使用C语言操作DSP的寄存器

使用C语言操作DSP的寄存器

	C语言	汇编语言
效率	相对较低	高
可读性	高	低
可移植性	强	基本无
函数库	丰富	无

C语言和汇编语言对比

寄存器的C语言访问

由于DSP的寄存器能够实现对系统和外设功能的配置与控制，因此在DSP的开发过程中，对于寄存器的操作是极为重要的，也是很频繁的，也就是说对寄存器的操作是否方便会直接影响到DSP的开发是否方便。幸好，F28335为用户提供了位定义和寄存器结构体的方式，能够很方便地访问和控制内部寄存器。接下来，将以外设串行通信接口SCI（Serial Communication Interface）为例，详细介绍如何使用C语言的位定义和寄存器结构体的方式来实现对SCI寄存器的访问，在这个过程中，大家也可以了解到F28335的头文件是如何编写的。

寄存器的C语言访问·了解SCI的寄存器

寄存器名	地址单元格			大小（×16bit）	功能描述
	SCI-A	SCI-B	SCI-C		
SCICCR	0x0000 7050	0x0000 7750	0x0000 7770	1	通信控制寄存器
SCICTL1	0x0000 7051	0x0000 7751	0x0000 7771	1	控制寄存器1
SCIHBAUD	0x0000 7052	0x0000 7752	0x0000 7772	1	波特率寄存器高字节
SCILBAUD	0x0000 7053	0x0000 7753	0x0000 7773	1	波特率寄存器低字节
SCICTL2	0x0000 7054	0x0000 7754	0x0000 7774	1	控制寄存器2
SCIRXST	0x0000 7055	0x0000 7755	0x0000 7775	1	接收状态寄存器
SCIRXEMU	0x0000 7056	0x0000 7756	0x0000 7776	1	仿真缓冲寄存器
SCIRXBUF	0x0000 7057	0x0000 7757	0x0000 7777	1	接收数据缓冲寄存器
SCITXBUF	0x0000 7059	0x0000 7759	0x0000 7779	1	发送数据缓冲寄存器
SCIFFTX	0x0000 705A	0x0000 775A	0x0000 777A	1	FIFO发送寄存器
SCIFFRX	0x0000 705B	0x0000 775B	0x0000 777B	1	FIFO接收寄存器
SCIFFCT	0x0000 705C	0x0000 775C	0x0000 777C	1	FIFO控制寄存器
SCIPRI	0x0000 705F	0x0000 775F	0x0000 777F	1	优先权控制寄存器

TMS320F28335的SCI模块具有相同功能的串行通信接口SCI-A、SCI-B和SCI-C，也就是说体现到硬件上的话，F28335可支持三个串口。SCI-A、SCI-B和SCI-C就像三胞胎一样，具有相同的寄存器，如表所示。

寄存器的C语言访问·了解SCI的寄存器

从表中可看出，外设SCI的每一个寄存器都占据1个字节，即16位宽度。从其地址分布来看，SCI-A的寄存器地址从0x0000 7050到0x0000 705F，中间缺少了0x0000 7058、0x0000705D、0x0000705E。SCI-B的寄存器地址从0x0000 7750到0x0000 775F，中间缺少了0x0000 7758、0x0000775D、0x0000 775E。SCI-C的寄存器地址从0x0000 7770到0x0000 777F，中间缺少了0x0000 7758、0x0000775D、0x0000 775E。中间缺少的这些地址为系统保留的寄存器空间，暂时还没有使用。表3.1所列出的寄存器位于F28335存储器空间的外设帧2内，是在物理上实实在在存在的存储器单元。实际上，这些寄存器就是预定义了具体功能的存储单元，系统会根据这些存储单元具体的配置来进行工作。

寄存器的C语言访问·了解SCI的寄存器

在自然语言中去描述SCI-A寄存器SCICCR的某个位的时候，可以读作“SCIA的通信控制寄存器SCICCR的第x位”。这么读的时候第一反应是什么？这不是和C语言结构体成员的表述方式一样吗？说明SCI-A的寄存器可以采用结构体的方式来表示。

