MSP430程序库<十四>DMA 程序库

直接存储器存取(DMADirect Memory Access)方式是用硬件实现存储器与存储器之间或存储器与 I\O 设备之间直接进行高速数据传送，不需要 CPU 的干预。这种方式通常用来传送数据块。MSP430f16x 系列单片机内部含有 DMA 模块，而且几乎内部所有外设都可以触发 DMA 开始存取数据。这里实现了这个模块的程序通用的函数库，方便使用。

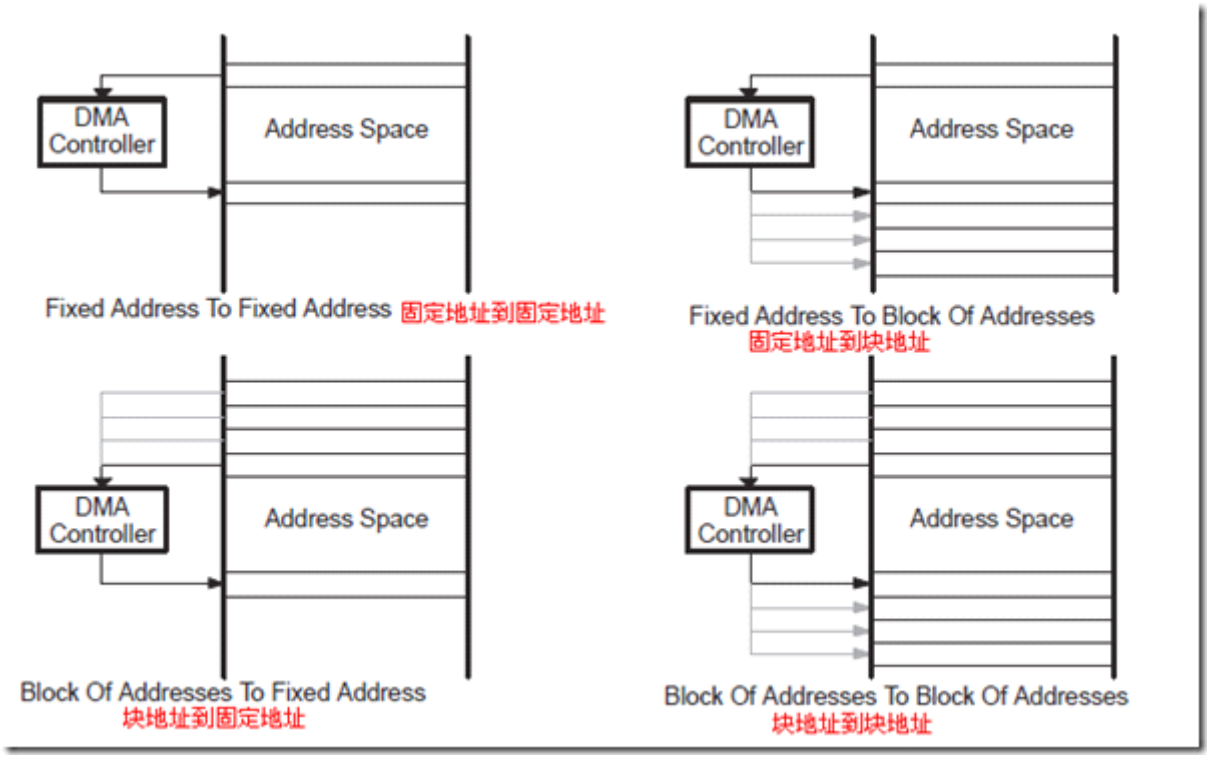
1.硬件介绍：

MSP430F15X/16X 系列单片机具有 DMA 控制器，从而能够为数据高速传输提供保证。例如，通过 DMA 控制器可以直接将 ADC 转换存贮器的内容传到 RAM 单元。

MSP430系列单片机扩展的 DMA 具有来之所有外设的触发器，不需要 CPU 的干预即可提供先进的可配置的数据传输能力，从而加速了基于 MCU 的信号处理进程，DMA 传输的触发来源对 CPU 来说是完全透明的，DMA 控制器可在内存与外部及外部硬件之间进行精确的传输控制。DMA 消除了数据传输延迟时间以及各种开销,从而可以解放16为 RISC CPU,以便其将更多的时间用于处理数据，而非执行正在处理的任务。

