DSP 基础学习 2：GPIO与第一个实验：流水LED灯

原创

未过河的卒 于 2018-12-25 16:59:39 发布 9856 收藏 87

分类专栏： DSP基础学习 文章标签： DSP 数字信号处理 TMS320F28335 单片机 TI

本文链接：https://blog.csdn.net/huagengpai1994/article/details/85237753

DSP 基础学习 2：GPIO与第一个实验：流水LED灯

鄙人使用的DSP是TMS320F28335PGFA，开发板是pop28335（师兄遗留的，开发板大同小异，貌似POP现在倒闭了？），仿真器是XDS100V3。电脑系统为windows8.1，CCS软件版本是6.0.1。

上一篇简单介绍了DSP以及怎么部署开发环境，本着以做为主的原则，这一篇就要做出第一个实验：流水LED灯。为了做出这个实验，还得学习GPIO相关的内容。本篇用到的资源在以下这个链接里下载。

https://download.csdn.net/download/huagengpai1994/10875803

里面有四个文件夹，分别是例程、数据手册、用户手册和应用手册。

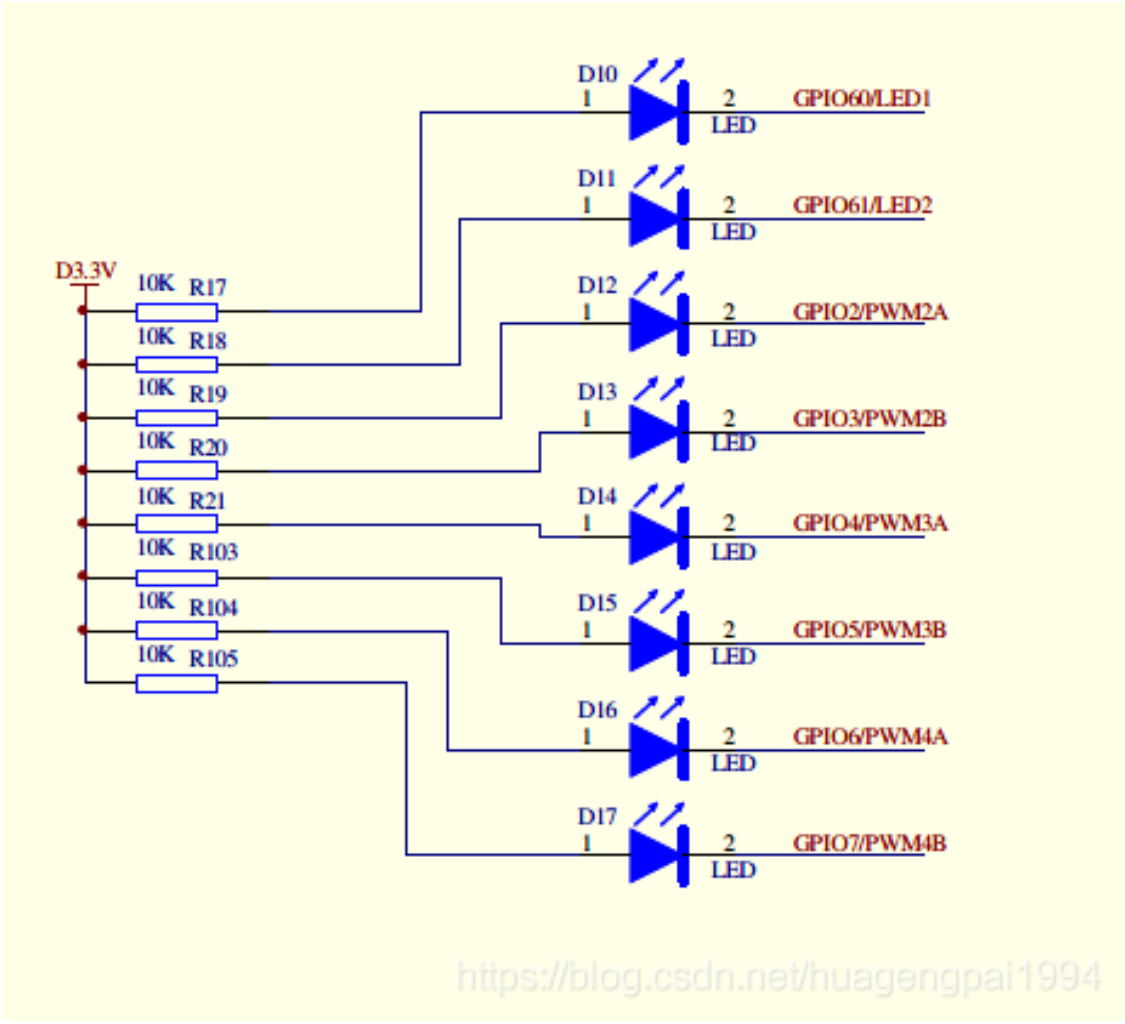
1 实验准备

1.1 实验简介

实验达到的效果是让led灯间歇依次高亮，也就是流水LED灯。

LED灯属于外设，要想控制外设就得知道硬件是怎么连接的，所以看原理图是必要的第一步。

在开发板原理图上可以找到LED灯部分，如下图。



可以看到有8个LED灯，一端通过限流电阻接在3.3V高电压上，D3.3表示直流3.3V。另一边分别接在GPIO60，GPIO61，GPIO2，GPIO3，GPIO4，GPIO5，GPIO6，GPIO7。

GPIO全称是：General Purpose Input Output (通用输入/输出)。它是DSP芯片用来与外界电路进行信息互换的端口。只要让GPIO输出低电压，那么对应的LED灯支路就会有电流通过，该LED灯被点亮，反之不亮。所以关键在于GPIO的使用。

1.2 GPIO介绍

DSP28335一共88个IO口分为3组

A组：GPIO0~GPIO31

B组：GPIO32~GPIO63

C组：GPIO64~GPIO87