寄存器的C语言访问·使用位定义的方法定义寄存器

C语言中一种被称为“位域”或者“位段”的数据结构，即把一个字节中的二进制位划分为几个不同的区域，并说明每个区域的位数。每个域都有一个域名，允许在程序中按域名进行操作。位域的定义和位域变量的说明同结构体定义和其成员说明类似，其语法格式为：

```
Struct 位域结构名
{
  类型说明符 位域名1：位域长度;
  类型说明符 位域名2：位域长度;
  …
  类型说明符 位域名n：位域长度;
};
```

其中，类型说明符就是基本的数据类型，可以是int、char型等。位域名可以任意取，能够反映其位域的功能就好，位域长度是指这个位域是由多少个位组成的。和结构体定义一样，大括号最后的“;”不可缺少，否则会出错。

寄存器的C语言访问·使用位定义的方法定义寄存器

图3-1是将一个名为bs字的16位划分成了3个位域，其中D0~D7共8位为位域a，D8~D9共2位为位域b，D10~D15共6位为位域c，用位域的方式来定义的话如例1所示。

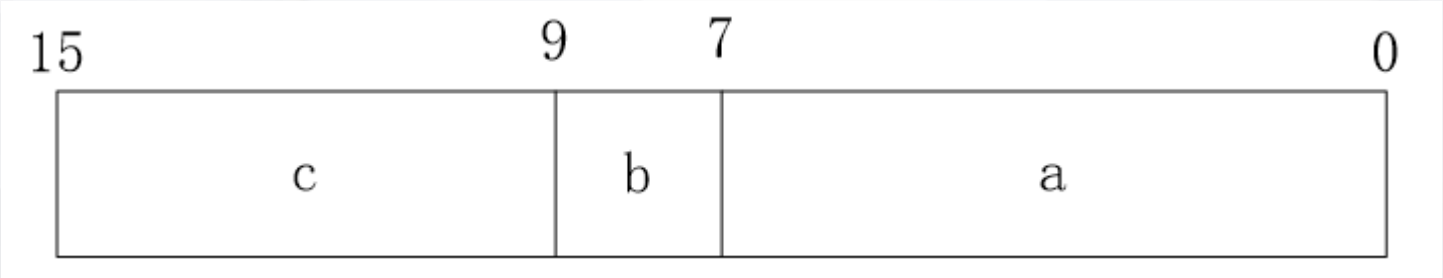

图3-1 bs的位域定义

【例3-1】 位域定义。

```
struct bs    //定义位域bs
{
  int a:8;
  int b:2;
  int c:6;
};
struct bs bs1; //声明bs型变量bs1
```

位域也是C语言中的一种数据结构，因此需要遵循先声明后使用的原则。例3-1中，声明了bs1，说明bs1是bs型的变量，共占2个字节，其中位域a占8位，位域b占2位，位域c占6位。

寄存器的C语言访问·使用位定义的方法定义寄存器

关于位域的定义还有以下几点说明：

1.位域的定义必须按从右往左的顺序，也就是说得从最低位开始定义。

2.一个位域必须存储在同一个字节中，不能跨两个字节。如果一个字节所剩空间不够放另一域时，应该从下一个单元起存放该域，如下所示：

```
struct bs
{
  int a:4;
  int :0;    //空域
  int b:5;   //从第二个字节开始存放
  int c:3;
};
```

在这个位域定义中，第一个位域a占第一个字节的4位，而第二个位域b占5位，很显然第一个字节剩下的4位不能够完全容纳位域b，所以第一个字节的后4位写0留空，b从第二个字节开始存放。

