目 录

第1章 硬件类型定义(hw_types.h)

1.1 HWREG( )、HWREGH( )、HWREGB( )

对 LM3S 系列单片机片内外设硬件寄存器的访问，可以采用 Luminary Micro 公司发布的《Stellaris 驱动库》头文件“ hw_types.h ”里定义的 3 个宏函数：HWREG( )、HWREGH( )、HWREGB( )，详见 表 1.1 的描述。

这 3 个宏函数用来访问 LM3S 系列单片机的硬件寄存器。它们在定义的时候因为被声明为 volatile 属性，所以这种访问不会被编译器优化掉，即每次读取时都返回硬件寄存器的当前值、每次写入时都会把最新的数值写入硬件寄存器。

利用宏函数访问硬件寄存器的具体例子如 程序清单 1.1 所示。

表 1.1 宏函数 HWREG( )、HWREGH( )、HWREGB( )

功能	HWREG(x)： 以全字（32 位）方式访问硬件寄存器 x。 HWREGH(x)：以半字（16 位）方式访问硬件寄存器 x。 HWREGB(x)：以字节（ 8 位）方式访问硬件寄存器 x。
原型	#define HWREG(x) (*((volatile unsigned long *)(x))) #define HWREGH(x) (*((volatile unsigned short *)(x))) #define HWREGB(x) (*((volatile unsigned char *)(x)))
参数	x：硬件寄存器的地址。
返回	读操作：返回硬件寄存器的当前值。 写操作：无。

程序清单 1.1 硬件寄存器访问示例

```
#include  "hw_types.h"

#define  SYSCTL_BASE    0x400FE000                  /*  定义系统控制模块的基址             */
#define  RCGC2          (SYSCTL_BASE+0x108)         /*  时钟门控寄存器 2，其位 1 控制 GPIOB */

void  timeDelay (unsigned long  ulVal)
{
    do {
    } while ( --ulVal != 0 );
}

int  main (void)
{
    timeDelay(500000L);                             /*  开机延迟                           */
    HWREG(RCGC2) |= 0x00000002;                     /*  选通 GPIOB 的时钟，即使能 GPIOB    */
    for (;;) {
    }
}
```

1.2 HWREGBITW( )、HWREGBITH( )、HWREGBITB( )

LM3S 系列单片机采用的是 ARM 公司设计先进的 Cortex-M3 内核。该内核采用了一项被称为“ Bit-banding ”的新技术，可以显著改善对片内 SRAM 以及片内外设的位操作性能。在头文件“ hw_types.h ”里定义有 3 个宏函数：HWREGBITW( )、HWREGBITH( )、HWREGBITB( )，为“ Bit-banding ”位操作提供了方便，专门用来访问 LM3S 系列单片机硬件寄存器当中的某个位。详见 表 1.2 的描述。

对存储单元或寄存器当中某一位的访问，按照传统的“ 读-改-写 ”分三步走的操作模式，效率不高；而采用“ Bit-banding ”后成为“ 直接读 ”和“ 直接写 ”，效率明显提高。在整个 4GB 存储空间里划分出了 2 个位操作区域，可以分别支持对片内 SRAM 和片内外设的位操作。“ Bit-banding ”的存储空间映射如 图 1.1 所示。

