MSP430 单片机入门及编程思想介绍

MSP430 单片机的片内外设极为丰富，这些硬件资源需要大量的寄存器来配置功能。MSP430 单片机内部有数百个寄存器和近千个控制位，这就注定它不能像 51 那样完全靠人肉去记忆、靠脚趾头去数位数。也就是说 MSP430 单片机需要设置的寄存器多如牛毛。但是我们需要明白一点，单片机的设计者已经把所有能为用户完成的事都做完了，所预留的寄存器是必须由“主人”控制的各种开关。相比于设计实际硬件电路将会遇到的各种问题，没有什么比拨一下开关更简单的事了。所以，对于配置寄存器，我们应该心怀感激和庆幸，在理解单片机设计者的设计意图后，配置寄存器也就不难了。理解单片机设计者的意图可以借助于阅读教材或用户指导。

单片机的学习没有“XX 天学会”“XX 小时入门”的可能性，但也不是非得受过“高等”教育才能学。学单片机只要学会一点 C 语言的皮毛就行，基础知识在参考教材的第二章写的非常详细。

德州仪器(TI)推出的 MSP430 单片机是一种基于 RISC 的 16 位混合信号处理器，专为满足超低功耗需求而精心设计。MSP430 单片机将智能外设、易用性、低成本以及业界最低功耗等优异特性完美结合在一起，能满足数以万计应用的要求。

单片机（Single chip microcomputer）是一种集成电路芯片，是采用超大规模集成电路技术把具有数据处理能力的中央处理器 CPU、随机存储器 RAM、只读存储器 ROM、多种 I/O 口和中断系统、定时器/计数器等功能（可能还包括显示驱动电路、脉宽调制电路、模拟多路转换器、A/D 转换器等电路）集成到一块硅片上构成的一个小而完善的微型计算机系统，在工业控制领域广泛应用。

工程师完成了复杂集成电路的设计，为了满足用户对不同功能的需求，会预留出接口供用户操作。**接口就类似于开关，有单刀单掷的、单刀双掷的和单刀多掷的。**用户不需要知道具体电路结构，只需要掌握每个接口代表的含义即可，编程的任务就是对接口进行操作，实现满足自己需要的功能。

为了帮助同学理解，可以把工程师比喻成电工，集成电路就是分布在墙体内的电线，电工会预留出控制开关。用户不需要知道墙体内的电线是如何分布的，只要知道每个开关的功能是什么，就可以通过控制开关实现控制不同的电器。单片机中预留的接口和开关非常相似，开关有打开和合上两种状态，接口一般也只有 1 和 0 两种状态。单片机的功能更为强大，所以预留的接口也更多，初学者首先要做的工作就是了解每个接口代表的是什么含义。这样就可以根据自己的需求控制不同的接口。

由于单片机的功能非常丰富，接口也非常多，为了方便用户操作，所以工程师会把接口进行整合，就形成了寄存器。寄存器整合的原则是把相同功能的接口或把控制相同外设的接口放在一起，可见一个寄存器对应多个接口。MSP430 是 16 位单片机，因此，寄存器一般也是 16 位的。一般来说，一个接口由应寄存器的一个位控制，也存在一个接口由寄存器的两个位或三个位一起控制。16 位的寄存器对应 16 位的二进制数，每个位只有 1 和 0 两种状态。**用户编程的过程就是通过对寄存器进行赋值以达到控制接口功能的过程。**实现一个功能，一般需要对多个接口进行操作，可以认为这些接口是串联的，只要有一个接口没有操作，功能就不会实现。

下面以 MSP430F5529 单片机的 I/O 口为例进行讲解，I/O 口是单片机最简单的外设，同时也是应用最广泛的外设。I/O 接口全称是 Input / Output Interface，指输入/输出设备通道。主机与外界的信息交换可以通过 I/O 口进行，既然是交换，那么，就有输出方向和输入方向两种状态，而信息主要是由高电平(即 1 电平)和低电平(即 0 电平)两种状态表示。所以，为了控制 I/O 口，单片机为每个 I/O 口都预留了多个操作接口，其中，有一个接口是控制方向的操作接口，当该方向接口为高电平时，表示 I/O 口输出，当该方向接口为低电平时，表示 I/O 口为输入；有一个接口是控制输出电平的操作接口，当该输出接口状态为 1 时，I/O 口输出高电平，当该输出接口状态为 0 时，I/O 口输出低电平。