下图是核心板原理如上DSP的部分，可以看到有很多GPIO口。

GPIO0/PWM1A	5	GPIO0/EPWM1A
GPIO1/PWM1B	6	GPIO1/EPWM1B/ECAP6/MFSRB
GPIO2/PWM2A	7	GPIO2/EPWM2A
GPIO3/PWM2B	10	GPIO3/EPWM2B/ECAP5/MCLKRB
GPIO4/PWM3A	11	GPIO4/EPWM3A
GPIO5/PWM3B	12	GPIO5/EPWM3B/MFSRA/ECAP1
GPIO6/PWM4A	13	GPIO6/EPWM4A/EPWMSYNCI/EPWMSYNCO
GPIO7/PWM4B	16	GPIO7/EPWM4B/MCLKRA/ECAP2
GPIO8/PWM5A	17	GPIO8/EPWM5A/CANTXB/ADCSOCAOn
GPIO9/PWM5B	18	GPIO9/EPWM5B/SCITXDB/ECAP3
GPIO10/PWM6A	19	GPIO10/EPWM6A/CANRXB/ADCSOCBOn
GPIO11/PWM6B	20	GPIO11/EPWM6B/SCIRXDB/ECAP4
GPIO12/TZ1n	21	GPIO12/TZ1n/CANTXB/MDXB
GPIO13/TZ2n	24	GPIO13/TZ2n/CANRXB/MDRB
GPIO16/TZ5n	27	GPIO16/SPISIMOA/CANTXB/TZ5n
GPIO17/TZ6n	28	GPIO17/SPISOMIA/CANRXB/TZ6n
GPIO18/CANRXA	62	GPIO18/SPICLKA/SCITXDB/CANRXA
GPIO19/CANTXA	63	GPIO19/SPISTEA/SCIRXDB/CANTXA
GPIO20/CANTXB	64	

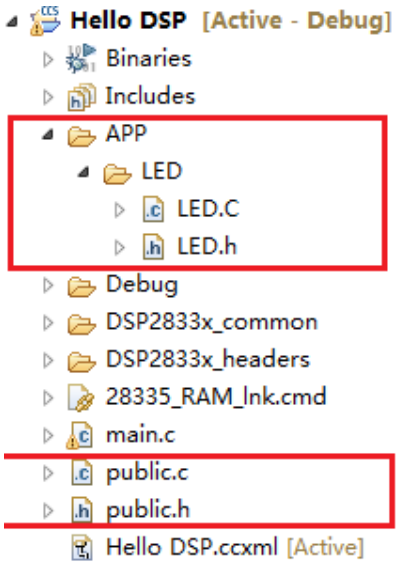
TI官方给出了一些手册用来说明芯片的使用。我这里有用户手册，数据手册和应用手册，下载链接在本文开头给出了。在用户手册的《sprufb0d(TMS320x2833x, 2823x System Control and Interrupts)》第六章General-Purpose Input/Output (GPIO)中详细说明了GPIO。这里就不挨个解读了，建议将该章通读一遍，虽然是英文的但应该都能看明白。