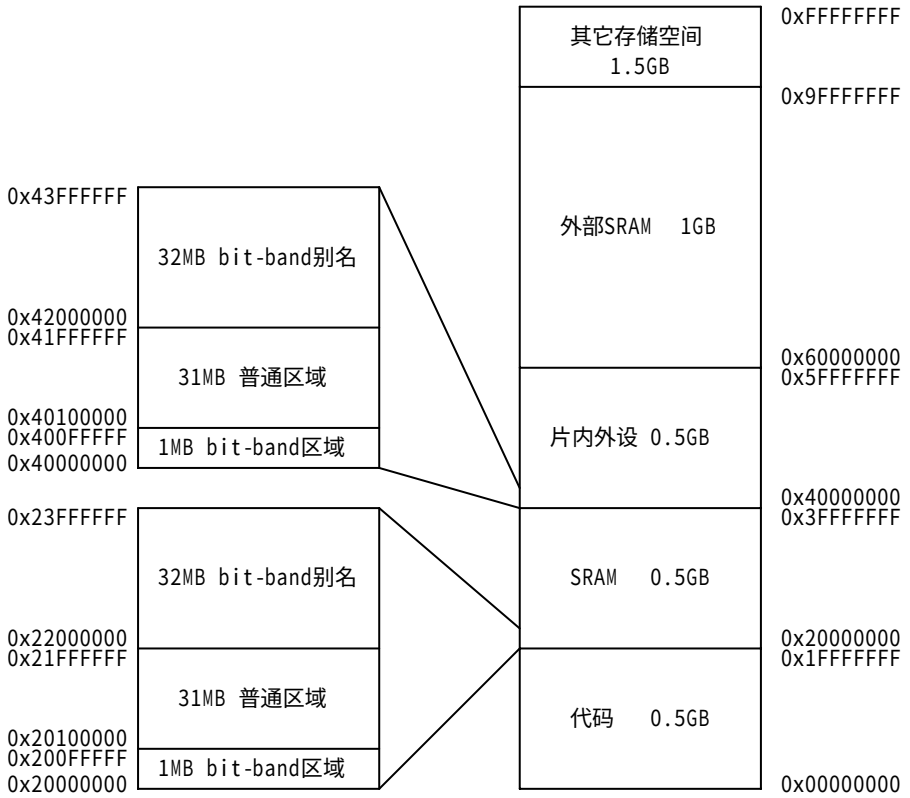

图 1.1 Bit-banding 存储空间映射

规则：对 32MB 别名区的全字访问映射为对 1MB bit-band 区的位访问。

注意：在LM3S系列中，每个GPIO模块数据寄存器GPIODATA的访问方式比较特殊，不能采用“ Bit-banding ”方式。

利用宏函数访问硬件寄存器当中某个位的具体例子如 程序清单 1.2 所示。

表 1.2 宏函数 HWREGBITW( )、HWREGBITH( )、HWREGBITB( )

功能	HWREGBITW(x,b)：以“ Bit-banding ”方式访问全字寄存器 x 当中的第 b 位。 HWREGBITH(x,b)：以“ Bit-banding ”方式访问半字寄存器 x 当中的第 b 位。 HWREGBITB(x,b)：以“ Bit-banding ”方式访问字节寄存器 x 当中的第 b 位。
原型	#define HWREGBITW(x,b) \ HWREG(((unsigned long)(x) & 0xF0000000) | 0x02000000 | \ (((unsigned long)(x) & 0x000FFFFF) << 5) | ((b) << 2))

	#define HWREGBITH(x,b) \ HWREGH(((unsigned long)(x) & 0xF0000000) | 0x02000000 | \ (((unsigned long)(x) & 0x000FFFFF) << 5) | ((b) << 2)) #define HWREGBITB(x,b) \ HWREGB(((unsigned long)(x) & 0xF0000000) | 0x02000000 | \ (((unsigned long)(x) & 0x000FFFFF) << 5) | ((b) << 2))
参数	x：bit-band 区地址，取值 0x20000000～0x200FFFFF 或 0x40000000～0x400FFFFF。 b：位地址，在 3 个宏函数中取值范围分别是 0～31、0～15、0～7。
返回	读操作：返回硬件寄存器 x 第 b 位的值（0 或 1）。 写操作：无。