MSP430F5529 有 63 个 I/O 口，每个 I/O 口都可以设置成输出高电平或低电平的功能，也可以配置为输入功能，接收高电平或低电平。如果每个 I/O 口的接口都独立操作，将会有 126 个接口变量，这样就非常麻烦。为了便于操作，一般把 8 个 I/O 口作为一个组，用一个 8 位的变量(即寄存器)与之对应，变量的每一位对应一个 I/O 口的接口。这样仅需要一个输出控制寄存器和一个方向控制寄存器就可以完成对 8 个 I/O 口的操作。

比如 MSP430F5529 的 63 个 I/O 分别编号为 1、2、3、4、5、6、7、8、...、63，其中，1~8 这个 8 个 I/O 口组合成 P1 口，P1 口采用编号方式为 P1.0~P1.7，即 P1.0 对应编号为 1 的 I/O 口，P1.1 对应的是编号为 2 的 I/O 口，.....，P1.7 对应的是编号为 8 的 I/O 口；9~16 这 8 个 I/O 口对应 P2 口，P2 口采用的编号方式为 P2.0~P2.7，即 P2.0 对应编号为 9 的 I/O 口，P2.1 对应的是编号为 10 的 I/O 口，.....，P2.7 对应的是编号为 16 的 I/O 口。而操作时只需要对 P1.0~P1.7 和 P2.0~P2.7 进行操作，这也是单片机的编号方法。把控制 P1.0~P1.7 这 8 个 I/O 口的方向的接口组合成一个寄存器称为 P1DIR，把控制 P1.0~P1.7 这 8 个 I/O 口的输出电平的接口组成一个寄存器称为 P1OUT。通过这两个寄存器就可以控制 P1 口 8 个 I/O 口的输出高低电平的功能了。下面介绍与 P1 口相关寄存器中的 P1DIR 和 P1OUT，

其中 P1DIR 是方向控制寄存器，如图 1 所示。可见，P1DIR 是个 8 位二进制数，每一位可以控制一个 I/O 口的输入输出功能，所以，可以控制 P1 口 8 个 I/O 口的输入输出方向。

7	6	5	4	3	2	1	0
P1DIR							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 1 P1DIR 寄存器

对于 P1DIR，其第 0 位与 P1.0 口相对应，当该位为 1 时，表示 P1.0 口被配置为输出功能，当该位为 0 时，表示 P1.0 被配置为输入功能。其余的位也是同样的道理。如果 P1DIR = 0000 0000，那么，表示 P1.0~P1.7 这 8 个 I/O 口都被配置为输入方向的 I/O 口；如果 P1DIR = 1111 1111，那么，表示 P1.0~P1.7 这 8 个 I/O 口都被配置为输出方向的 I/O 口。如果想让 P1.0 这个 I/O 口是输出功能，P1.1~P1.7 这 7 个 I/O 口是输入功能，则只要使 P1DIR = 0000 0001 即可。相同的道理，P2DIR 对应 P2 的 8 个 I/O 口。

P1OUT 寄存器是输出控制寄存器，如图 2 所示。可见，P1OUT 是个 8 位二进制数，每一位可以控制一个 I/O 口是高电平还是低电平，所以，可以控制 P1 口 8 个 I/O 口是高电平还是低电平。

7	6	5	4	3	2	1	0
P1OUT							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 2 P1OUT 寄存器

对于 P1OUT，其第 0 位与 P1.0 口相对应，当该位为 1 时，表示 P1.0 口被配置为高电平，当该位为 0 时，表示 P1.0 被配置为低电平。其余的位也是同样的道理。如果 P1OUT = 0000 0000，那么，表示 P1.0~P1.7 这 8 个 I/O 口都被配置为低电平；如果 P1OUT = 1111 1111，那么，表示 P1.0~P1.7 这 8 个 I/O 口都被配置为高电平。如果想让 P1.0 这个 I/O 口是高电平，P1.1~P1.7 这 7 个 I/O 口是低电平，则只要使 P1OUT = 0000 0001 即可。相同的道理，P2OUT 对应 P2 的 8 个 I/O 口。