MSP430F16x 系列单片机的 DMA 模块有以下特点：数据传送不需要 CPU 介入，完全由 DMA 控制器自行管理。在整个地址空间范围内传输数据，块方式传输可达65536字节；能够提高片内外设数据吞吐能力，实现高速传输，每个字或者字节的传输仅需要2个 MCLK；减少系统功耗，即使在片内外设进行数据输入或输出时，CPU 也可以处于超低功耗模式而不需唤醒；字节和字数据可以混合传送：DMA 传输可以是字节到字节、字到字、字节到字或者字到字节。当字到字节传输时，只有字中较低字节能够传输，当从字节到字传输时,传输到字的低字节，高字节被自动清零；四种传输寻址模式：固定地址到固定地址、固定地址到块地址、块地址到固定地址以及块地址到块地址；触发方式灵活：边沿或者电平触发。单个、块或突发块传输模式：每次触发 DMA 操作，可以根据需要传输不同规模的数据

DMA 的四种寻址模式如下图所示：



DMA 控制器模块：3个独立的传输通道：通道0、通道1和通道2。每个通道都有源地址寄存器、目的地址寄存器、传送数据长度寄存器和控制寄存器。每个通道的触发请求可以分别允许和禁止；可配置的通道优先权：优先权裁决模块，传输通道的优先级可以调整，对同时有触发请求的通道进行优先级裁决，确定哪个通道的优先级最高。MSP430的 DMA 控制器可以采用固定优先级，还可以采用循环优先级。程序命令控制模块，每个 DMA 通道开始传输之前，CPU 要编程给定相关的命令和模式控制，以决定 DMA 通道传输的类型；可配置的传送触发器：触发源选择模块，DMAREQ（软件触发）、Timer_ACCR2输出、Timer_BCCR2输出、I2C 数据接收准备好、I2C 数据发送准备好、USART 接收发送数据、DAC12模块 DAC12IFG、ADC12模块的 ADC12IFGx、DMAxIFG、DMAE0 外部触发源。并且还具有触发源扩充能力。

DMA 有六种传输模式：单字或者单字节传输；块传输；突发块传输；重复单字或者单字节传输；重复块传输；重复突发块传输。前三个，传输完成后 DMAEN 自动复位；再次传输时需要重新置位 DMAEN 位以使能 DMA 通道。后三个为重复模式，一次传输完成后，DMAEN 不复位；再次出发时，可以再次启动数据传输。六种传输模式通过 DMADTx 寄存器设置：

DMADTx	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

单字或者单字节传输：DMA 通道被定义为单字或者单字节传输模式，每个字或者字节的传输都要触发信号触发。设置 DMADTx=0 就定义了单字或者单字节传输模式，规定的传输完毕后 DMAEN 位自动清除，如果需要再次传输，必须重新置位 DMAEN。如果设置

DMADTx＝4 为重复单字或者单字节传输模式，DMAEN 位一直保持置位，每次触发伴随一次传输。DMAxSZ 寄存器保存传输的单元个数，如果该寄存器为0，则没有传输。传输之前 DMAxSZ 寄存器的值写入到一个临时的寄存器中，每次操作之后 DMAxSZ 做减操作。当 DMAxSZ 减为零的时候，它所对应的临时寄存器将原来的值重新置入 DMAxSZ，同时相应的 DMAIFG 标志置位。

块传输模式：在块传输模式，每次触发可以传输一个数据块。设置 DMADTx=1 为块传输模式，每个数据块传输完毕，DMAEN 位自动清除，在触发传输下一个数据块之前，该位要被重新置位。在传输某个数据块期间，其他的传输请求将被忽略。设置 DMADTx=5 为重复块传输模式，某个数据块传输完毕，DMAEN 位仍然保持置位，之后，新的触发可以引起又一次数据块传送。DMAxSZ 寄存器保存数据块所包含的单元个数。DMASRCINCR 和 DMADSTINCR 反映在数据块传输过程中的目的地址和源地址的变化情况。在块传输或者重复块传输过程中，DMAxSA，DMAxDA，DMAxSZ 寄存器的值写入到对应的临时寄存器中，DMAxSA，DMAxDA 寄存器所对应的临时值在块传输过程中增加或者减少，而 DMAxSZ 在块传输过程中减计数，始终反映当前数据块还有多少单元没有传输完毕，当 DMAxSZ 减为0，它所对应的临时寄存器将原来的值重新置入 DMAxSZ，同时相应的 DMAIFG 被置位。在块传输过程中，CPU 暂停工作，不参与数据的传输。数据块需要 2×MCLK×DMAxSZ 个时钟周期。当每个数据块传输完毕，CPU 按照暂停前的状态重新开始执行。