寄存器的C语言访问·使用位定义的方法定义寄存器

3.位域的长度不能大于一个字节的长度，也就是说一个位域不能超过8位。

4.位域可以无位域名，这时，它只用作填充或调整位置。无名的位域是不能使用的，如下所示：

```
struct bs
{
  int a:4;
  int :2;   //这2位不能使用
  int b:2;
  int c:5;
  int d:3;
};
```

寄存器的C语言访问·使用位定义的方法定义寄存器

掌握了C语言中位域的知识后，下面以SCI-A的通信控制寄存器SCICCR为例来说明如何使用位域的方法来定义寄存器。图3-2为SCI-A通信控制寄存器SCICCR的具体定义。

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICHAR2	SCICHAR1	SCICHAR0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

图3-2 SCI-A通信控制寄存器SCICCR

寄存器的C语言访问·使用位定义的方法定义寄存器

SCI-A模块所有的寄存器都是8位的，当一个寄存器被访问时，寄存器数据位于低8位，高8位为0。SCICCR的D0~D2为字符长度控制位SCICHAR，占据了3位。D3为SCI多处理器模式控制位ADDRIDLE_MODE，占据1位。D4为SCI回送从测试模式使能位LOOPBKENA，占据1位。D5为SCI极性使能位PARITYENA，占据1位。D6为SCI奇/偶极性使能位PARITY，占据1位。D7为SCI结束位的个数STOPBITS，也占据1位。D8~D15为保留，共8位。因此，可以将寄存器SCICCR用位域的方式表示为如例3-2所示的数据结构。

寄存器的C语言访问·使用位定义的方法定义寄存器

【例3-1】 位域定义。

```
struct  SCICCR_BITS
{
  Uint16 SCICHAR:3;              // 2:0    字符长度控制位
  Uint16 ADDRIDLE_MODE:1;   // 3      多处理器模式控制位
  Uint16 LOOPBKENA:1;          // 4      回送测试模式使能位
  Uint16 PARITYENA:1;          // 5      极性使能位
  Uint16 PARITY:1;               // 6      奇/偶极性选择位
  Uint16 STOPBITS:1;           // 7      停止位个数
  Uint16 rsvd1:8;               // 15:8    保留
};

struct SCICCR_BITS bit;
bit.SCICHAR=7;                    //SCI字符长度控制位为8位
```

寄存器的C语言访问·使用位定义的方法定义寄存器

在寄存器中，被保留的空间也要在位域中定义，只是定义的变量不会被调用，如例3-2中的rsvd1，为8位保留的空间。一般位域中的元素是按地址的顺序来定义的，所以中间如果有空间保留，那么需要一个变量来代替，虽然变量并不会被调用，但是必须要添加，以防后续寄存器位的地址混乱。

例3-2还声明了一个SCICCR_BITS的变量bit，这样就可以通过bit来实现对寄存器的位的访问了。例子中是对位域SCICHAR赋值，配置SCI字符控制长度为8位（SCICHAR的值为7，对应于字符长度为8位）。

寄存器的C语言访问·声明共同体

使用位定义的方法定义寄存器可以方便的实现对寄存器的功能位进行操作，但是有时候如果需要对整个寄存器进行操作，那么位操作是不是就显的有些麻烦了呢？所以很有必要引入能够对寄存器整体进行操作的方式，这样想要进行整体操作的时候就用整体操作的方式，想要进行位操作的时候就用位操作的方式。这种二选一的方式是不是想起C语言的共同体了呢？例3-3为对SCI的通信控制寄存器SCICCR进行共同体的定义，使得用户可以方便选择对位或者寄存器整体进行操作。

【例3-3】SCICCR的共同体定义

```
union SCICCR_REG
{
  Uint16 all;  //可实现对寄存器整体操作
  struct SCICCR_BITS bit; //可实现位操作
};
union SCICCR_REG SCICCR ;
SCICCR.all=0x007F;
SCICCR.bit.SCICHAR=5;
```

