟输入端；也可以将J52的引脚短接，从目标板J33的第四脚引到DSP的模拟输入端。
4. 通过对采样电阻大小的选取和PWM占空比的控制，使得两次采样的结果尽可能相等。可以使用闭环控制方法，将第二次的采样结果作为反馈量，第一次的采样结果作为给定参考量，PWM的占空比作为通过PI控制等控制方法使反馈量跟踪到给定量，也可以设计其他控制策略。

4.2 实验目的

本实验综合所学的DSP的知识和前面做过的实验，设计一个涉及到多个外设的嵌入式系统，要求能够熟练配置系统时钟和各个外设。

考察对芯片多个外设的配置的掌握情况，考察学生的综合设计和嵌入式开发的能力。

附录：

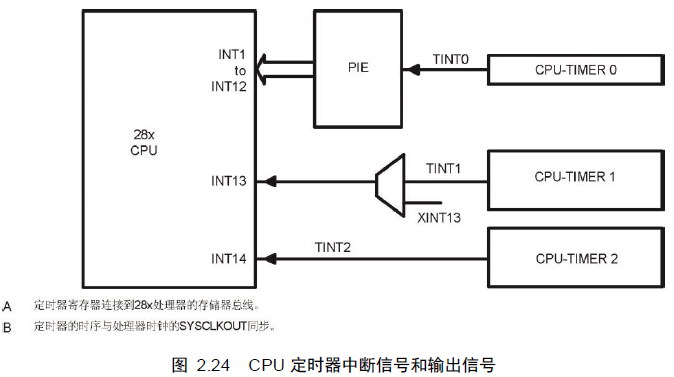
以下给出部分外设使用的示例实验，仅供参考。

一、定时器的使用

1.1 CPU定时器概述

F28x有3个32位的CPU定时器（TIMER0/1/2），这3个定时器具有完全相同的控制结构，其内部结构如图所示。其中，CPU定时器1及CPU定时器2被保留用作DSP BIOS或者其他的实时操作系统（RTOS），CPU定时器0可供用户使用。





1.2 CPU定时器寄存器简介

1. 计数寄存器（TIMERxTIM）（其中，x=0，1，2，下同）。

32位计数寄存器（TIMH:TIM）中，TIM为计数寄存器低16位，TIMH为计数寄存器高16位。每一个定时器时钟周期（TDDRH:TDDR+1）/ SYSCLKOUT，TIM减1，其中，（TDDRH:TDDR）是定时器预定标分频值。当（TIMH:TIM）减到0时，（TIMH:TIM）重装定时器周期值（PRDH:PRD），同时产生定时器中断。

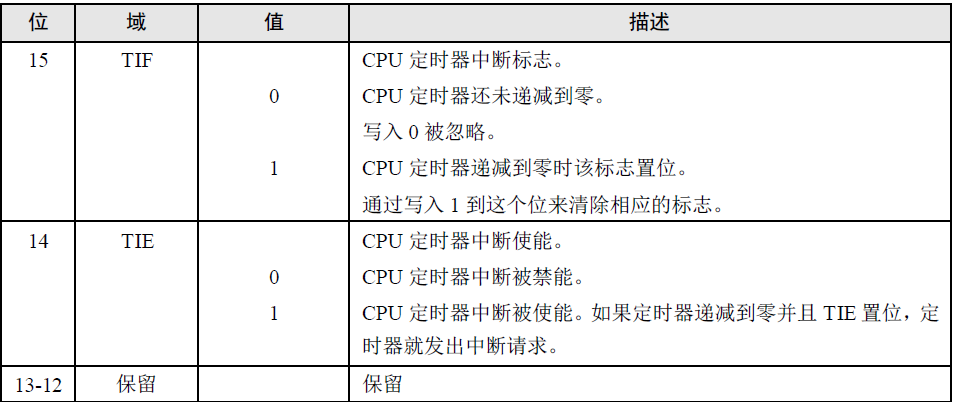
1. 周期寄存器（TIMERxPRD）

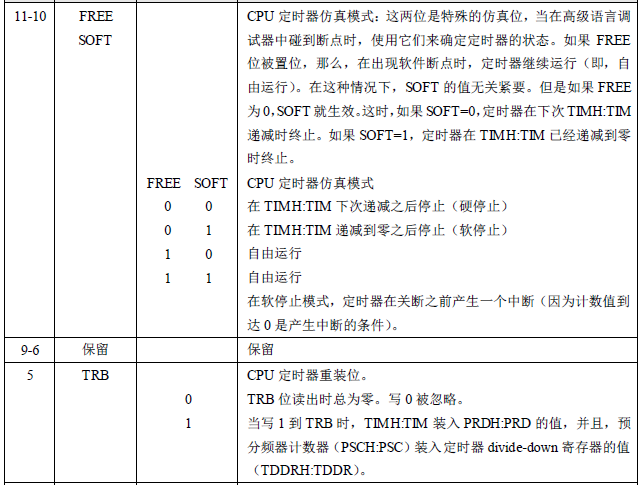
32位周期寄存器（PRDH:PRD）中，PRD为周期寄存器低16位，PRDH为周期寄存器高16位。当（TIMH:TIM）减到0时，在下一个定时器输入时钟周期开始时，（TIMH:TIM）将重装（PRDH:PRD）的周期值；或者当定时器控制寄存器（TCR）的TRB置位时，（PRDH:PRD）的周期值也装入（TIMH:TIM）。