程序清单 1.2 硬件寄存器位访问示例

```
#include  "hw_types.h"

#define  SYSCTL_BASE    0x400FE000                  /*  定义系统控制模块的基址             */
#define  RCGC2          (SYSCTL_BASE+0x108)         /*  时钟门控寄存器 2，其位 1 控制 GPIOB */

void  timeDelay (unsigned long  ulVal)
{
    do {
    } while ( --ulVal != 0 );
}

int  main (void)
{
    timeDelay(500000L);                             /*  开机延迟                           */
    HWREGBITW(RCGC2, 1) = 1;                        /*  选通 GPIOB 的时钟，即使能 GPIOB    */
    for (;;) {
    }
}
```

第2章 通用输入输出端口(gpio.h)

2.1 使能 GPIO 模块

LM3S 系列所有片内外设只有在 RCGCn 寄存器相应的控制位置位后才可以工作，否则被禁止。暂时不用的片内外设被禁止后可以节省功耗。

LM3S 系列不同型号的芯片 GPIO 模块数量也不相同，但都要求在使用之前使能。使能的方法是调用头文件“ sysctl.h ”里的函数 SysCtlPeripheralEnable( )。例如，要使能 GPIOB 模块的操作为：

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

2.2 GPIO 管脚配置

对于 LM3S 系列的单片机，如果要访问其 GPIO 管脚，则必须先进行正确的配置。

2.2.1 GPIODirModeSet( )

LM3S 系列的 GPIO 管脚有两大类用法：作为 I/O、非 I/O 功能。作为 I/O 时，可以配置成输入或输出。设置为非 I/O 功能后，管脚状态由该功能模块的硬件逻辑来决定。调用函数 GPIODirModeSet( )可以将指定的 GPIO 管脚设置成输入或输出，或者由硬件控制，详见 表 2.1 的描述。

表 2.1 函数 GPIODirModeSet( )

功能	设置所选 GPIO 端口指定管脚的方向和模式。
原型	void GPIODirModeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulPinIO);
参数	ulPort：所选 GPIO 端口的基址。应当取下列值之一： GPIO_PORTA_BASE // GPIOA 的基址（0x40004000） GPIO_PORTB_BASE // GPIOB 的基址（0x40005000） GPIO_PORTC_BASE // GPIOC 的基址（0x40006000） GPIO_PORTD_BASE // GPIOD 的基址（0x40007000） GPIO_PORTE_BASE // GPIOE 的基址（0x40024000） GPIO_PORTF_BASE // GPIOF 的基址（0x40025000） GPIO_PORTG_BASE // GPIOG 的基址（0x40026000） GPIO_PORTH_BASE // GPIOH 的基址（0x40027000） ucPins：指定管脚的位组合表示。应当取下列值之间的任意“ 或运算 ”组合： GPIO_PIN_0 // GPIO 管脚 0 的位表示（0x01） GPIO_PIN_1 // GPIO 管脚 1 的位表示（0x02） GPIO_PIN_2 // GPIO 管脚 2 的位表示（0x04） GPIO_PIN_3 // GPIO 管脚 3 的位表示（0x08） GPIO_PIN_4 // GPIO 管脚 4 的位表示（0x10） GPIO_PIN_5 // GPIO 管脚 5 的位表示（0x20） GPIO_PIN_6 // GPIO 管脚 6 的位表示（0x40） GPIO_PIN_7 // GPIO 管脚 7 的位表示（0x80）

	ulPinIO：管脚的方向或模式。应当取下列值之一： GPIO_DIR_MODE_IN // 输入方向 GPIO_DIR_MODE_OUT // 输出方向 GPIO_DIR_MODE_HW // 硬件控制
返回	无。

2.2.2 GPIOPadConfigSet( )

LM3S 系列的 GPIO 可以工作在多种模式下，对用户来说非常灵活，能够满足不同场合的需求。GPIO 管脚输出强度可以选择 2mA、4mA、8mA 或者带转换速率（Slew Rate）控制的 8mA 驱动。驱动强度越大表明带负载能力越强，但功耗也越高。对绝大多数应用场合选择 2mA 驱动即可满足要求。GPIO 管脚类型可以配置成输入、推挽、开漏三大类，每一类当中还有上拉、下拉的区别。对于配置用作输入端口的管脚，端口可按照要求设置，但是对输入唯一真正有影响的是上拉或下拉终端的配置。