2 代码解读

把上篇的模板拷贝过来一份，直接使用该模板进行开发。

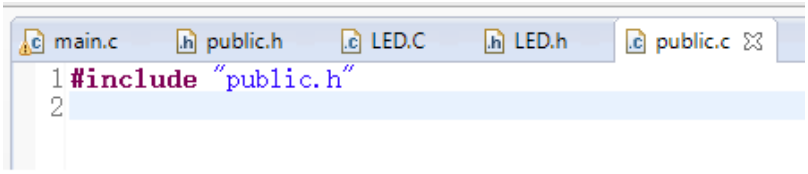
鄙人的本程序也是参考别人的例程开发的（买板子的时候店家给的，不知道是不是官方的），该例程只有一个main.c文件。所有例程都在资料里，本篇的相关的内容应该是例程17。

2.1 准备工作



在Hello DSP 文件夹下新建文件夹APP，然后在其中新建LED文件夹，接着在LED文件夹下新建LED.C和LED.h两个文件。APP是用来保存外设的文件，LED是一个外设。新建public.c和public.h，用来包含公共的头文件，否则main.c的内容过于多。这样是基于模块化的编程，将功能分离，逻辑上更简洁。

在public.c中写入以下代码：



在public.h中写入以下代码：

第1,2,7行代码是固定程式，在这里就是这样去写。C语言里这叫条件编译，是为了防止重定义。

第4,5行，表示包含这两个头文件，第一个头文件是为了包含DSP系统的基本文件，包含它相当于包含了另两个文件。第二个就是包含我们编写的外设文件，LED.h。

2.2 LED.h

在LED.h中写入一下代码。

```h
#ifndef LED_H
#define LED_H

#include "DSP28x_Project.h"

/*************************************宏定义*************************************************/
#define LED1_OFF  GpioDataRegs.GPBSET.bit.GPIO60 = 1       //LED D10 点亮
#define LED1_ON   GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1      //LED D10 熄灭
#define LED2_OFF  GpioDataRegs.GPBSET.bit.GPIO61 = 1       //LED D11 点亮
#define LED2_ON   GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1      //LED D11 熄灭
#define LED3_OFF  GpioDataRegs.GPASET.bit.GPIO2 = 1       //LED D12 点亮
#define LED3_ON   GpioDataRegs.GPACLEAR.bit.GPIO2 = 1      //LED D12 熄灭
#define LED4_OFF  GpioDataRegs.GPASET.bit.GPIO3 = 1       //LED D13 点亮
#define LED4_ON   GpioDataRegs.GPACLEAR.bit.GPIO3 = 1      //LED D13 熄灭
#define LED5_OFF  GpioDataRegs.GPASET.bit.GPIO4 = 1       //LED D14 点亮
#define LED5_ON   GpioDataRegs.GPACLEAR.bit.GPIO4 = 1      //LED D14 熄灭
#define LED6_OFF  GpioDataRegs.GPASET.bit.GPIO5 = 1       //LED D15 点亮
#define LED6_ON   GpioDataRegs.GPACLEAR.bit.GPIO5 = 1      //LED D15 熄灭
#define LED7_OFF  GpioDataRegs.GPASET.bit.GPIO6 = 1       //LED D16 点亮
#define LED7_ON   GpioDataRegs.GPACLEAR.bit.GPIO6 = 1      //LED D16 熄灭
#define LED8_OFF  GpioDataRegs.GPASET.bit.GPIO7 = 1       //LED D17 点亮
#define LED8_ON   GpioDataRegs.GPACLEAR.bit.GPIO7 = 1       //LED D17 熄灭

#define DELAY_TIME 2000000               //延时时间
/*********************************************************************************************/

/*************************************函数声明*************************************************/
void Init_LedGpio(void);
void delay(Uint32 t);
void LED_Start(void);
/*********************************************************************************************/
#endif
```

