03.分析第一个 LED 闪烁的工程

注意，教程使用 28069，实验使用 28035，请自行解决对照问题!

在上一节中，我们导入并调试了工程 Example_2806xGpioToggle。本节中，将对该工程进行具体解析，通过对该工程的说明，之后将尝试独立新建一个工程。

当 DSP 完成引导启动后，将进入 main 主程序开始执行说明。本节暂不对引导启动进行说明，直接针对主函数进行分析。

以下，将通过分析 Example_2806xGpioToggle.c 文件的构建进行说明。

3.1 头文件引入

在 Example_2806xGpioToggle.c 中，首先进行了头文件的引入。即：

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
```

按住 ctrl 并点击该语句，将跳转到该头文件。DSP28x_Project.h 中包含了三个头文件。即：

```
#include "F2806x_Cla_typedefs.h"// F2806x CLA Type definitions
#include "F2806x_Device.h"      // F2806x Headerfile Include File
#include "F2806x_Examples.h"    // F2806x Examples Include File
```

3.1.1 F2806x_Cla_typedefs.h

查看该头文件，其中主要包含了针对 CLA（控制律加速器）外设的数据类型定义以及部分宏定义。

3.1.2 F2806x_Device.h

该头文件包含的主要内容如下。

1. 器件定义

头文件内定义了使用的器件。TI 所提供的器件外设驱动可供某一系列器件使用，但其中的差异需要指明外设具体型号。例如，对于 TMS320F2806x 系列器件，x 代表了多种不同具体型号。例如 F28069 核心板所使用的是 28069UPZ 器件，这里将其定义为目标器件。

```
#define   DSP28_28069UPZ   TARGET
```

2. 通用 CPU 指令定义

头文件定义了 CPU 级的部分控制指令。通过宏定义 define 的形式将汇编形式的指令转化为简单的指令。

```
#define  EINT    __asm(" clrc INTM") //清中断
#define  DINT    __asm(" setc INTM") //禁止中断
#define  ERTM    __asm(" clrc DBGM") //使能全局中断
#define  DRTM    __asm(" setc DBGM")  //使能全局实时中断
#define  EALLOW __asm(" EALLOW") //写保护关闭
#define  EDIS    __asm(" EDIS")  //写保护开启
#define  ESTOP0 __asm(" ESTOP0") //仿真停止
```

定义了中断组 M_INTx 和位操作指令 BITx。这种写法方便了程序设计，也增加了程序的可读性。例如，如果相对 GPIO 的倒数第二位 IO 进行操作，对应的位写入应该是二进制的 0b100，也就是十进制的 4。如果写程序时还要这样计算位操作，将非常繁琐和容易出错。如果有位定义，可以直接使用 BIT2 即可。

3. 引入了全部外设的头文件

形如 F2806x_Adc.h 的是头文件是外设库的头文件。这里引入了 F2806x 系列器件的全部头文件。

```
//---------------------------------------------------------------------------
// Include All Peripheral Header Files:
//

#include "F2806x_Adc.h"                   // ADC Registers
#include "F2806x_BootVars.h"              // Boot ROM Variables
#include "F2806x_Cla.h"                   // Control Law Accelerator Registers
#include "F2806x_Comp.h"                  // Comparator Registers
#include "F2806x_CpuTimers.h"             // 32-bit CPU Timers
#include "F2806x_DevEmu.h"                // Device Emulation Registers
#include "F2806x_Dma.h"                           // Direct Memory Access Registers
#include "F2806x_ECan.h"                  // Enhanced CAN Registers
#include "F2806x_ECap.h"                  // Enhanced Capture
#include "F2806x_EPwm.h"                  // Enhanced PWM
#include "F2806x_EQep.h"                  // Enhanced QEP
#include "F2806x_Gpio.h"                  // General Purpose I/O Registers
#include "F2806x_HRCap.h"                       // High Resolution Capture Registers
#include "F2806x_I2c.h"                   // I2C Registers
#include "F2806x_Mcbsp.h"                       // McBSP Registers
#include "F2806x_NmiIntrupt.h"            // NMI Interrupt Registers
#include "F2806x_PieCtrl.h"               // PIE Control Registers
#include "F2806x_PieVect.h"               // PIE Vector Table
#include "F2806x_Spi.h"                   // SPI Registers
#include "F2806x_Sci.h"                   // SCI Registers
#include "F2806x_SysCtrl.h"               // System Control/Power Modes
#include "F2806x_Usb.h"                         // USB Registers
#include "F2806x_XIntrupt.h"              // External Interrupts
```

4. 定义了器件具体外设

对于不同的器件所包含的外设种类和个数种类可能不同。及时 F28069 系列 DSP，仍具有多个有差别的型号。这里，通过宏定义目标器件和使用条件编译，可以确定器件具体的外设情况。