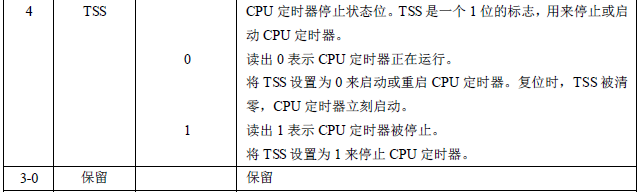
1. 定时器分频寄存器（TDDRH:TDDR）

16位分频寄存器（TDDRH:TDDR）中，TDDR为分频寄存器低8位，TDDRH为分频寄存器高8位。每过一个定时器时钟周期，定时器计数器寄存器（TIMH:TIM）减1。当预定标器计数器（PSCH:PSC）减到0，一个系统时钟周期后，（TDDRH:TDDR）的值重装（PSCH:PSC），同时（TIMH:TIM）减1。无论何时，用软件置定时器重装位（TRB）为1，（TDDRH:TDDR）重装（PSCH:PSC）。

1. 控制寄存器（TIMERxTCR）TCR是一个16位的寄存器。







1.3 定时器的周期

当TSS(TIMERxTCR[4])=0时，CPU定时器以系统时钟SYSCLKOUT作为时钟源，此时，预定标计数器（PSCH:PSC）以装载的分频值（TDDRH:TDDR）为基数，对每一个时钟源周期开始减计数。预定标计数器（PSCH:PSC）从分频值（TDDRH:TDDR）减至0时，称为一个定时器周期（定时器时钟），其表达式如式（2.1）所示。

 （2.1）

定时器周期的倒数为定时器时钟频率，其表达式如式（2.2）所示。

 （2.2）

其中：0 （TDDRH:TDDR） 65535

预定标计数器（PSCH:PSC）减至0时，重新装入定时器分频值（TDDRH:TDDR）,且计数器（TIMH:TIM）以装载的周期值（PRDH:PRD）为基数，对每一个定时器时钟周期减1计数；当计数器（TIMH:TIM）减至0时，定时器周期值（PRDH:PRD）重新装入计数器（TIMH:TIM）并请求一个中断。

1.4 定时器的使用

定时器的主要功能是产生定时中断，以Example\_2802xLEDBlink为例说明定时器功能。

**voidmain**(**void**)

{

**InitSysCtrl**();//初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，//初始化外设时钟，低速外,4分频

DINT;//关总中断

**InitPieCtrl**();//初始化PIE控制寄存器。默认状态为禁止所有PIE中断并清除所//有标志

IER = 0x0000;//

IFR = 0x0000;//禁用 CPU 中断并清除所有 CPU 中断标志：

**InitPieVectTable**();//初始化中断向量表

EALLOW; //允许写入受保护的寄存器

PieVectTable.TINT0 = &cpu\_timer0\_isr;//将定时器中断TINT0的中断服务地址////指向cpu\_timer0\_isr这个函数

EDIS; //禁止写入

**InitCpuTimers**();//初始化定时器配置

**ConfigCpuTimer**(&CpuTimer0, 60, 500000);//配置CPU计时器0中断每500毫秒//为单位:60 MHz CPU 频率,周期T=60\*500000/60M =0.5S

CpuTimer0Regs.TCR.all = 0x4001;//控制寄存器:启动定时器

EALLOW;

GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;//IO34用作普通IO

GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;//IO34用作输出

EDIS;

IER |= M\_INT1;//使能中断INT1

PieCtrlRegs.PIEIER1.bit.INTx7 = 1;//使能定时器中断

EINT; // Enable Global interrupt INTM

ERTM; // Enable Global realtime interrupt DBGM

**for**(;;);

}

**interruptvoidcpu\_timer0\_isr**(**void**)

{

CpuTimer0.InterruptCount++;

GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;//IO34数据取反

PieCtrlRegs.PIEACK.all = PIEACK\_GROUP1;