#ifndef #define #endif 是固定程式的条件编译。

#include "DSP28x_Project.h"是为了利用DSP自带的库文件。

宏定义了16个LED灯亮灭，因为GPIO60和GPIO61在B组，所以是在GPBSET和GPBCLEAR。SET置1表示输出高电压，此时LED灯灭。CLEAR置1表示输出低电压，这样LED就亮了。bit表示该组下的一个GPIO口。这种写法是C语言的结构体语法，不懂的话可以去百度学习一下。很显然此处使用一个比较复杂的结构体GpioDataRegs去控制DSP内部寄存器，进而设置想要的GPIO状态。说它复杂是因为该结构体里面又嵌套了结构体，不是有很多小点符号嘛。

#define DELAY_TIME 2000000 是宏定义了一个变量，用来给出延迟时间。

最后的三个函数声明是我们要给出的外设函数，.h文件声明，.c文件详细定义，这都是C语言的基本语法。

2.3 LED.c

在LED.c中写入以下代码：

```h
#include "LED.h"

void Init_LedGpio(void)
{
 EALLOW;
 //LED D10
 GpioCtrlRegs.GPBPUD.bit.GPIO60 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPBSET.bit.GPIO60 = 1;// Load output latch
```

```
 GpioCtrlRegs.GPBMUX2.bit.GPIO60 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPBDIR.bit.GPIO60  = 1;// GPIO11 = output

 //LED D11
 GpioCtrlRegs.GPBPUD.bit.GPIO61 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPBSET.bit.GPIO61 = 1;// Load output latch
 GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPBDIR.bit.GPIO61  = 1;// GPIO11 = output

 //LED D12
 GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO2 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;// GPIO11 = output

 //LED D13
 GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO3 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;// GPIO11 = output

 //LED D14
 GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO4 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;// GPIO11 = output

 //LED D15
 GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO5 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;// GPIO11 = output

 //LED D16
 GpioCtrlRegs.GPAPUD.bit.GPIO6 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO6 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;// GPIO11 = output

 //LED D17
 GpioCtrlRegs.GPAPUD.bit.GPIO7 = 0;// Enable pullup on GPIO11
 GpioDataRegs.GPASET.bit.GPIO7 = 1;// Load output latch
 GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;// GPIO11 = GPIO
 GpioCtrlRegs.GPADIR.bit.GPIO7 = 1;// GPIO11 = output

 EDIS;
}
void delay(Uint32 t)
{
 Uint32 i = 0;
 for (i = 0; i < t; i++);
}


void LED_Start(void)
{
 LED1_ON;           //LED1 D10 点亮
 delay(DELAY_TIME);         //延时
 LED1_OFF;           //LED1 D10 熄灭
```

```
LED2_ON;           //LED2 D11 点亮
delay(DELAY_TIME);         //延时
LED2_OFF;           //LED2 D11 熄灭

LED3_ON;           //LED3 D12 点亮
delay(DELAY_TIME);         //延时
LED3_OFF;           //LED3 D12 熄灭

LED4_ON;           //LED4 D13 点亮
delay(DELAY_TIME);         //延时
LED4_OFF;           //LED4 D13 熄灭

LED5_ON;           //LED4 D13 点亮
delay(DELAY_TIME);         //延时
LED5_OFF;           //LED4 D13 熄灭

LED6_ON;           //LED4 D13 点亮
delay(DELAY_TIME);         //延时
LED6_OFF;           //LED4 D13 熄灭

LED7_ON;           //LED4 D13 点亮
delay(DELAY_TIME);         //延时
LED7_OFF;           //LED4 D13 熄灭

LED8_ON;           //LED4 D13 点亮
delay(DELAY_TIME);         //延时
LED8_OFF;           //LED4 D13 熄灭
}
```

void Init_LedGpio(void)的作用是对LED初始化。EALLOW和EDIS指令其实是两条汇编指令。

#define EALLOW asm(" EALLOW")
#define EDIS asm(" EDIS")

这样做是因为TI的DSP为了提高安全性能，将很多关键寄存器作了保护处理。通过状态寄存器1（ST1）的位6设置与复位，来决定是否允许DSP指令对关键寄存器进行操作。这些关键寄存器包括：器件仿真寄存器、FLASH寄存器、CSM寄存器、PIE矢量表、系统控制寄存器、GPIOMux寄存器等等。