通过上述分析，如果想使用 P1.0 这个 I/O 口实现驱动 LED 的功能，则只需要使 P1.0 这个 I/O 口输出高电平即可。那么，需要做两个工作：

一、把 P1.0 口配置为输出方向；

二、把 P1.0 口配置为高电平。

分析过程：控制 P1.0 口方向的接口位于 P1DIR 的第 0 位，当 PIDIR 的第 0

位的值为 0 时，P1.0 口是输入方向，当 P1DIR 的第 0 位的值为 1 时，P1.0 口是输出方向，所以，需要让 P1DIR 的第 0 位值为 1。控制 P1.0 口电平的接口位于 P1OUT 的第 0 位，当 P1OUT 的第 0 位值为 0 时，P1.0 是低电平，当 P1OUT 的第 0 位值为 1 时，P1.0 是高电平，所以，需要让 P1OUT 的第 0 位值为 1，P1.0 口被配置为高电平。

可见，完成 P1.0 口输出高电平，只需要两句代码：

```
P1DIR = 0x01; /*0x 表示十六进制，0x01 = 0000 0001，把 P1.0 配置为输出*/
P1OUT = 0x01; /*把 P1.0 配置为高电平*/
```

当执行完这两句代码之后，寄存器 P1DIR 的第 0 位就被置为 1，寄存器 P1OUT 的第 0 位也被置为 1 了，P1.0 口实现了输出高电平的功能。所以，如果新建了工程，只需要把上面两句话复制到主函数中(注：void main()是主函数的标识，其后的大括号内的代码是主函数的具体实现代码)，然后编译下载就可以点亮与 P1.0 引脚相连的 LED 灯。反应在最小实验板(即租借的开发板)上的现象是红色的 LED 灯亮。

完整程序如下：

```
#include <msp430f5529.h>
void main()
{
WDTCTL = WDTPW + WDTHOLD;/*这句不能少*/
P1DIR = 0x01;
P1OUT = 0x01;
}
```

无论此前 P1DIR 和 P1OUT 的值为多少，执行完上述代码之后，P1DIR 的值和 P1OUT 的值分别如图 3 和图 4 所示：

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 3 P1DIR 的值

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

图 4 P1OUT 的值

也就是说 PIDIR 的第 0 位被赋值为 1，第 1 位至第 7 位全部被赋值为 0；P1OUT

的第 0 位被赋值为 1，第 1 位至第 7 位全部被赋值为 0。可见，在控制 P1.0 接口的同时，P1.1~P1.7 的接口也被改写了，但是，实际上我只想改写与 P1.0 对应的接口。

为了只对单独的一个 I/O 口对应的接口进行操作，而不影响其余 I/O 口的接口，就引入了“按位或”、“按位与”、“按位取反”、“按位异或”等运算符。这就是为什么同学们在看例程时，发现例程代码的形式如下：

① P1DIR |= 0x01;

② P1OUT &= ~0x01;

③ P2OUT ^= BIT0;

“按位或”的运算符是“|”，“按位与”的运算符是“&”，“按位取反”的运算符是“~”，“按位异或”的运算符是“^”。“按位或”和“或”不同，“按位或”的运算符号是“|”，是数值运算，两个数经过“按位或”结果还是一个数，而“或”的运算符号是“||”，是逻辑运算，两个数经过“或”结果作为逻辑判断条件，只有两种结果：“真”和“假”。“按位与”和“与”区别与此类似。

例如：0x01 | 0x00 = 0x01；

0x01 | 0x10 = 0x11；

0x01 || 0x00 = 真；

0x01 || 0x10 = 真；

下面分别介绍“按位或”运算、“按位与”运算、“按位取反” 运算和“按位异或”运算。在进行按位运算之前需要把参与运算的数化成二进制数，二进制数与十六进制数之间的转换见辅导材料。