例3-3先是定义了一个共同体SCICCR_REG，然后声明了一个SCICCR_REG变量SCICCR，接下来变量SCICCR就可以对寄存器实现整体操作或者进行位操作，很方便。例3-3中，先是通过整体操作，对寄存器的各个位进行了配置，SCICHAR位被赋值为7，也就是说SCI数据位长度为8，紧接着，变量SCICCR通过位操作的方式，将SCICHAR的值改为5，即SCI数据的长度最终被设置为6。

寄存器的C语言访问·创建结构体文件

从前面的表3-1可以看出来，SCI模块除了寄存器SCICCR之外，还有许多的寄存器，为了便于管理，需要创建一个结构体，用来包含SCI模块的所有的寄存器，如例3-4所示。

```
struct  SCI_REGS
{
union SCICCR_REG     SCICCR;     //通信控制寄存器
union SCICTL1_REG    SCICTL1;    // 控制寄存器1
Uint16               SCIHBAUD;   // 波特率寄存器(高字节)
Uint16               SCILBAUD;   // 波特率寄存器(低字节)
union SCICTL2_REG    SCICTL2;    // 控制寄存器2
union SCIRXST_REG    SCIRXST;    // 接收状态寄存器
Uint16               SCIRXEMU;   // 接收仿真缓冲寄存器
union SCIRXBUF_REG   SCIRXBUF;   // 接收数据寄存器
Uint16  rsvd1;                   // 保留
Uint16               SCITXBUF;   // 发送数据缓冲寄存器
union SCIFFTX_REG    SCIFFTX;    // FIFO 发送寄存器
union SCIFFRX_REG    SCIFFRX;    // FIFO 接收寄存器
union SCIFFCT_REG    SCIFFCT;    // FIFO 控制寄存器
Uint16 rsvd2;                    // 保留
Uint16 rsvd3;                    // 保留
union SCIPRI_REG     SCIPRI;     // FIFO 优先级控制寄存器
};
extern volatile struct SCI_REGS SciaRegs;
extern volatile struct SCI_REGS ScibRegs;
extern volatile struct SCI_REGS ScicRegs;
```

寄存器的C语言访问·创建结构体文件

例3-4所示的SCI寄存器结构体SCI_REGS中，有的成员是union形式的，有的是Uint16形式的，定义为union形式的成员既可以实现对寄存器的整体操作，也可以实现对寄存器进行位操作，而定义为Uint16的成员只能直接对寄存器进行操作。

在3.1.1中提到过，无论是SCI-A、SCI-B，还是SCI-C，在其寄存器存储空间中，有3个存储单元是被保留的，在对SCI的寄存器进行结构体定义时，也要将其保留。如例3-4所示，保留的寄存器空间采用变量来代替，但是该变量不会被调用，如rsvd1、rsvd2、rsvd3。

寄存器的C语言访问·创建结构体文件

在定义了结构体SCI_REGS之后，需要声明SCI_REGS型的变量SciaRegs、ScibRegs、ScicRegs，分别用于代表SCI-A的寄存器、SCI-B的寄存器和SCI-C的寄存器。关键字extern的意思是“外部的”，表明这个变量在外部文件中被调用，是一个全局变量。关键字volatile的意思是“易变的”，使得寄存器的值能够被外部代码任意改变，例如可以被外部硬件或者中断任意改变，如果不使用关键字volatile，则寄存器的值只能被程序代码所改变。

寄存器的C语言访问·创建结构体文件

前面是以SCICCR为例来介绍如何使用位定义的方式表示某个寄存器，又以SCI模块为例来讲解如何用结构体文件来表示一个外设模块的所有寄存器，如果根据前面的介绍，将SCI所有的寄存器用位定义的方式来表示，然后根据需要来定义共同体，最后定义寄存器结构体文件，可以发现，原来这就是F28335的头文件DSP2833x_Sci.h的内容。现在明白头文件是怎么编写出来了吧，因为F28335的寄存器结构是固定的，因此，系统的头文件可以现成的拿来使用，一般情况下不需要再做修改了。