}

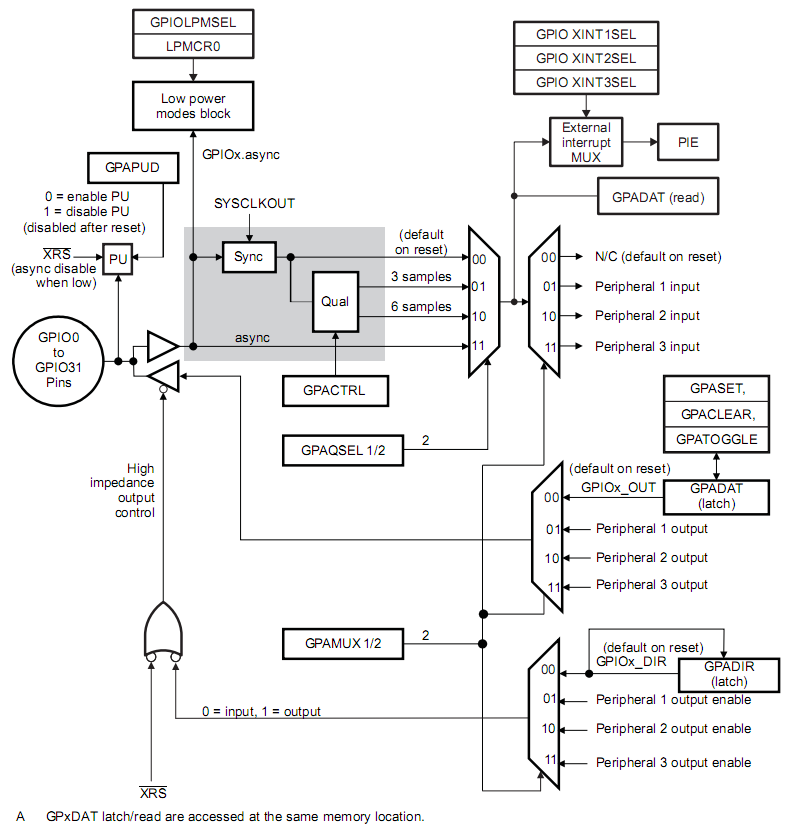
实验现象：LED灯闪烁

参考例程：

二、GPIO的使用

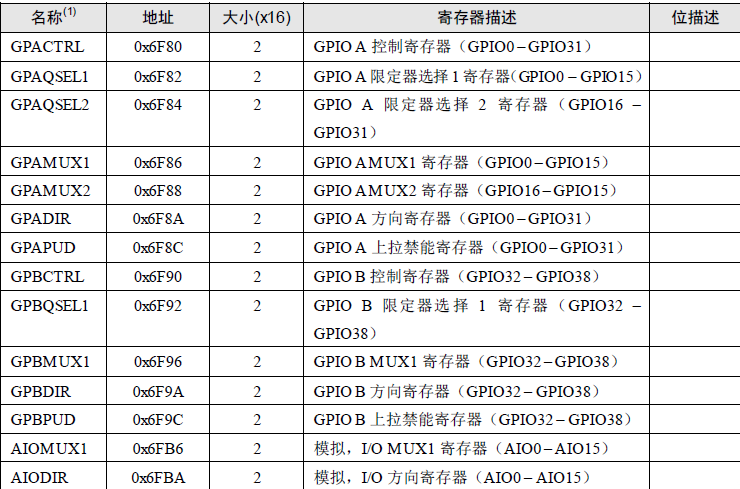
2.1 多功能复用GPIO介绍

除了单个引脚位-I/O功能之外，每个GPIO使能的引脚还可以复用为多达3个独立的外设信号。有3个I/O端口。Port A由GPIO0 –GPIO31组成，Port B由GPIO32 –GPIO38组成。模拟端口包括AIO0 –AIO15。下图显示了GPIO模块操作的基本模式。注意：JTAG引脚也提供GPIO功能。



2.2 配置GPIO寄存器

1. 控制寄存器



1. 数据寄存器



1. 功能介绍

GPxMUX寄存器：多路复用配置

当GpioCtrlRegs.GPAMUX2.bit.GPIOX = 0; 相应引脚配置为IO功能

当GpioCtrlRegs.GPAMUX2.bit.GPIOX = 1;相应引脚配置为外设功能

GPxDIR寄存器:方向控制

当GpioCtrlRegs.GPADIR.bit. GPIOX = 0; 引脚配置为输入

当GpioCtrlRegs.GPADIR.bit. GPIOX = 1; 引脚配置为输出

GPxDAT寄存器：数据寄存器

当GpioDataRegs.GPADAT.bit.GPIOX = 0;在引脚设置为输出的条件下，把引脚拉低为低电平

当GpioDataRegs.GPADAT.bit.GPIOX = 1;在引脚设置为输出的条件下，把引脚拉低为低电平

GPxSET寄存器：置位寄存器，只能写1，当引脚为输出时写1将引脚拉高为高电平

GPxCLEAR寄存器：清除寄存器，只能写1，当引脚为输出时写1将引脚拉低为低电平

GPxTOGGLE寄存器：切换寄存器，只能写1，引脚为输出时将引脚拉为反向电平

2.3 GPIO的使用

下边是将GPIO29设置为输出的指令：

EALLOW; // 允许访问受保护的寄存器

GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 0; // 设置GPIO29 为GPIO

GpioCtrlRegs.GPADIR.bit.GPIO29 = 1; // 设置GPIO29 为输出

EDIS; // 禁止访问受保护的寄存器

指令的作用如注释。之后，就可通过以下指令在GPIO29引脚上输出高低电平。

GpioDataRegs.GPASET.bit.GPIO29 = 1; // 设置GPIO029为高电平

GpioDataRegs.GPACLEAR.bit.GPIO29= 1; // 设置GPIO029为低电平

GpioDataRegs.GPATOGGLE.bit.GPIO29= 1; // 切换 GPIO29 电平

GpioDataRegs.GPADAT.bit.GPIO29 = 1; // 设置GPIO029为高电平

注意：上面3条设置0无效，下面一条用作引脚信号的采集为好。

下边是将GPIO29设置为输出的指令：

EALLOW; // 允许访问受保护的寄存器

GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0; // 设置GPIO7 为GPIO

GpioCtrlRegs.GPADIR.bit.GPIO7 = 1; // 设置GPIO7 为输出

GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 0; // 设置GPIO29 为GPIO

GpioCtrlRegs.GPADIR.bit.GPIO29 = 0; // 设置GPIO29 为输入

EDIS; // 禁止访问受保护的寄存器

指令的作用如注释。之后，就可通过以下指令采集GPIO29的信号。

#define Col\_GPIO29 GpioDataRegs.GPADAT.bit.GPIO29

// 在文件头部定义

...

if(Col\_GPIO29==0)

GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;

// 若采集到 GPIO29 为低电平则切换GPIO7的电平

参考例程：

LED灯对应引脚为： GPIO0、1、2、3

可以通过改变相应IO引脚状态通过LED观察

**跑马灯例程**：

**voidmain**(**void**)

{

InitSysCtrl();

DINT;

InitPieCtrl();

IER = 0x0000;

IFR = 0x0000;

InitPieVectTable();

EALLOW;

GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;

GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0;

GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;

GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;

GpioCtrlRegs.GPADIR.bit.GPIO0 = 1;

GpioCtrlRegs.GPADIR.bit.GPIO1 = 1;

GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;

GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;

EDIS;

**while** (1) {

Gpio\_horse\_R\_L();

//delay\_loop();

}

}

**voidGpio\_horse\_R\_L**(**void**)

{

**int** i=0;

**longint** j=0;

**int** index=0x01;

**for** (i = 0; i < 4; i++)

{

GpioDataRegs.GPACLEAR.all |= index;