```
#if (DSP28_28062UPZ || DSP28_28063UPZ || DSP28_28064UPZ || DSP28_28065UPZ || DSP28_28066UPZ || DSP28_28067UPZ
|| DSP28_28068UPZ || DSP28_28069UPZ)
#define DSP28_EPWM1   1
#define DSP28_EPWM2   1
#define DSP28_EPWM3   1
#define DSP28_EPWM4   1
#define DSP28_EPWM5   1
#define DSP28_EPWM6   1
#define DSP28_EPWM7   1
#define DSP28_EPWM8           1
#define DSP28_ECAP1    1
#define DSP28_ECAP2    1
#define DSP28_ECAP3    1
#define DSP28_EQEP1           1
#define DSP28_EQEP2    1
#define DSP28_HRCAP1 1
#define DSP28_HRCAP2 1
#define DSP28_HRCAP3 1
#define DSP28_HRCAP4 1

#define DSP28_SPIA       1
```

```
#define DSP28_SPIB              1
#define DSP28_I2CA      1
#define DSP28_SCIA      1
#define DSP28_SCIB      1
#define DSP28_ECANA             1
#define DSP28_MCBSPA            1
#define DSP28_USB0              1

#define DSP28_COMP1             1
#define DSP28_COMP2             1
#define DSP28_COMP3             1
#endif
```

3.1.3 F2806x_Examples.h

F2806x 系列 DSP 的系统可以有多种设置。例如，时钟可以使用外部或内部不同的时钟源，同时可以设置不同的系统主频。

在示例程序中，系统的初始化等均做出了相应的设置。在 F2806x_Examples.h 头文件中对初始化所需的参数进行了定义。如对时钟源的选择，系统主频的设置参数等。

3.2 Main 函数解析

3.2.1宏定义以及函数声明

Example_2806xGpioToggle 示例程序中，可以通过三种方式实现 IO 电平的翻转。同样采用的是条件编译的手段。这里通过宏定义的方式实现对编译条件的选择。

```
// Select the example to compile in.    Only one example should be set as 1
// the rest should be set as 0.
#define EXAMPLE1 1    // Use DATA registers to toggle I/O's
#define EXAMPLE2 0    // Use SET/CLEAR registers to toggle I/O's
#define EXAMPLE3 0    // Use TOGGLE registers to toggle I/O's
```

此处，选择 EXAMPLE1 方式进行编译，即使用 DATA 寄存器进行 IO 的翻转。

Example_2806xGpioToggle 示例程序中使用了几个函数，这几个函数的定义均在 Example_2806xGpioToggle.c 文件中。而是用这些函数需要进行声明，因此，在程序开始调用前进行函数的声明。

```
// Prototype statements for functions found within this file.
void delay_loop(void);
void Gpio_select(void);
void Gpio_example1(void);
void Gpio_example2(void);
void Gpio_example3(void);
```

3.2.2系统的初始化流程

当 DSP 开始执行 main 程序后，首先需要对系统进行初始化操作。即使 Example_2806xGpioToggle 示例程序中只是简单翻转 IO，但其初始化流程适用于大多数程序，具有普遍意义。

以下，将逐条解析其初始化流程。

1. 系统控制初始化

```
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2806x_SysCtrl.c file.
   InitSysCtrl();
```

该函数完成系统的初始化操作。包括禁止看门狗、ADC 校准、将系统时钟设置为 90MHz 以及外设的时钟使能。看门狗、ADC 等外设均在后续章节中详细介绍，这里暂且知道其已初始化完成即可。

值得注意的是，初始化函数 InitPeripheralClocks()将 F28069 器件的全部时钟均使能。很多读者有过 MCU 等学习经验，知道任何外设在使用时要首先要使能时钟。在 C2000 系列示例程序学习过程中，外设初始化往往不包含外设时钟使能。原因在于在系统初始化过程中，已经使能了全部外设时钟。需要说明的是，如有可能，未使用的外设其时钟最好关闭，这样有助于系统功耗的降低。

2. GPIO 初始化

该处完成对系统 GPIO 功能的初始化。但在 Example_2806xGpioToggle 示例程序中没有使用 InitGpio()函数，而是使用了 Gpio_select()函数。现在不必对函数的实现细节太过关注，只需要知道该函数的作用是将 GPIO 的 A 口和 B 口相应管脚配置为 GPIO，方向为输出或者输入。控制 LED 闪烁的管脚对应作为输出的 IO 端口。

需要注意的是，Gpio_select()中的四行程序应将 GPAMUX1 改为 GPBMUX1。原程序稍有错误。而程序正常执行的原因是:在芯片上电复位后，我们所使用的 IO 管脚默认即为 GPIO 功能，不需要额外的初始化。修改后的程序如下所示。