当如例3-4所示，定义了结构体SCI_REGS型的变量SciaRegs、ScibRegs和ScicRegs之后，就可以方便地实现对寄存器的操作了，下面以对SCI-A的寄存器SCICCR的操作为例，来介绍在开发程序时，是如何进行书写的。

【例3-5】对SCICCR按位进行操作。

```
SciaRegs.SCICCR.bit.STOPBITS=0;        //1位停止位
        SciaRegs.SCICCR.bit.PARITYENA=0;       //禁止极性功能
        SciaRegs.SCICCR.bit.LOOPBKENA=0;       //禁止回送测试模式功能
        SciaRegs.SCICCR.bit.ADDRIDLE_MODE=0;   //空闲线模式
        SciaRegs.SCICCR.bit.SCICHAR=7;         //8位数据位
```

【例3-6】对SCICCR整体进行操作。

```
SciaRegs.SCICCR.all=0x0007;
```

【例3-7】对SCIHBAUD和SCILBAUD进行操作。

```
SciaRegs.SCIHBAUD=0;
SciaRegs.SCILBAUD=0xF3;
```

由于SCIHBAUD和SCILBAUD定义时是Uint16型的，所以不能使用.all或者.bit的方式来访问了，只能直接给寄存器整体进行赋值。上面介绍的3种操作几乎涵盖了在F28335开发过程中对寄存器操作的所有方式，也就是说掌握了这3种方式，可以实现对F28335各种寄存器的操作了。

寄存器的C语言访问·创建结构体文件

CCS为用户书写程序时提供了非常方便的功能，譬如书写语句SciaRegs.SCICCR.bit.STOPBITS=0，先在CCS中输入SciaRegs，然后输入“.”，就会弹出一个下拉列表框，将SCI-A模块下所有的寄存器列了出来，如图3-3所示。单击列表框中寄存器SCICCR，便输入了寄存器SCICCR。在这里一定要注意的是，必须输入SciaRegs，每个字母的大小写都必须符合，否则是不会出现下拉列表框的。在输入SCICCR之后，继续输入成员操作符“.”，弹出新的下拉列表框，如图3-4所示。列表框中是共同体变量SCICCR的两个成员all或者bit。如果要对寄存器进行整体操作，就单击all，如果对寄存器进行位操作，就单击bit。在这里，选择单击bit，然后继续输入“.”，还是会弹出一个下拉列表框，里面列出了寄存器SCICCR的所有位域，也就是bit的所有成员，单击列表框中的STOPBITS，便完成了输入。