（一）“按位或”运算符（|）

参加运算的两个对象，按二进制位进行“或”运算。

运算规则：0|0=0；　1|0=1；　0|1=1；　1|1=1；

可以发现：任何数和 0 或，其值不改变，和 1 或，其值变为 1。

例如:3|5 即 0000 0011 | 0000 0101 = 0000 0111，因此，3|5 的值得 7。

“按位或”运算的特殊作用：

（1）常用来对一个数据的某一位或某些位置 1。

方法：对于 X，如果想让 X 的某一位或某些位变为 1，而其余的位保持不变，那么，可以另选一个位数与 X 相同的数 Y，取 Y 对应的某一位或某些位为 1，其余的位为 0，然后把 X 与 Y 进行“按位或”运算即可。

例：将 X=10100000 的低 4 位置 1 ，用 X | 0000 1111 = 1010 1111 即可得到。

将 X=10100000 的第 0 位置 1 ，用 X | 0000 0001 = 1010 0001 即可得到。

（2）取一个数中指定位。

方法：对于 X，如果想看其中的某一位或某些位是什么值，那么可以另选一个位

数与 X 相同的数 Y，取 Y 对应的某一位或某些位为 0，其余的位为 1，然后把 X 与 Y 进行“按位或”运算即可。

例：取 X=10101110 的低四位，用 X & 1111 0000 = 1111 1110 即可得到；

取 X=10101110 的第 3 位，用 X & 1111 0111 = 1111 1111 即可得到；

（二）“按位与”运算符（&）

参加运算的两个数据，按二进制位进行“与”运算。

运算规则：0&0=0；　　1&0=0；　　0&1=0；　　1&1=1；

可以发现：任何数和 0 与，其值变为 0，和 1 与，其值不改变。

例如：3&5 即 0000 0011 & 0000 0101 = 0000 0001，因此，3&5 的值得 1。

“按位与”运算的特殊用途：

（1）清零。如果想将一个单元清零，即使其全部二进制位为 0，只要与一个各位都为零的数值相与，结果为零。如果想让一个单元中的某一位进行清零，只需要选一个和该单元相同长度的二进制数 Y，使所选的数 Y 的对应的某一位为 0，其余的位都为 1 即可。

例：

使 X=10101110 第 7 位清 0，用 X & 0111 1111 = X&0x7F = 0010 1110 即可得到；

使 X=10101110 第 0 位清 0，用 X & 1111 1110 = X&0xFE = 1010 1110 即可得到；

使 X=10101110 第 1 位清 0，用 X & 1111 1101 = X&0xFD = 1010 1100 即可得到；

（2）取一个数中指定位。

方法：对于 X，如果想看其中的某一位或某些位是什么值，那么可以另选一个位数与 X 相同的数 Y，取 Y 对应的某一位或某些位为 1，其余的位为 0，然后把 X 与 Y 进行“按位与”运算即可。

例：取 X=10101110 的低四位，用 X & 0000 1111 = 0000 1110 即可得到；

取 X=10101110 的第 3 位，用 X & 0000 1000 = 0000 1000 即可得到；

（三）取反运算符（~）

参加运算的一个数据，按二进制位进行取反运算。

运算规则：~1=0；　~0=1；

可以发现：对一个二进制数按位取反，即将 0 变 1，1 变 0。

使一个数的最低位为零，可以表示为：X&~1。

~1 的值为 1111 1111 1111 1110，再按“与”运算，最低位一定为 0。注意“~”运算符的优先级比算术运算符、关系运算符、逻辑运算符和其他运算符都高。

例：~3 = ~0000 0011 = 1111 1100

~1010 0011 1111 0000 = 0101 1100 0000 1111

（四）异或运算符（^）

参加运算的两个数据，按二进制位进行“异或”运算。“异或”，顾名思义，相异

为 1，相同为 0。（PS：对应的是“同或”，即相同为 1，相异为 0。）

运算规则：0^0=0；　1^0=1；　0^1=1；　1^1=0；

可以发现：任何数与 0 异或，其值不改变，与 1 异或，相当于取反。

“异或运算”的特殊作用：

（1）使特定位翻转

对于 X，如果想让其中的某一位或某些位进行翻转，那么可以另选一个位数与 X 相同的数 Y，取 Y 对应的某一位或某些位为 1，其余的位为 0，然后把 X 与 Y 进行“按位异或”运算即可。