GpioDataRegs.GPASET.all |= 0x0F-index;

**if**(i==3)

index=0x01;

**else**

index<<=1;

**for**(j=0;j<1000000;j++);

}

}

注：如果新建工程，除了main函数以外，需要添加相应的C文件和h文件，如果是直接修改例程，只需要将main函数进行修改即可。

三、ePWM

PWM模块是28027中重要模块之一，目的在于输出精确可靠的PWM波形。本次实验主要目的是产生并用示波器观察PWM波形。

* 1. ePWM模块概述

ePWM模块描述了一条由EPWMxA和EPWMxB两个PWM输出组成的完整的PWM通道。

每个ePWM模块包含以下功能部件和特性：

1. 专用的16位时基计数器，可控制自己那对PWM输出的频率或周期
2. 两个PWM输出（EPWMxA和EPWMxB），它们可以配置成：
3. 两个独立的、单边沿操作的PWM输出
4. 两个独立的、双边沿对称操作的PWM输出
5. 一个独立的、双边沿非对称操作的PWM输出
6. 通过软件异步控制PWM信号
7. 可编程的相位控制，配置相对于其它ePWM模块的相位（领先或滞后多少）
8. 周期性地硬件锁定（同步）相位关系
9. 死区发生器，带独立的上升沿延迟和下降沿延迟控制
10. 触发区配置可编程，可配置成在出现故障时周期性触发（周期性触发）或单次触发（单次触发）
11. 产生故障触发条件后（trip condition）可将PWM输出强制变为高电平、低电平或高阻状态
12. 比较器模块输出和触发区输入可以产生事件、滤波（filtered）事件或故障触发条件
13. 所有事件都可以触发CPU中断和ADC开始转换（SOC）
14. 事件预分频因子（prescaling）可编程，使得中断的CPU开销最少
15. PWM被高频载波信号斩波，这对于脉冲变压器门极驱动非常有用

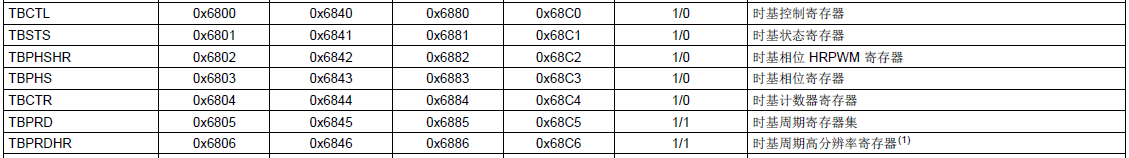
3.2 具体功能模块说明

3.2.1 时基（TB）模块

每个ePWM模块都有自己的时基子模块，它决定ePWM模块所有事件的时序。内置的同步逻辑电路使得几个ePWM模块的时基可以当作一个系统一起工作。图10.4描述了时基模块在ePWM中的位置。

时基模块主要是配置PWM的频率和周期，管理几个PWM模块的同步关系，设置PWM计数方式。

其中重要的寄存器包括以下几个，在配置时需要根据所需波形而配置。



3.2.2 计数器-比较（CC）子模块

计数器-比较子模块用于输入时基计数器的值。这个值被连续拿来与计数器-比较A（CMPA）和计数器-比较B（CMPB）的值进行比较。当时基计数器等于其中一个比较寄存器的值时，计数器-比较单元会产生一个适当的事件。

计数器-比较：

使用CMPA和CMPB寄存器根据可编程的时间戳（time stamp）产生事件

— CTR = CMPA：时基计数器等于计数器-比较A寄存器（TBCTR = CMPA）

— CTR = CMPB：时基计数器等于计数器-比较B寄存器（TBCTR = CMPB）

如果正确配置动作限定器子模块，可用于控制PWM占空比

shadow寄存器保存新的比较值，以阻止在有效的PWM周期期间出现讹误或谬误。

3.2.3 动作限定器（AQ）子模块

1. 动作限定器子模块的用途

动作限定器子模块负责以下事情：

根据以下事件限定并产生动作（置位、清零、切换（toggle））

CTR = PRD：时基计数器的计数值等于周期（TBCTR = TBPRD）

CTR = Zero：时基计数器的计数值等于零（TBCTR = 0x0000）

CTR = CMPA：时基计数器的计数值等于计数器-比较A寄存器（TBCTR = CMPA）

CTR = CMPB：时基计数器的计数值等于计数器-比较B寄存器（TBCTR = CMPB）

在这些事件同时发生时管理优先级

在时基计数器递增和递减时单独控制这些事件

1. 重要寄存器说明



动作限定器子模块控制在发生某一特定事件时EPWMxA和EPWMxB两个输出信号的行为。此外，动作限定器子模块的事件输入还受计数器方向（递增还是递减）的限制。不管在递增阶段还是递减阶段都可以单独控制输出上的动作。

施加到EPWMxA和EPWMxB输出信号上的动作可能是：

①置1（高电平）：将EPWMxA或EPWMxB输出信号设置成高电平

②清零（低电平）：将EPWMxA或EPWMxB输出信号设为低电平

③高低电平切换（Toggle）：如果当前EPWMxA或EPWMxB被拉高，则将输出拉低。如果当前EPWMxA或EPWMxB被拉低，则将输出拉高。

④Do Nothing（不采取任何动作）：EPWMxA或EPWMxB输出信号保持为当前设置值。尽管“Do Nothing”选项使得事件不可以在EPWMxA和EPWMxB输出信号上采取动作，但该事件仍然可以触发中断和ADC开始转换。

