MSP430程序库<八>DAC12的使用

MSP430 带有的 DAC12 模块，可以将运算处理的结果转换为模拟量，以便操作被控制对象的工作过程。DA 是在控制操作过程中常用的器件之一；MSP430有些系列中含有 DAC12 模块，给需要使用 DA 的方案提供了许多方便。这里实现较为简单的 DAC 的驱动，方便以后使用。

1.硬件介绍：

MSP430x14x 系列不含 DAC12模块，所以本文的实现只能用于16系列等含有 DAC12模块的单片机中。

MSP430F169 单片机的 DAC12 模块有2 个 DAC 通道，这两个通道在操作上是完全平等的。并且可以用 DAC12GRP 控制位将多个 DAC12通道组合起来，实现同步更新，硬件还能确保同步更新独立于任何中断或者 NMI 事件。

这个 DAC12模块有以下特点：8位或12位分辨率可调、可编程时间对能量的损耗、可选内部或外部参考源、支持二进制原码和补码输入、具有自校验功能、可以多路 DAC 同步更新、还可以用 DMA 等。

这里实现的是较为简化的版本，需要可以自己添加或改写功能，如：初始化函数内部调用自校验的函数，可以在每一次初始化时候均自校验。

DAC12每个模块只有两个寄存器：控制寄存和数据寄存器，控制寄存器用来初始化和设置模块的使用，数据寄存器用来存放要输出的电压数字量。169的 DAC 的寄存器如下：

DAC12_0控制寄存器　　DAC12_0CTL
DAC12_0数据寄存器　　DAC12_0DAT
DAC12_1控制寄存器　　DAC12_1CTL
DAC12_1数据寄存器　　DAC12_1DAT

控制寄存器每一位的功能如下：

DAC12REFx：选择 DAC12的参考源
0，1 Vref+
2，3 Veref+
DAC12RES：选择 DAC12分辨率
0 12位分辨率
1 8分辨率
DAC12LSELx：锁存器触发源选择
当 DAC12锁存器得到触发之后，能够将锁存器中的数据传送到 DAC12的内核。
当 DAC12LSELx=0的时候，DAC 数据更新不受 DAC12ENC 的影响。
0 DAC12_XDAT 执行写操作将触发（不考虑 DAC12ENC 的状态）
1 DAC12_XDAT 执行写操作将触发（考虑 DAC12ENC 的状态）
2 Timer_A3.OUT1的上升沿
3 Timer_B7.OUT2的上升沿
DAC12CALON：DAC12校验操作控制
置位后启动校验操作，校验完成后自动被复位。校验操作可以校正偏移误差。
0 没有启动校验操作
1 启动校验操作
DAC12IR：DAC12输入范围
设定输入参考电压和输出的关系

0 DAC12的满量程为参考电压的3倍（不操作 AVcc）
1 DAC12的满量程为参考电压
DAC12AMPx：DAC12运算放大器设置
0 输入缓冲器关闭，输出缓冲器关闭，高阻
1 输入缓冲器关闭，输出缓冲器关闭，0V
2 输入缓冲器低速低电流，输出缓冲器低速低电流
3 输入缓冲器低速低电流，输出缓冲器中速中电流
4 输入缓冲器低速低电流，输出缓冲器高速高电流
5 输入缓冲器中速中电流，输出缓冲器中速中电流
6 输入缓冲器中速中电流，输出缓冲器高速高电流
7 输入缓冲器高速高电流，输出缓冲器高速高电流
DAC12DF：DAC12的数据格式
0 二进制
1 二进制补码
DAC12IE：DAC12的中断允许
0 禁止中断
1 允许中断
DAC12IFG：DAC12的中断标志位
0 没有中断请求
1 有中断请求
DAC12ENC：DAC12转换控制位
DAC12LSEL>0的时候，DAC12ENC 才有效。
0 DAC12停止
1 DAC12转换
DAC12GRP：DAC12组合控制位
0 没有组合
1 组合

详细的有关 DAC12的资料可以参考 TI 提供的用户指南。

2.程序实现：

DAC12模块的程序比较简单，因为每组只有一个寄存器用来控制；本程序实现的功能如下：DAC 模块初始化，完成两个 DAC 模块的初始化，可以根据参数判断要是、初始化的是哪个模块或两个都初始化，或是两个一组同时更新；用参数传递 DAC12AMPx 的值，方便设置，程序中注释很详细，如果不理解，可以直接设 AMPx 为5或0x05；校准函数，完成 DAC12 模块的自校准，也是通过参数传递要校准的模块；电压输出函数，同样这个也是用参数传递要输出的模块。