在头文件“ gpio.h ”里的函数 GPIOPadConfigSet( )可以提供上述功能的配置，详见 表 2.2 的描述。

关于转换速率（Slew Rate）的解释。对输出信号采取适当舒缓的转换速率控制对抑制信号在传输线上的反射和电磁干扰非常有效。按照 LM3S 系列《数据手册》里给出的数据，在 8mA 驱动下，GPIO 输出上升和下降时间额定值都为 6ns，而在使能 8mA 转换速率控制以后，上升和下降时间额定值增加到 10ns 和 11ns，有了明显的延缓。8mA 驱动在使能转换速率控制后，并不影响其静态驱动能力，仍然是 8mA。

表 2.2 函数 GPIOPadConfigSet( )

功能	设置所选 GPIO 端口指定管脚的驱动强度和类型。
原型	void GPIOPadConfigSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulStrength, unsigned long ulPadType);
参数	ulPort：所选 GPIO 端口的基址。 ucPins：指定管脚的位组合表示。 ulStrength：指定输出驱动强度。应当取下列值之一： GPIO_STRENGTH_2MA // 2mA 驱动强度 GPIO_STRENGTH_4MA // 4mA 驱动强度 GPIO_STRENGTH_8MA // 8mA 驱动强度 GPIO_STRENGTH_8MA_SC // 带转换速率（Slew Rate）控制的 8mA 驱动 ulPadType：指定管脚类型。应当取下列值之一： GPIO_PIN_TYPE_STD // 推挽 GPIO_PIN_TYPE_STD_WPU // 带弱上拉的推挽 GPIO_PIN_TYPE_STD_WPD // 带弱下拉的推挽 GPIO_PIN_TYPE_OD // 开漏 GPIO_PIN_TYPE_OD_WPU // 带弱上拉的开漏 GPIO_PIN_TYPE_OD_WPD // 带弱下拉的开漏 GPIO_PIN_TYPE_ANALOG // 模拟比较器
返回	无。

2.2.3 GPIOPinTypeGPIOOutput( )

该函数实际上是通过调用函数 GPIODirModeSet( )和 GPIOPadConfigSet( )实现的，将指定管脚设置为 2mA 驱动强度的推挽输出模式。详见 表 2.3 的描述。

表 2.3 函数 GPIOPinTypeGPIOOutput( )

功能	设置所选 GPIO 端口指定的管脚为输出模式。
原型	void GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins);
参数	ulPort：所选 GPIO 端口的基址。 ucPins：指定管脚的位组合表示。
返回	无。

2.2.4 GPIOPinTypeGPIOInput( )

该函数实际上是通过调用函数 GPIODirModeSet( )和 GPIOPadConfigSet( )实现的，将指定管脚设置为输入模式。详见 表 2.4 的描述。

表 2.4 函数 GPIOPinTypeGPIOInput( )

功能	设置所选 GPIO 端口指定的管脚为输入模式。
原型	void GPIOPinTypeGPIOInput(unsigned long ulPort, unsigned char ucPins);
参数	ulPort：所选 GPIO 端口的基址。 ucPins：指定管脚的位组合表示。
返回	无。

2.3 GPIO 管脚读写操作

对 GPIO 管脚的读写操作是通过函数 GPIOPinWrite( )和 GPIOPinRead( )实现的。这是两个非常重要而且很常用的库函数。

2.3.1 GPIOPinWrite( )

调用该函数时，如果管脚已经被配置为输出状态，则 GPIO 管脚状态立即更新；如果管脚配置为输入状态，则该函数的执行不会有任何效果。详见 表 2.5 的描述。

利用函数 GPIOPinWrite( )写 GPIO 管脚的具体例子如 程序清单 2.1 所示。

表 2.5 函数 GPIOPinWrite( )