1. 波形配置方法

关于如何产生不对称波形，通过配置CMPA和CMPB以及计数方式来得到对称和不对称波形。具体方式如下：

①使用“先递增后递减”计数模式产生一个对称的**PWM**：

如果在计数值为0时装载CMPA/CMPB，那么使用大于等于1的CMPA/CMPB值如果在计数值为周期时装载CMPA/CMPB，那么使用小于等于“TBPRD-1”的CMPA/CMPB值这就意味着，一个PWM周期中会一直有一个长度至少为一个TBCLK周期的脉冲，当它非常短时系统可将其忽略。

②使用递减计数模式产生一个非对称的**PWM**：要得到占空比为50%～0%的非对称PWM，需要使用以下配置：在计数值等于周期时装载CMPA/CMPB。用周期动作11（period action）将PWM清零，用compare-up动作将PWM置位。将比较值从0调节到TBPRD，实现50%～0%的PWM占空比

③使用递增模式产生一个非对称的**PWM**：要得到占空比为0%～100%的非对称PWM，需要使用以下配置：在计数值等于TBPRD时装载CMPA/CMPB。用0动作（Zero action）将PWM置位，用比较动作将PWM清零。将比较值从0调节到“TBPRD+1”，实现0%～100%的占空比

3.2.4 死区发生器（DB）子模块

1. 死区子模块的用途

“动作限定器（AQ）子模块”这一章讨论了通过“使用ePWM模块的CMPA和CMPB资源完全控制边沿的位置”这一方法如何产生所需的死区。如果需要更多典型的、带优先级控制且基于边沿延迟的死区，那就应该使用这里介绍的死区子模块。

死区子模块的主要功能如下：

产生合适的信号对（EPWMxA和EPWMxB），可从单个EPWMxA输入

将信号对编程为：

* 高电平有效（AH）
* 低电平有效（AL）
* 高电平有效互补（AHC）
* 低电平有效互补（ALC）

向上升沿添加可编程延迟（RED）

向下降沿添加可编程延迟（FED）

在信号通道可完全旁路

1. 重要寄存器配置



DBCTL[IN\_MODE]:

00:EPWMxA in是下降沿延迟和上升沿延迟的源

01:EPWMxA in是下降沿延迟的源EPWMxB in是上升沿延迟的源

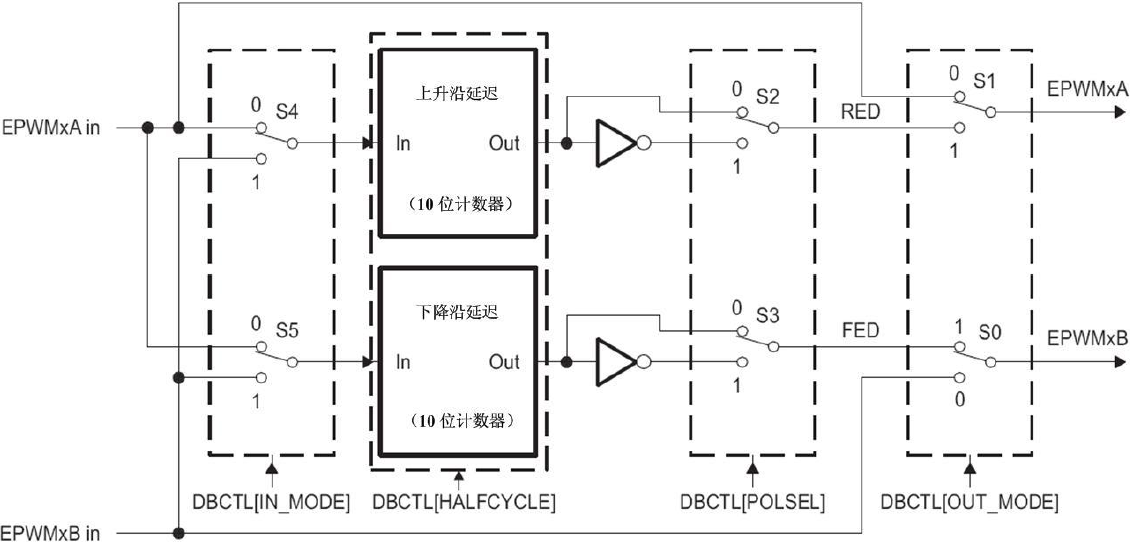
10:EPWMxA in是上升沿延迟的源EPWMxB in是下降沿延迟的源

11:EPWMxB in是下降沿延迟和上升沿延迟的源

若需要产生死区，一般选择00或者11.

输出模式控制：输出模式由DBCTL[OUT\_MODE]位配置。这些位决定输入信号是添加了下降沿延迟，还是上升沿延迟或者是下降沿与上升沿两种延迟，亦或两种延迟都没有添加。

极性控制（DBCTL[POLSEL]:让用户可以确定上升沿延迟信号和/或下降沿延迟信号在从死区子模块发送出去之前是否被反相（invert）。



控制和监控触发区子模块



**实验例程：以死区产生为例**

**打开例程，在void InitEPwmXExample()中有关于PWM寄存器的配置说明，如下：**

**voidInitEPwm1Example**()

{

/\*\*\*配置时基\*\*\*/

EPwm1Regs.TBPRD = 6000; // Set timer period

EPwm1Regs.TBPHS.half.TBPHS = 0x0000; // Phase is 0

EPwm1Regs.TBCTR = 0x0000; // Clear counter

// Setup TBCLK

EPwm1Regs.TBCTL.bit.CTRMODE = TB\_COUNT\_UPDOWN; // Count up

EPwm1Regs.TBCTL.bit.PHSEN = TB\_DISABLE; // Disable phase loading

EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB\_DIV4; // Clock ratio to SYSCLKOUT

EPwm1Regs.TBCTL.bit.CLKDIV = TB\_DIV4;