例：X=10101110，使 X 低 4 位翻转，用 X ^ 0000 1111 = 1010 0001 即可得到。

（2）与 0 相异或，保留原值

例：X=10101110，X ^ 0000 0000 = 1010 1110。

至此，我们已经学习 5 中赋值运算：“按位或”运算、“按位与”运算、“按位取反” 运算、“按位异或”运算和“简单赋值”即“=”。在参考教材第 12 页也有介绍。我们学习 MSP430 单片机编程使用最多的就是这 5 种赋值运算。我们已经可以对下面几句话进行解读了。

① P1DIR |= 0x01;

② P1OUT &= ~0x01;

③ P2OUT ^= BIT0;

对于①：其功能是把 P1DIR 寄存器的第 0 位置为 1，即 P1.0 被配置为输出；

对于②：其功能是把 P1OUT 寄存器的第 0 位置位 0，即 P1.0 被配置为低电平；

对于③：其功能是把 P2OUT 寄存器的第 0 位取反。

补充：X &= a 等价于 X = X & a；

X |= a 等价于 X = X | a；

X ^= a 等价于 X = X ^ a；

BIT0 = 0x0001;　对应 Px.0

BIT1 = 0x0002;　对应 Px.1

BIT2 = 0x0004;　对应 Px.2

BIT3 = 0x0008;　对应 Px.3

BIT4 = 0x0010;　对应 Px.4

BIT5 = 0x0020;　对应 Px.5

BIT6 = 0x0040;　对应 Px.6

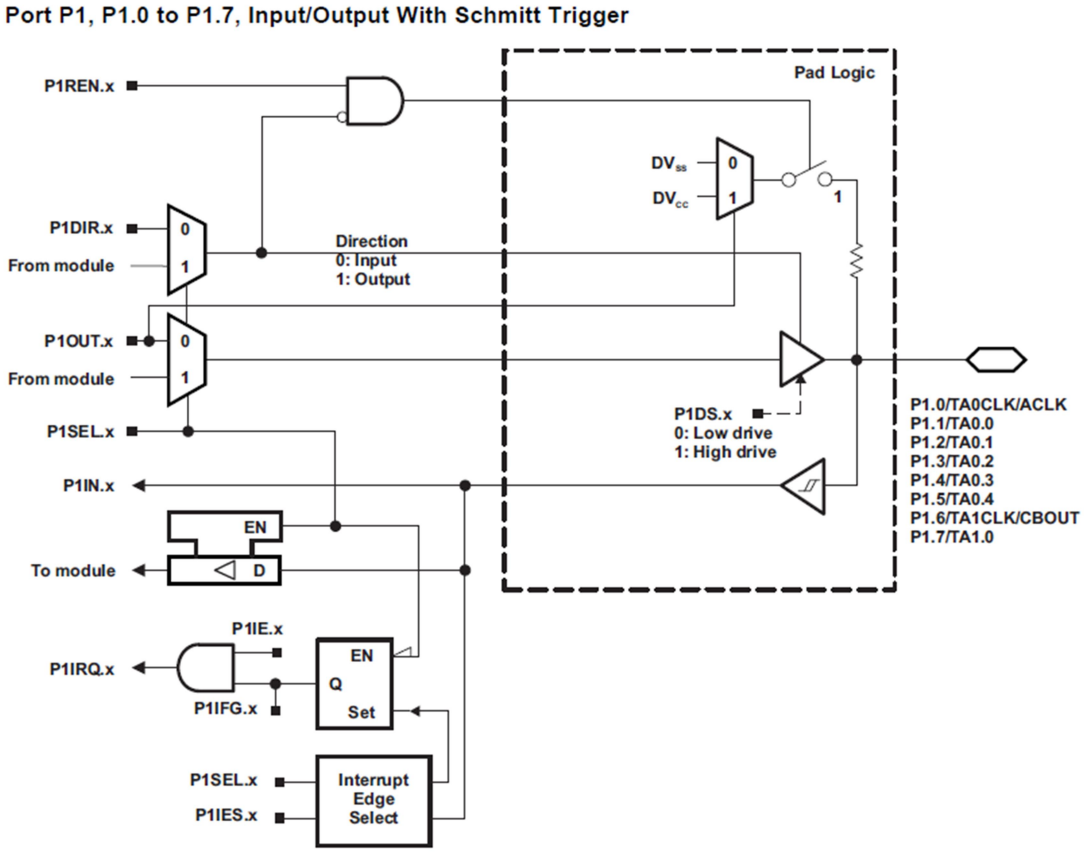
BIT7 = 0x0080;　对应 Px.7

其中，Px 可以是 P1，P2，P3，…，P8；也可以是 PA，PB（这部分先不了解）。

注意看门狗只能用“=”。

附图：

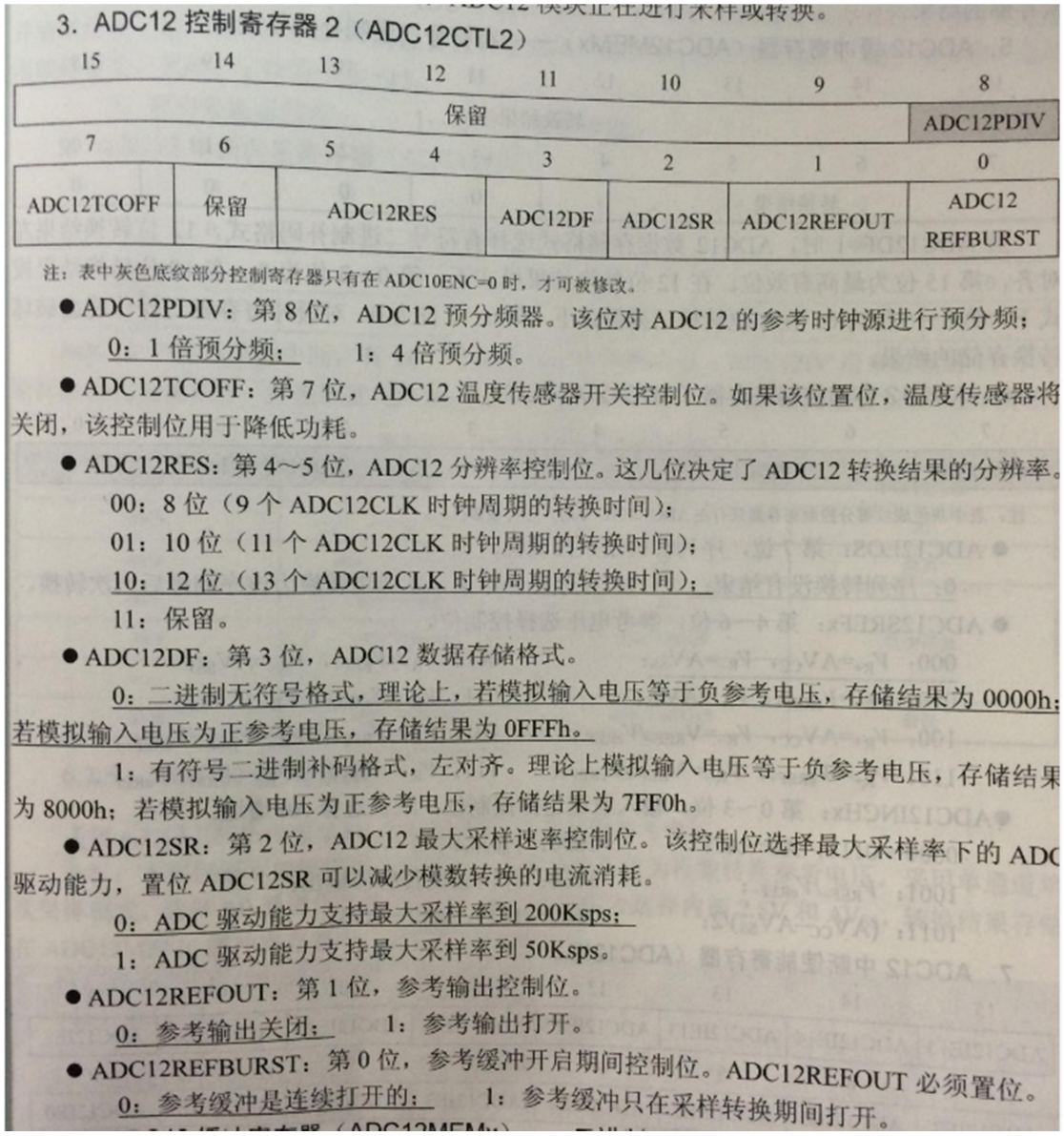
P1.0~P1.7 的 I/O 口电路如附图 1 所示，可见，I/O 预留的接口有 PIDIR.x、P1OUT.x、P1IN.x、P1REN.x、P1SEL.x、P1IE.x、P1IES.x 等操作接口。视频就是对着每个外设的电路图讲解每个接口的功能。书上有更详细的介绍！