DSP由于在上电复位之后，状态寄存器基本上都是清零，而这样的状态下正是上述特殊寄存器禁止改写的状态。为了能够对这些特殊寄存器进行初始化，所以在对上述特殊寄存器进行改写之前，一定要执行汇编指令asm（“EALLOW”）或者宏定义EALLOW来设置状态寄存器1的C6位，在设置完寄存器之后，一定要注意执行汇编指令asm（“EDIS”）或者宏定义EDIS来清除状态寄存器1的C6位，来防止杂散代码或指针破坏寄存器内容。

总之写上这两个指令，夹在中间的指令才会有效。

8个LED灯的初始化套路是一致的，所以只解读第一个。

GpioDataRegs.GPBSET.bit.GPIO60 = 1; 刚才说过了是让LED灯灭，写在这里是为了初始化的时候让它灭。

GpioCtrlRegs是控制寄存器结构体，刚才说明的GpioDataRegs是数据寄存器结构体。数据寄存器反映的是管脚的状态，控制寄存器是对管脚进行设置。数据寄存器会将指定管脚置0置1就够了，控制寄存器见手册说明如下：

6.2 Configuration Overview

The pin function assignments, input qualification, and the external interrupt (XINT1 – XINT7, XNMI) sources are all controlled by the GPIO configuration control registers. In addition, you can assign pins to wake the device from the HALT and STANDBY low power modes and enable/disable internal pullup resistors. Table 39 and Table 40 list the registers that are used to configure the GPIO pins to match the system requirements.

Table 39. GPIO Control Registers

Name (1)	Address	Size (x16)	Register Description	Bit Description
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0-GPIO31)	Figure 53
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0-GPIO15)	Figure 55
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16-GPIO31)	Figure 56
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0-GPIO15)	Figure 47
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16-GPIO31)	Figure 48
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0-GPIO31)	Figure 59
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register (GPIO0-GPIO31)	Figure 62
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32-GPIO63)	Figure 54
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32-GPIO47)	Figure 57
GPBQSEL2	0x6F94	2	GPIO B Qualifier Select 2 Register (GPIO48 - GPIO63)	Figure 58
GPBMUX1	0x6F96	2	GPIO B MUX 1 Register (GPIO32-GPIO47)	Figure 49
GPBMUX2	0x6F98	2	GPIO B MUX 2 Register (GPIO48-GPIO63)	Figure 50
GPBDIR	0x6F9A	2	GPIO B Direction Register (GPIO32-GPIO63)	Figure 60
GPBPUD	0x6F9C	2	GPIO B Pull Up Disable Register (GPIO32-GPIO63)	Figure 63
GPCMUX1	0x6FA6	2	GPIO C MUX 1 Register (GPIO64-GPIO79)	Figure 51
GPCMUX2	0x6FA8	2	GPIO C MUX 2 Register (GPIO80-GPIO87)	Figure 52
GPCDIR	0x6FAA	2	GPIO C Direction Register (GPIO64-GPIO87)	Figure 61
GPCPUD	0x6FAC	2	GPIO C Pull Up Disable Register (GPIO64-GPIO87)	Figure 64

(1) The registers in this table are EALLOW protected. See Section 7.2 for more information.

GpioCtrlRegs.GPBPUD.bit.GPIO60 = 0;表示引脚有一个上拉电阻，上拉电阻的原理很简单此处不说明，达到的效果就是默认高电平。可能会有人说我设置该引脚高电平它就是高电平，设置低电平就是低电平，干嘛要默认啊。没有上拉电阻或者下拉电阻，叫悬空状态。悬空状态下管脚的电压很容易忽高忽低，可能你手指头摸摸电路板上的线管脚电压值就变了。所以要上拉稳定在高电平，只有你置低电平它才是低电平。

GpioCtrlRegs.GPBMUX2.bit.GPIO60 = 0; GpioCtrlRegs.GPBMUX2是复用选择器，用来设置IO口的复用功能。DSP只有88个IO口，为了充分利用引脚的功能，大部分引脚都有复用功能，此处我们把引脚当做通用IO口来用，所以此位设置为0。以后碰到复用功能的时候再说明。

GpioCtrlRegs.GPBDIR.bit.GPIO60 = 1; GpioCtrlRegs.GPBDIR是用来配置是用作输入口(2)还是输出口(1)，这里置1，用作输出。