寄存器的C语言访问·创建结构体文件

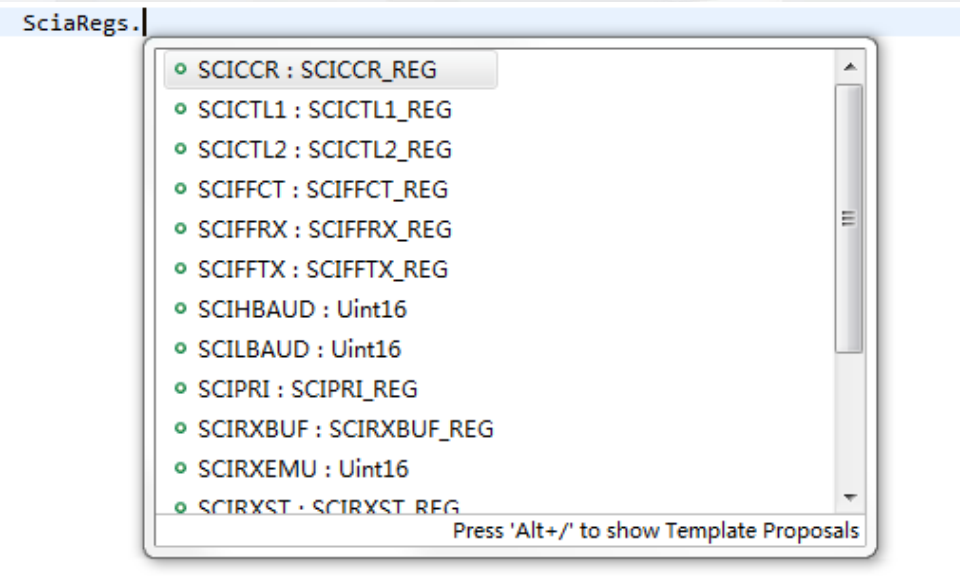

图3-3输入寄存器SCICCR

寄存器的C语言访问·创建结构体文件

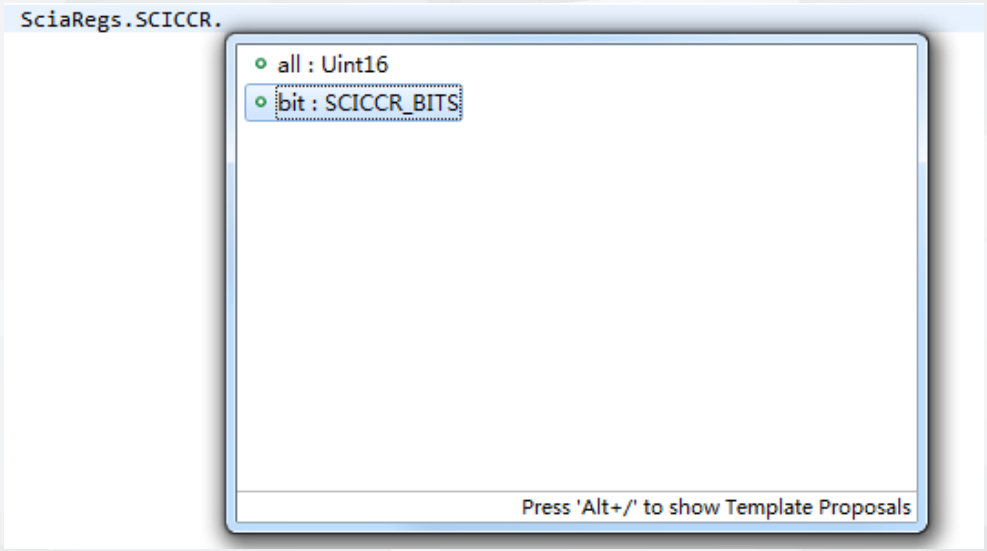

图3-4 输入bit

寄存器的C语言访问·创建结构体文件

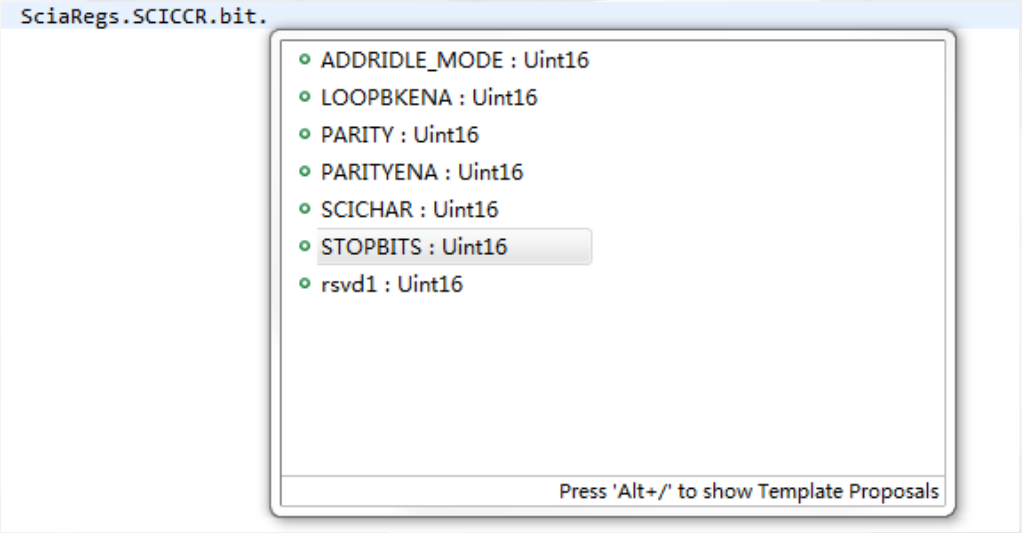

图3-5 输入位域STOPBITS

寄存器的C语言访问·创建结构体文件

这是CCS的感应功能，很显然，使用感应功能的前提是工程加载了F28335的头文件，其下拉列表框中的内容都是头文件中所定义的结构体或者共同体的成员。因为C语言是区分大小写的，所以在最先手动输入外设寄存器名字的时候一定要注意字母的大小写，否则CCS也无法感应。能够对寄存器的位域进行提示和操作是使用位定义和寄存器结构体方式访问寄存器最显著的优点。