```
void Gpio_select(void)
{
      EALLOW;
      GpioCtrlRegs.GPAMUX1.all = 0x00000000;    // All GPIO
      GpioCtrlRegs.GPAMUX2.all = 0x00000000;    // All GPIO
      GpioCtrlRegs.GPBMUX1.all = 0x00000000;    // All GPIO
      GpioCtrlRegs.GPADIR.all = 0xCFFFFFFF;     // All outputs
      GpioCtrlRegs.GPBDIR.all = 0x0000000F;     // All outputs
      EDIS;

}
```

3. 中断管理初始化

C2000 系列 DSP 的中断管理较为复杂，后续有专门章节进行讲解。这里只需要知道，以下语句将完成中断模块的初始化。

```
// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize 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 F2806x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   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 F2806x_DefaultIsr.c.
// This function is found in F2806x_PieVect.c.
   InitPieVectTable();
```

3.2.3外设初始化与用户代码

初始化完成后，用户需要对所用外设进行初始化，并添加用户代码。

Example_2806xGpioToggle 示例程序中仅仅用到 GPIO 功能，不需要对其他外设进行初始化。

用户代码是通过条件编译所确定的 Gpio_example1()函数。

```
void Gpio_example1(void)
    {
        // Example 1:
        // Toggle I/Os using DATA registers
        for(;;)
        {
            GpioDataRegs.GPADAT.all     =0xffffffff; //赋值修改为0xffffffff
            GpioDataRegs.GPBDAT.all     =0xffffffff; //赋值修改为0xffffffff
            delay_loop();

            GpioDataRegs.GPADAT.all     =0x0;   //赋值修改为0x0
            GpioDataRegs.GPBDAT.all     =0x0;   //赋值修改为0x0
            delay_loop();
         }
}
```

可以看到，程序执行到该函数后将执行一个死循环。先将 GPIO A 口与 B 口的 DATA 寄存器全部写为 1，执行 delay_loop()函数；再将 GPIO A 口与 B 口的 DATA 寄存器全部写为 0，执行 delay_loop()函数。如此循环执行。

delay_loop()函数非常简单，就是执行一个 1000000 次的 for 循环。执行如此多的循环需要耗费 CPU 一定时间，故可以用作延时函数。

```
void delay_loop()
    {
        long        i;   //将数据类型从short改为long
        for (i = 0; i < 1000000; i++) {}   //将循环次数改为1 000 000
    }
```

至此，应当大致猜想和体会到，正是通过控制 IO 的 DATA 寄存器可以实现 GPIO 电平的控制，电平的变化将反应为 LED 的亮灭。而亮灭的过程不断循环则形成闪烁效果。

在后续的 GPIO 介绍章节中，将详细说明对 GPIO 的操作使用方法。

3.3 外设与初始化源文件的引入

在 F2806x_Device.h 中，我们引入了器件的全部头文件。Example_2806xGpioToggle 示例程序中除 GPIO 外没有使用其他外设，所以不需要引入外设的源文件。

在初始化过程中，引用了一些初始化函数，而这些初始化函数定义在相应的源文件中。如下表所示。

函数	源文件	源文件说明

InitSysCtrl()	F2806x_SysCtrl.c	系统控制与初始化
InitPieCtrl()	F2806x_PieCtrl.c	PIE 控制寄存器初始化
InitPieVectTable()	F2806x_PieVect.c	PIE 中断向量表初始化
	F2806x_DefaultIsr.c	PIE 默认中断服务函数

3.4 位域结构体支持源文件引入

C2000 系列 DSP 头文件和外设示例程序使用位域结构体方法，映射和访问基于 F28x 外设寄存器。位于结构体方法将在后续章节详细说明，这里仅仅简要说明其所需的支持源文件。

F2806x_GlobalVariableDefs.c	定义结构体变量，并分配相应数据区
F2806x_Headers_nonBIOS.cmd	链接命令文件，指定数据区具体地址，对应寄存器实际地址

3.5 链接器命令文件引入

除分配寄存器地址外，还需要对 CPU 的 RAM 或者 FLASH 区块进行分配。28069_RAM_lnk.cmd 文件负责对内存区块的分配。

学习过程中，大多数情况使用 RAM_lnk，使程序在 RAM 中运行。程序在 RAM 中运行使得调试更加迅速，同时不会发生 flash 锁死等情况；但 RAM 运行的程序在掉电后会丢失，需要重新烧录。

后续章节将介绍在 flash 烧写的方法。

3.6 器件启动文件引入

当 F28x 器件启动后，需要启动引导到指定代码区域。F2806x_CodeStartBranch.asm 负责该功能。

3.7 延时函数源文件引入

在 F2806x_Examples.h 文件中，有一个精确的 DELAY_US(A)函数，该函数的宏定义原型 DSP28x_usDelay()函数原型在 F2806x_usDelay.asm 源文件中。