附图 1 I/O 口电路图

AD 转换器的一个寄存器 ADC12CTL2 如附图 2 所示（参考课本截图），可以发现寄存器的每一位数值不同，所代表的功能介绍的已经非常详细，用户只要根据自己的需要对寄存器进行赋值就可以了。

有些同学反映说书看不懂，个人感觉那是因为你还不懂这本书的怎么用。如果你在 MSP430F5529 单片机面前是个小学生，那么 MSP430 单片机原理与应用这本书就像是一本新华字典。你可以在字典里面查你不认识的字，同样，这本书对单片机留出的每个接口也都要介绍！



3. ADC12 控制寄存器 2（ADC12CTL2）

15	14	13	12	11	10	9	8
保留							ADC12PDIV
7	6	5	4	3	2	1	0
ADC12TCOFF	保留	ADC12RES		ADC12DF	ADC12SR	ADC12REFOUT	ADC12 REFBURST

注：表中灰色底纹部分控制寄存器只有在 ADC10ENC=0 时，才可被修改。

● ADC12PDIV：第 8 位，ADC12 预分频器。该位对 ADC12 的参考时钟源进行预分频：

0：1 倍预分频；　　1：4 倍预分频。

● ADC12TCOFF：第 7 位，ADC12 温度传感器开关控制位。如果该位置位，温度传感器将关闭，该控制位用于降低功耗。

● ADC12RES：第 4～5 位，ADC12 分辨率控制位。这几位决定了 ADC12 转换结果的分辨率。

00：8 位（9 个 ADC12CLK 时钟周期的转换时间）；

01：10 位（11 个 ADC12CLK 时钟周期的转换时间）；

10：12 位（13 个 ADC12CLK 时钟周期的转换时间）；

11：保留。

● ADC12DF：第 3 位，ADC12 数据存储格式。

0：二进制无符号格式，理论上，若模拟输入电压等于负参考电压，存储结果为 0000h；若模拟输入电压为正参考电压，存储结果为 0FFFh。

1：有符号二进制补码格式，左对齐。理论上模拟输入电压等于负参考电压，存储结果为 8000h；若模拟输入电压为正参考电压，存储结果为 7FF0h。

● ADC12SR：第 2 位，ADC12 最大采样速率控制位。该控制位选择最大采样率下的 ADC 驱动能力，置位 ADC12SR 可以减少模数转换的电流消耗。

0：ADC 驱动能力支持最大采样率到 200Ksps；

1：ADC 驱动能力支持最大采样率到 50Ksps。

● ADC12REFOUT：第 1 位，参考输出控制位。

0：参考输出关闭；　　1：参考输出打开。

● ADC12REFBURST：第 0 位，参考缓冲开启期间控制位。ADC12REFOUT 必须置位。

0：参考缓冲是连续打开的；　　1：参考缓冲只在采样转换期间打开。

附图 2 ADC12CTL2 寄存器介绍