如果选中SciaRegs.SCICCR，便可以观察到寄存器SCICCR的每一个位域的值，如图3-6所示，这也是使用位定义和寄存器结构体方式访问寄存器的优点，当然前提是程序已经执行了这条赋值语句。

寄存器的C语言访问·创建结构体文件

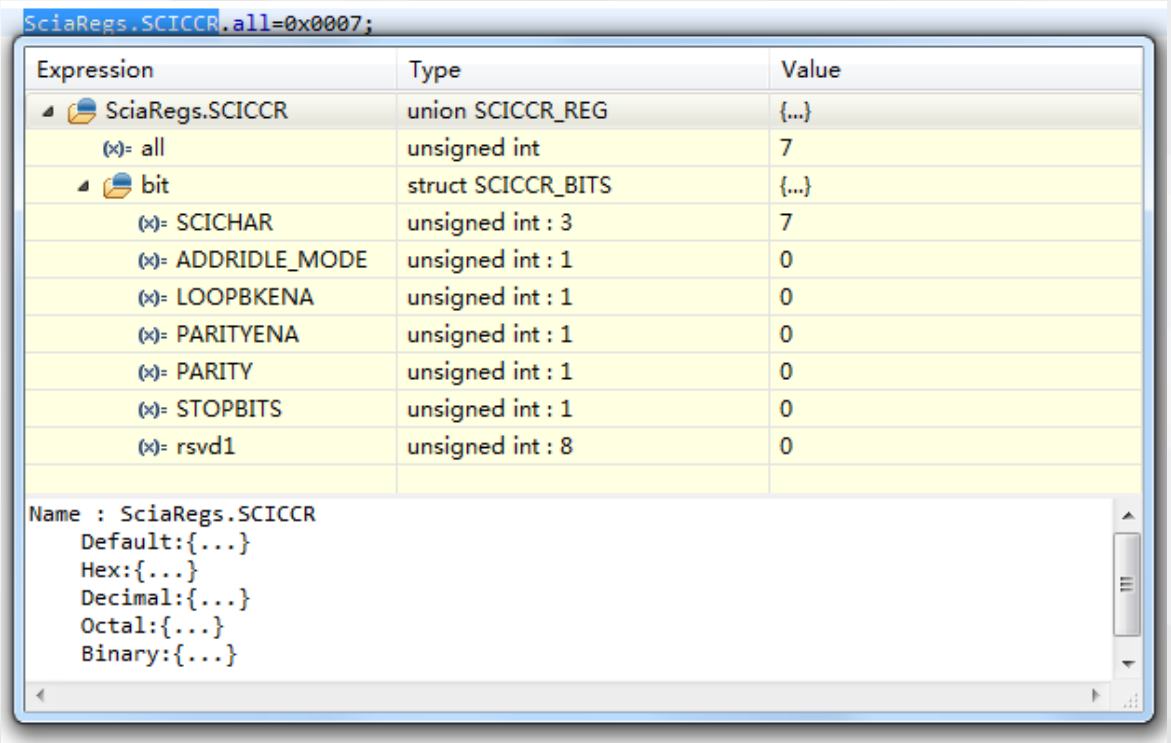

图3-6 在CCS中观察SCICCR

寄存器文件的空间分配

值得注意的是，之前所做的工作只是将F28335的寄存器按照C语言中位域定义和寄存器结构体的方式组织了数据结构，当编译时，编译器会把这些变量分配到存储空间中，但是很显然还有一个问题需要解决，就是如何将这些代表寄存器数据的变量同实实在在的物理寄存器结合起来呢？