/\*\*\*比较寄存器\*\*\*/

EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC\_SHADOW; // Load registers every ZERO

EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC\_SHADOW;

EPwm1Regs.CMPCTL.bit.LOADAMODE = CC\_CTR\_ZERO;

EPwm1Regs.CMPCTL.bit.LOADBMODE = CC\_CTR\_ZERO;

// Setup compare

EPwm1Regs.CMPA.half.CMPA = 3000;

/\*\*\*动作限定寄存器\*\*\*/

EPwm1Regs.AQCTLA.bit.CAU = AQ\_SET; // Set PWM1A on Zero

EPwm1Regs.AQCTLA.bit.CAD = AQ\_CLEAR;

EPwm1Regs.AQCTLB.bit.CAU = AQ\_CLEAR; // Set PWM1A on Zero

EPwm1Regs.AQCTLB.bit.CAD = AQ\_SET;

// Active Low PWMs - Setup Deadband

EPwm1Regs.DBCTL.bit.OUT\_MODE = DB\_FULL\_ENABLE; //完全使能deadband

EPwm1Regs.DBCTL.bit.POLSEL = 1;//DB\_ACTV\_LO;

EPwm1Regs.DBCTL.bit.IN\_MODE = DBA\_ALL;

//00 EPWMxA In（来自动作限定器）是下降沿延迟和上升沿延迟的源

//01 EPWMxB In（来自动作限定器）是上升沿延迟信号的源

// EPWMxA In（来自动作限定器）是下降沿延迟信号的源

//10 EPWMxA In（来自动作限定器）是上升沿延迟信号的源

// EPWMxB In（来自动作限定器）是下降沿延迟信号的源

//11 EPWMxB In（来自动作限定器）是上升沿延迟和下降沿延迟信号的源

EPwm1Regs.DBRED = EPWM1\_MIN\_DB;//延迟计数器。频率可选为时钟频率的2倍或1倍

EPwm1Regs.DBFED = EPWM1\_MIN\_DB;

EPwm1\_DB\_Direction = DB\_UP;

/\*\*\*事件触发器，配置中断，soc\*\*\*/

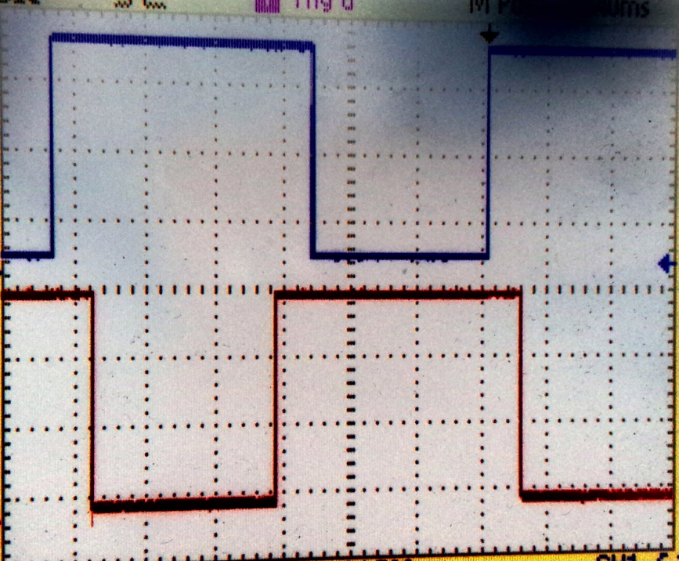
EPwm1Regs.ETSEL.bit.INTSEL = ET\_CTR\_ZERO; // Select INT on Zero event

EPwm1Regs.ETSEL.bit.INTEN = 1; // Enable INT

EPwm1Regs.ETPS.bit.INTPRD = ET\_3RD; // 事件预分频

}

最后结果用示波器观察如下:



四、ADC

4.1 ADC模块概述

两个采样/保持电路由多达16个模拟输入通道驱动。其他型号器件上可用的模拟输入通道数目可参阅相应的数据手册。这个转换器可以用一个内部帶隙基准电压3作为参考源实现真实电压转换，也可以用一对外部基准电压（VREFHI/LO）作为参考源实现比例（ratiometric）转换。

两个8通道模块能够自动排序，每个模块可以通过多路选择器(MUX)选择8通道中的任何一个通道。在级联模式下，自动排序器将变成16通道。对于每个通道而言，一旦ADC转换完成，将会把转换结果存储到结果寄存器(ADCRESUILT)中。自动排序器允许对同一个通道进行多次采样，用户可以完成过采样算法，这样可以获得更高的采样精度。

ADC模块主要包括以下几个特点：

①内置两个采样/保持（S/H）电路的12位ADC内核

②同步采样模式或顺序采样模式

③模拟输入的满量程：0V～3.3V（固定的），或者VREFHI～VREFLO（比例模式）

④以全系统时钟运行，无需预分频

⑤多达16个多路复用输入通道

⑥16个SOC，其触发源、采样窗口和通道均可配置

⑦16个结果寄存器（可单独寻址），用于存储转换值

⑧多个触发源

* S/W —软件立即启动
* ePWM 1～ePWM 7
* GPIO XINT2
* CPU定时器0/1/2
* ADCINT1/2

⑨9个灵活的PIE中断，在任意转换之后可以配置中断请求

4.2 ADC具体功能说明

1. SOC的工作原理

与之前的那些ADC不同，这个ADC不基于序列发生器（sequencer），而是基于SOC。术语SOC是一种配置设置，它定义的是单通道单转换。该设置包含3个配置：启动转换的触发源、转换通道和采集（采样）窗口尺寸。每个SOC都是单独配置的，并且可以是触发源、通道和/或采集窗口的任意组合。这一特性提供了一种非常灵活的转换配置方式，包括从“使用不同触发器、不同通道的单独采样”到“使用单个触发器、相同通道的过采样”。