初始化：

```
/*************************************************************
* 函数名称：DAC12Init
* 功    能：DAC12用到的相关资源初始化
* 参    数：
*           module 模块   0:使用模块 DAC12_0
*                         1:使用模块 DAC12_1
*                         2:使用模块 DAC12_0/1
```

```
*                       3:使用模块 DAC12_0/1 共同更新
*           DAC12AMPx：DAC 运算放大器设置：
*               0 输入缓冲器关闭，输出缓冲器关闭，高阻
*               1 输入缓冲器关闭，输出缓冲器关闭，0V
*               2 输入缓冲器低速/电流，输出缓冲器低速/电流
*               3 输入缓冲器低速/电流，输出缓冲器中速/电流
*               4 输入缓冲器低速/电流，输出缓冲器高速/电流
*               5 输入缓冲器中速/电流，输出缓冲器中速/电流
*               6 输入缓冲器中速/电流，输出缓冲器高速/电流
*               7 输入缓冲器高速/电流，输出缓冲器高速/电流
* 返 回 值：char，设置成功返回1，参数错误返回0
* 说    明：其他默认为：12位方案、写入即更新输出，module 模
*           块为3时，两个都写入更新；DAC12的满量程为参考电
*           压；内部2.5v 参考电压：需要 AD 设置参考源打开2.5.
*************************************************************/
charDAC12Init(charmodule,charDAC12AMPx)
{
    if(DAC12AMPx>7)
    {
        return(0);
    }
 //-------------------------设置模块-----------------------------
   switch(module)
    {
     case0:case'0': DAC12_0Init(DAC12AMPx); break;  //模块0
    case1:case'1': DAC12_1Init(DAC12AMPx); break;  //模块1
       case2:case'2':
            DAC12_0Init(DAC12AMPx);
            DAC12_1Init(DAC12AMPx);
         break;                                    //模块0、1
       case3:case'3':
            DAC12_0Init(DAC12AMPx);
            DAC12_1Init(DAC12AMPx);
            DAC12_0CTL |= DAC12GRP;
         break;  //无校验
    default:          return(0);                          //参数错误
   }
    return(1);
}
```

这里参数无效返回0，设置完成返回1，不过要注意的是在使用 DAC 之前，必须开启内部参考源（在 ADC 模块里面，具体可以参考使用示例）。

DAC12_0Init 和 DAC12_1Init 函数内容一样，只不过控制寄存器分别是 DAC12_0CTL 和 DAC12_0CTL，这里只给出 DAC12_0Init 的函数，另一个参考源程序：

```
voidDAC12_0Init(charDAC12AMPx)
```

```
{
 // Internal ref gain 1
   DAC12_0CTL = DAC12SREF_0 + DAC12IR;
    DAC12_0CTL |= DAC12LSEL_1 + (DAC12AMPx << 5);
    DAC12_0CTL |= DAC12ENC;
}
```

这个函数仅仅完成控制寄存器的设置。选内部参考源，输出满量程是参考电压的1倍，更新方式：写入即更新，如果 group 设置，则两个都写入才更新。

校准函数：完成 DAC12模块自校准，

```
voidDAC12Cal(charmodule)
{
    switch(module)
    {
        case0:case'0':
         DAC12_0CTL |= DAC12CALON;                  // 启动效验 DAC
        while((DAC12_0CTL & DAC12CALON) != 0);   // 等待效验完成
        break;                                    //模块0
       case1:case'1':
         DAC12_1CTL |= DAC12CALON;                  // 启动效验 DAC
        while((DAC12_1CTL & DAC12CALON) != 0);   // 等待效验完成
        break;                                    //模块1
       case2:case'2':
        case3:case'3':
         DAC12_0CTL |= DAC12CALON;                  // 启动效验 DAC
        while((DAC12_0CTL & DAC12CALON) != 0);   // 等待效验完成
        DAC12_1CTL |= DAC12CALON;                  // 启动效验 DAC
        while((DAC12_1CTL & DAC12CALON) != 0);   // 等待效验完成
        break;                                    //模块0、1
    default:        return;                     //参数错误
   }
}
```

参数含义和前初始化的函数相同，为了使用函数时一致。

输出函数：

```
voidDAC12Out(charmodule,intout)
{
    switch(module)
    {
     case0:case'0': DAC12_0DAT=out; break;       //模块0
    case1:case'1': DAC12_1DAT=out; break;       //模块1
       case2:case'2':
     case3:case'3': DAC12_0DAT=out; DAC12_1DAT=out; break;   //模块0、
1
    default:        return;                                 //参数错误
```

```
    }
}
```

输出函数的参数也和初始化的 module 参数含义相同，这个函数比较简单，只是按照要输出的值赋给 DAT 寄存器。

DAC12的程序库就这么多，DAC12还可以严格按时间更新数据，以输出一定频率的波形，可以设置为 TA out1上升沿更新数据，或 TB out2上升沿更新。另外还可以和 DMA 共同使用，完成更复杂的功能；这里均没有实现，需要的话可以根据寄存器功能来实现。

程序部分就到这了。

3.使用示例：

这里的使用方式依然和之前的程序一样，加入 C 文件，包含 H 文件即可，另外，使用本程序，必须开启内部 AD 参考源，为 DAC12模块提供参考电压。

```
voidmain( void)
{
  // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
     ClkInit();
  DAC12Init(3,5);                    //初始化
 DAC12Cal(2);                        //校准
 ADC12CTL0 = REF2_5V + REFON;      //开启内部参考源 2.5v 必须有；以供 DA 使用
    DAC12Out(2,0x666);
}
```

ADC12CTL0 = REF2_5V + REFON;这句即是开启参考电压2.5v 以供 DA 使用。

DAC12模块的程序库就到这里了。

相关文章及附件下载：http://www.ideyi.org/bbs/article_1077_371617.html