功能	向所选 GPIO 端口的指定管脚写入一个值，以更新管脚状态。
原型	void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);
参数	ulPort：所选 GPIO 端口的基址。 ucPins：指定管脚的位组合表示。 ucVal：写入指定管脚的值。 ucPins 指定的管脚对应的 ucVal 当中的位如果是 1，则置位相应的管脚，如果是 0，则清零相应的管脚。
返回	无。

程序清单 2.1 GPIO 管脚写操作示例

```
#include  "hw_types.h"
#include  "hw_memmap.h"
#include  "hw_sysctl.h"
#include  "hw_gpio.h"
#include  "src/sysctl.h"
#include  "src/gpio.h"

#define  PB0    GPIO_PIN_0
#define  PB1    GPIO_PIN_1

void  timeDelay (unsigned long  ulVal)
{
    do {
    } while ( --ulVal != 0 );
}

int  main (void)
{
    timeDelay(500000L);                                     /*  开机延迟                    */

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);            /*  使能 GPIOB 模块             */
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, (PB0 | PB1));    /*  设置 PB0 和 PB1 为输出模式  */

    for (;;) {
        GPIOPinWrite(GPIO_PORTB_BASE, (PB0 | PB1), 0x01);   /*  置位 PB0，清零 PB1          */
        timeDelay(200000L);

        GPIOPinWrite(GPIO_PORTB_BASE, (PB0 | PB1), 0x02);   /*  清零 PB0，置位 PB1          */
        timeDelay(200000L);
    }
}
```

2.3.2 GPIOPinRead( )

读取所选 GPIO 端口指定管脚的值。输入和输出管脚的值都能返回。详见 表 2.6 的描述。注意：参数 ucPins 未指定的管脚读回的值是 0。

利用函数 GPIOPinRead( )读 GPIO 管脚状态的具体例子如 程序清单 2.2 所示。

表 2.6 函数 GPIOPinRead( )

功能	读取所选 GPIO 端口指定管脚的值。
原型	long GPIOPinRead(unsigned long ulPort, unsigned char ucPins);
参数	ulPort：所选 GPIO 端口的基址。 ucPins：指定管脚的位组合表示。

返回	返回 1 个位组合的字节。它提供了由 ucPins 指定管脚的状态，对应的位值表示 GPIO 管脚的高低状态。ucPins 未指定的管脚位值是 0。返回值已强制转换为 long 型，因此位 31:8 应该忽略。

程序清单 2.2 GPIO 管脚读写操作示例

```
#include  "hw_types.h"
#include  "hw_memmap.h"
#include  "hw_sysctl.h"
#include  "hw_gpio.h"
#include  "src/sysctl.h"
#include  "src/gpio.h"

#define  PB0    GPIO_PIN_0
#define  PB1    GPIO_PIN_1

void  timeDelay (unsigned long  ulVal)
{
    do {
    } while ( --ulVal != 0 );
}

int  main (void)
{
    unsigned char  ucVal;

    timeDelay(500000L);                                     /*  开机延迟                    */

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);            /*  使能 GPIOB 模块             */
    GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, PB0);             /*  设置 PB0 为输入模式         */
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, PB1);            /*  设置 PB1 为输出模式         */

    for (;;) {
        ucVal = GPIOPinRead(GPIO_PORTB_BASE, PB0);          /*  读取 PB0 的状态             */
        GPIOPinWrite(GPIO_PORTB_BASE, PB1, (ucVal << 1));   /*  将 PB0 的状态写入 PB1       */
        timeDelay(300);
    }
}
```

恭喜！到现在为止，您已经学会如何利用 Luminary Micro 公司《Stellaris 驱动库》编写最简单的 LM3S 系列单片机 C 语言应用程序了。