SOCx的触发源由ADCSOCxCTL寄存器中的TRIGSEL字段以及ADCINTSOCSEL1或ADCINTSOCSEL2寄存器中的相应位联合配置。另外，软件通过ADCSOCFRC1寄存器也可以强制产生一个SOC事件。SOCx的通道和采样窗尺寸都通过ADCSOCxCTL寄存器的CHSEL和ACQPS字段来配置。

1. ADC采集（采样和保持）窗口

各外部驱动器驱动模拟信号的能力各不相同。有些电路需要更长的时间才能正确地将电荷（charge）传送到ADC的采样电容器（sampling capacitor）。为了解决这个问题，ADC可对每个SOC采样窗口的长度进行控制。每个ADCSOCxCTL寄存器都包含一个6位字段ACQPS，该字段决定采样和保持（S/H）窗的长度。对于SOC来说，写入该字段的值要比采样窗的期望值（单位为周期数）小1。因此，如果该字段为15表示采样时间其实为16个时钟周期。采样周期的最小值为7（ACQPS = 6）。总采样时间等于采样窗长度加上ADC转换时间（共13个ADC时钟）。

1. **AD转换的优先级**

**当数个SOC标志同时被设置，两种形式的优先级顺序中的一种决定它们转换的顺序。默认的决定方式是轮转。在这种策略中，没有某个SOC会有比其它更高的优先级。优先级由轮转指针决定。**ADCSOCPRIORITYCTL寄存器中的RRPOINTER指向最后转换的SOC。最高优先级SOC就是下一个比RRPOINTER值大的SOC，在SOC0到SOC15中轮回。复位时的值是32，因为0表示转换已经发生。当RRPOINTER值为32，最高优先级的是SOC0。当ADCCTL1.RESET被置位或者SOCPRICTL寄存器被写入，RRPOINTER被设备复位。

ADCSOCPRIORITYCTL寄存器的SOCPRIORITY可用于配置所有SOC的优先级。如果一个SOC被设置成高优先级，它将会当前转换完成之后中断轮转，把自己插入到下一次转换中。当转换完成，轮转在被中断处继续。如果两个高优先级的SOC同时被触发，编号较低的SOC被优先考虑。

1. **同时采样模式**

在某些应用中，保持两个采样的信号之间的最小延迟是非常重要的。ADC模块包括双采样保持电路，允许两个不同的通道同时采样。同时采样模式是通过ADCSAMPLEMODE寄存器为两个soc配置的。偶数SOC与接着的奇数SOC作为一对，使用同一个使能位。这一对的动作如下：

1. 其中一个SOCx的触发将开始一对的转换。
2. 一对通道的转换包括A和B对应的CHSEL的值（0-7）。
3. 两个通道同时采样。
4. A通道先转换。
5. A通道转换结束，偶数EOCx将会产生一个脉冲。B通道转换结束，奇数EOCx将会产生一个脉冲。
6. A通道的转换结果将会存放在偶数ADCRESULTx寄存器中，A通道的转换结果将会存放在偶数ADCRESULTx寄存器中
7. **转换结束和中断操作**

由于有16个独立的SOCx配置，所以有16个EOCx标志。在序列采样中，EOCx是直接与SOCx相关联的。在同时采样模式中，如上5所述。根据ADCCTL1.INTPULSEPOS的设定，EOCx脉冲将会发生在转换开始或者结束时。

ADC模块包括9个能被PIE标志或者通过PIE的中断，每个中断都可以配置接受EOCx信号作为中断源。哪个EOCx信号作为中断源是在INTSELxNy寄存器中配置的。另外，ADCINT1和ADCINT2信号可作为一个SOCx的触发。这有利于建立一个连续的转换。

1. **上电序列**

ADC复位后是关闭状态。在写任意ADC寄存器之前必须置位PCLKCR0寄存器中的ADCENCLK位。启动ADC的操作序列如下：

1. 如果希望使用外部参考源，在ADCCTL1寄存器的ADCREFSEL中使能这种模式。
2. 在ADCCTL1寄存器（5-7位ADCPWDN,ADCBGPWD,ADCREFPWD）中一起启动参考源，带隙和模拟电路。
3. 通过设置ADCCTL1寄存器的ADCENABLE使能ADC。
4. 在首次转换之前延时1毫秒。
5. **ADC校准**

任何转换器都固有一个零偏移误差和满量程的增益误差。该ADC出厂校时在25摄氏度校正两者，同时允许用户修改任何偏移量的校正应对应用程序环境的影响，如环境温度。除非处在某些仿真环境下，或者需要修改出厂设置，用户不需要执行任何特定的操作。ADC将会在设备引导过程中得到合适的校正。

1. **厂家设定与校准功能**

在制造和测试过程中，德州仪器伴随着一对内部晶振的设置，校正一些ADC设置。这些设置内嵌在保留的OTP memory中，作为一个C语言可调用函数Device\_cal()，在Boot ROM启动引导过程中，程序调用这个函数写出厂设置到各个有效寄存器。在这种情况发生时，ADC和内部振荡器不会保留他们的指定参数。如果引导程序在仿真过程中被跳过，用户必须确保校准设置能被写入各寄存器，以确保ADC和内部振荡器满足在数据手册中的要求。这可以手动调用Device\_cal()，或者在应用程序中设定。