表中还有一个变量是GPAQSEL，本次实验没有用到，它的作用是输入限定，这个比较复杂，用到的时候再具体说明吧，总的效果就是要限定最大输入消除噪声。

总的来看，void Init_LedGpio(void)的作用就是将GPIO配置好，完成初始化。

void delay(Uint32 t)，语义很明显了，是让CPU空跑，这样就可以间歇性的闪烁。前面宏定义的DELAY_TIME 2000000，差不多空跑跑这么多就是半秒把。很显然这种方式不仅计时不准还很浪费CPU的资源，最好的做法应该是用定时器定时，时间到了进入中断，不过这个以后再用。

void LED_Start(void) 的语义也很简单，点亮第一个灯，CPU空跑1秒，再关闭第一个灯同时点亮第二个灯，CPU再空跑1秒，依此进行到第8个灯。LEDx_ON和LEDx_OFF是LED.h中宏定义的函数，前面已经说过了。

现在外设的函数准备都完成了，下面去写main.c中的代码。

```c
#include "public.h"     // Device Headerfile and Examples Include File

void main(void)
{

   InitSysCtrl();//Initialize System Control

   extern Init_LedGpio();//Initalize GPIO

   //Disable Interrupt
   DINT;
   IER = 0x0000;
   IFR = 0x0000;

   InitPieCtrl();//Initialize Pie

   InitPieVectTable();

   while(1)            //死循环
  {
    extern LED_Start();

  }
}
```

main.c中只需要包含public.h就可以了，具体要包含哪些内容属于public.h公共头文件去写的。

InitSysCtrl();定义在DSP2833x_SysCtrl.c中，下为其函数具体内容，关闭关门狗，初始化PLL，初始化外设时钟。

```
void InitSysCtrl(void)
{

 // Disable the watchdog
 DisableDog();

 // Initialize the PLL control: PLLCR and DIVSEL
 // DSP28_PLLCR and DSP28_DIVSEL are defined in DSP2833x_Examples.h
 InitPll(DSP28_PLLCR,DSP28_DIVSEL);


 // Initialize the peripheral clocks
 InitPeripheralClocks();
}
```

这属于时钟部分，以后会细讲。这里 InitSysCtrl();只要认为是把时钟打开并设置为150MHZ就OK了。

Init_LedGpio();是我们自己写的初始化LED的函数。

下面这段代码是初始化的一部分，和中断、向量表等有关，以后再说，先默认这样去初始化。这些函数的定义都可以在包含的系统文件里找到。

```
//Disable Interrupt
 DINT;
 IER = 0x0000;
 IFR = 0x0000;

 InitPieCtrl();//Initialize Pie

 InitPieVectTable();
```

最后循环执行LED_Start()，效果就是LED间歇性依此点亮。

至此，代码讲解完毕。将文件Build后Debug，然后就可以在仿真器里面跑了，然后就可以在板子上看到效果。

3 结尾细节

如果在main.c中不加那两个extern，Bulid完毕会出现两个Warning：function “Init_LedGpio” declared implicitly；function “LED_Start” declared implicitly。也就是说我们在main.c中调用的两个函数隐含声明。网上说要把该函数声明为全局函数就好，于是鄙人在LED.h里对函数声明加了extern。这样做的道理是如果不声明为extern，那么编译器认为该函数只应该在LED.c中使用，毕竟delay()函数可没有declared implicitly，因为我只在LED.c中用了delay()。这样还是不行，看来该网友的搞法在DSP的CCS里行不通，虽然按照规范写法，编译器应该找不到，但毕竟智能嘛，找到了，程序能跑，不过给你俩warning改去吧。于是我将extern加在main.c函数调用那里，意思是让编译器去其他地方找这俩函数，这次还就通过了，看来这样是对的。

鄙人在main.c中加了关键字extern,依然会出现这两个warning，但是实验效果达到了，这个以后再研究吧，有懂的可以评论说一下。

另外，鄙人在跑的时候仿真器连接出了问题，问题如下：

```
Error connecting to the target:
(Error -151 @ 0x0)
One of the FTDI driver functions used during
the connect returned bad status or an error.
```

竟然显示连接不到仿真器，但我的仿真器灯都亮的好好的呢，设备管理器里也显示连接正常。百度了一下，有人说重新装仿真器驱动就好了。于是我关闭了CCS后，重新装了一下驱动，果然好了。出现这个问题很有可能是仿真器设置错误，即没有选定正确的型号，或者就是仿真器和电脑的接触不好，如果一切正常那么就是驱动出了问题，重装一下就好了。