这个工作需要两步来完成：第一步使用DATA_SECTION的方法将寄存器文件分配到数据空间中的某个数据段；第二步在CMD文件中，将这个数据段直接映射到这个外设寄存器所占的存储空间。通过这两步，就可以将寄存器文件同物理寄存器相结合起来了，下面详细讲解。

寄存器文件的空间分配

（1）使用DATA_SECTION方法将寄存器文件分配到数据空间

编译器产生可重新定位的数据和代码模块，这些模块就称为段。这些段可以根据不同的系统配置分配到相应的地址空间，各段的具体分配方式在CMD文件中定义。关于CMD文件，将在下一章节中详细讲解。在采用硬件抽象层设计方法的情况下，变量可以采用“# pragma DATA_SECTION”命令分配到特殊的数据空间。在C语言中，“# pragma DATA_SECTION”的编程方式如下：

```
# pragma DATA_SECTION (symbol,”section name”);
```

（1）使用DATA_SECTION方法将寄存器文件分配到数据空间

其中，symbol是变量名，而section name是数据段名。下面以变量SciaRegs和ScibRegs为例，将这两个变量分配到名字为SciaRegsFile和ScibRegsFile的数据段。

【例3-8】将变量分配到数据段。

```
#pragma DATA_SECTION(SciaRegs,"SciaRegsFile");
volatile struct SCI_REGS SciaRegs;
#pragma DATA_SECTION(ScibRegs,"ScibRegsFile");
volatile struct SCI_REGS ScibRegs;
```

（1）使用DATA_SECTION方法将寄存器文件分配到数据空间

例3-8其实是DSP2833x_GlobalVariableDefs.c文件中的一段，其作用就是将SciaRegs和ScibRegs分配到名字为SciaRegsFile和ScibRegsFile的数据段。CMD文件会将每个数据段直接映射到相应的存储空间里。表3-1说明了SCI-A寄存器映射到起始地址为0x0000 7050的存储空间。使用分配好的数据段，变量SciaRegs就会分配到起始地址为0x0000 7050的存储空间。那如何将数据段映射到寄存器对应的存储空间呢？这得研究一下CMD文件中的内容了。

（2）将数据段映射到寄存器对应的存储空间

【例3-9】将数据段映射到寄存器对应的存储空间

```
    /*********************************************************************
    * 存储器SRAM.CMD文件
    * 将SCI寄存器文件结构分配到相应的存储空间
    *********************************************************************/
    MEMORY
    {
PAGE 1 :
     SCI_A      : origin = 0x007050, length = 0x000010
     SCI_B      : origin = 0x007750, length = 0x000010
    }
    SECTIONS
    {
    SciaRegsFile      : > SCI_A,      PAGE = 1
      ScibRegsFile      : > SCI_B,      PAGE = 1
    }
```

（2）将数据段映射到寄存器对应的存储空间

从例3-9可以看到，首先在MEMORY部分，SCI_A寄存器的物理地址从0x007050开始，长度为16，SCI_B寄存器的物理地址从0x007750开始，长度也为16。然后在SECTIONS部分，数据段SciaRegsFile被映射到了SCI_A，而ScibRegsFile被映射到了SCI_B，实现了数据段映射到相应的存储器空间。

通过以上两部分的操作，才完成了将外设寄存器的文件映射到寄存器的物理地址空间上，这样才可以通过C语言来实现对F28335寄存器的操作。