1. **ADC零点偏移校准**

零点偏移误差被定义为，当转换一个在VREFLO电压时得到的结果。这个基本误差会影响ADC的所有转换，包括满刻度的增益和线性度指标，决定了转换器的直流精度。零点偏移误差可能是正的，或者是负的，正的意味着转换VREFLO时得到一个正的结果。负的意味着转换一个高于VREFLO的电压结果仍会是0。为了更正这种错误，两种误差的补码都会被写入ADCOFFTRIM寄存器。这个寄存器的值在AD转换结果保存到ADC结果寄存器之前会被用到。此操作被完全包含在ADC内核，所以结果的定时将不会受到影响，ADC能够保持全动态范围通过修改微调值。调用Device\_cal()把厂家校正的零点偏移写到ADCOFFTRIM寄存器，用户能够修改ADCOFFTRIM的值以减少环境造成偏移误差。这个可以通过设置ADCCTRL1的VREFLOCONV位实现，不需要任何一个ADC通道。

如下步骤重新校准ADC偏移：

1. Set ADCOFFTRIM to80 (50h)
2. SetADCCTL1.VREFLOCONV to 1
3. Perform multiple conversions on B5 (i.e. sampleVREFLO) and take an average to account for board noise
4. Set ADCOFFTRIM to 80 (50h) minus the averageobtained in step 3
5. SetADCCTL1.VREFLOCONV to 0.
6. **ADC满量程增益校准**

增益错误是一个增量，随着输入电压的增加。满量程增益错误发生在输入电压最大值的时候。如同偏移误差一样，增益误差可能是正的也可能是负的。一个正的满量程增益误差，意味着输入未来最大值之前转换结果就已经到达最大值。一个负的满量程增益误差，意味着转换结果永远达不到最大值。校正函数Device\_cal()会写一个厂家调整值到ADCREFTRIM寄存器以矫正ADC的满量程增益误差。这个寄存器在调用Device\_cal()之后不应该被改动。

1. **ADC偏移电流校正**

为了增加ADC的精度，Device\_cal()函数同样会向ADC的一个寄存器写入厂家调整值矫正偏移电流，这个寄存器在调用Device\_cal()之后不应该被修改。

**ADC例程：****（温度传感器）**

**综合例程：**

**Main.C如下：**

**#include**"main.h"

**#pragma** CODE\_SECTION(xint\_isr, "ramfuncs");

**#pragma** CODE\_SECTION(adc\_isr, "ramfuncs");

**extern**Uint16 RamfuncsLoadStart;

**extern**Uint16 RamfuncsLoadEnd;

**extern**Uint16 RamfuncsRunStart;

**voidmain**(**void**) {

**InitSysCtrl**(); //初始化系统时钟，选择内部晶振1，10MHZ，12倍频，2分频，初始化外设时钟，低速外,4分频

DINT;

//关总中断

IER = 0x0000; //关CPU中断

IFR = 0x0000; //清CPU中断标志

**InitPieCtrl**(); //关pie中断

**InitPieVectTable**(); //清中断向量表

EALLOW;

/\*\*配置中断向量表\*\*\*\*\*/

// PieVectTable.ECAP1\_INT = &ecap1\_isr;

// PieVectTable.SCIRXINTA = &sciaRxFifoIsr;

// PieVectTable.SCITXINTA = &sciaTxFifoIsr;

PieVectTable.ADCINT1 = &adc\_isr;

PieVectTable.TINT0 = &cpu\_timer0\_isr;

PieVectTable.XINT1 = &xint\_isr;

EDIS;

**MemCopy**(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);

**InitFlash**();

**InitAdc**(); // 初始化ADC引脚

**InitCpuTimers**(); // 初始化时钟

**ConfigCpuTimer**(&CpuTimer0, 60, 500000);

CpuTimer0Regs.TCR.all = 0x4041;

init\_gpio(); //初始化GPIO

Adc\_Config(); //配置ADC

Init\_EPwm1(); //初始化PWM4

TM1638\_Init(); //初始化LED

Lcd\_Init(); //初始化LCD

/\*\*使能CPU中断\*\*/

IER |= M\_INT1;

IER |= M\_INT4;

IER |= M\_INT6;

IER |= M\_INT9;

/\*\*使能pie中断\*\*/

PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Enable the PIE block

PieCtrlRegs.PIEIER1.bit.INTx1 = 1; // ADCINT1

PieCtrlRegs.PIEIER1.bit.INTx4 = 1; // XINT1

PieCtrlRegs.PIEIER1.bit.INTx7 = 1; // TINT0

PieCtrlRegs.PIEIER4.bit.INTx1 = 1; // ECAP1\_INT

// PieCtrlRegs.PIEIER9.bit.INTx1 = 1; // SCIRXINTA

// PieCtrlRegs.PIEIER9.bit.INTx2 = 1; // SCITXINTA

XIntruptRegs.XINT1CR.all = 0x0005; // 外部中断 n 控制寄存器

/\*\*开总中断\*\*/

EINT;

ERTM;

LCD\_Clear(WHITE); //LCD填充白色背景色

POINT\_COLOR = BLACK; //设置画笔为黑色

LCD\_ShowString(30, 100, "now the temp is:");//LCD显示字符串

**while** (1) {

LED\_Show(1, degC % 10, 0); //LED显示温度个位

LED\_Show(2, degC / 10, 0); //LED显示温度十位

LCD\_ShowNum(158, 100, degC, 2); //LCD显示温度

LCD\_ShowString(190, 100, "C");

// QDTFT\_Test\_Demo();

**if**(GpioDataRegs.GPADAT.bit.GPIO2==1)

GpioDataRegs.GPACLEAR.bit.GPIO0=1;

**else**

GpioDataRegs.GPASET.bit.GPIO0=1;

**if**(GpioDataRegs.GPADAT.bit.GPIO3==1)

GpioDataRegs.GPACLEAR.bit.GPIO1=1;

**else**

GpioDataRegs.GPASET.bit.GPIO1=1;

delay\_loop();

}

}