突发块传输模式：这个和块传输模式类似，只不过每传输4个字或字节，DMA 释放内部总线，CPU 运行2个 MCLK 周期；在传输过程中 CPU 有20%的执行时间，而块传输需要等 DMA 完全传送完之后，CPU 方能运行。

DMA 触发源：每个通道的触发源有 DMAxTSELx 位进行控制的，这些位必须在 DMAEN 位为0是进行设置，否则可能出现不可预料的 DMA 触发。

DMAxTSELx	Operation
0000	DMAREQ bit (software trigger)
0001	TACCR2 CCIFG bit
0010	TBCCR2 CCIFG bit
0011	URXIFG0 (UART/SPI mode), USART0 data received (I2C mode)
0100	UTXIFG0 (UART/SPI mode), USART0 transmit ready (I2C mode)
0101	DAC12_0CTL DAC12IFG bit
0110	ADC12 ADC12IFGx bit
0111	TACCR0 CCIFG bit
1000	TBCCR0 CCIFG bit
1001	URXIFG1 bit
1010	UTXIFG1 bit
1011	Multiplier ready
1100	No action
1101	No action
1110	DMA0IFG bit triggers DMA channel 1 DMA1IFG bit triggers DMA channel 2 DMA2IFG bit triggers DMA channel 0
1111	External trigger DMAE0

另外，单片机的中断程序不影响 DMA 的传输，当 DMA 传输过程中，单片机不响应中外部 NMI 中断(必须 DMA 的控制位 ENNMI 位为1时响应 NMI 中断，否则不予处理)外的所有中断；必须等待 DMA 数据传送结束之后才运行系统的中断处理程序。

DMA 的中断：数据传送过程中，DMAxSZ 寄存器值减为0时，DMA 置位 DMAIFG，DMA 的中断和 DAC12模块共享中断向量，使用中断时需要软件判断具体是那个中断。中断响应后 DMAIFG 不会自动复位，使用时必须软件清零 DMAIFG 位。

DMA 的寄存器如下：

Register	Short Form	Register Type	Address	Initial State
DMA control 0	DMACTL0	Read/write	0122h	Reset with POR
DMA control 1	DMACTL1	Read/write	0124h	Reset with POR
DMA channel 0 control	DMA0CTL	Read/write	01E0h	Reset with POR
DMA channel 0 source address	DMA0SA	Read/write	01E2h	Unchanged
DMA channel 0 destination address	DMA0DA	Read/write	01E4h	Unchanged
DMA channel 0 transfer size	DMA0SZ	Read/write	01E6h	Unchanged
DMA channel 1 control	DMA1CTL	Read/write	01E8h	Reset with POR
DMA channel 1 source address	DMA1SA	Read/write	01EAh	Unchanged
DMA channel 1 destination address	DMA1DA	Read/write	01ECh	Unchanged
DMA channel 1 transfer size	DMA1SZ	Read/write	01EEh	Unchanged
DMA channel 2 control	DMA2CTL	Read/write	01F0h	Reset with POR
DMA channel 2 source address	DMA2SA	Read/write	01F2h	Unchanged
DMA channel 2 destination address	DMA2DA	Read/write	01F4h	Unchanged
DMA channel 2 transfer size	DMA2SZ	Read/write	01F6h	Unchanged

有关每个寄存器的详细内容参考 ti 提供的用户指南。

2.程序实现：

DMA 的使用主要是 DMA 寄存器的初始设置，设置完成后，DMA 接到触发信号即可自动传输数据。

设置函数如下：

```
voidDMAInit(charchannel,chartrigger,chartransMode,charsrcMode,cha
rdstMode,
              unsigned intsrc,unsigned intdst,unsigned intsize)
{
    unsigned int*DMAxCTL,*DMAxSA,*DMAxDA,*DMAxSZ;

    DMACTL0 = trigger << (channel << 2);
  DMACTL1 = 0x04;           //DMA 收到触发请求时，等待当前指令执行完成后

  switch(channel)          //选择当前设置哪个 DMA 通道
    {
        case0:
            DMAxCTL = (unsigned int*)&DMA0CTL;
            DMAxSA = (unsigned int*)&DMA0SA;
            DMAxDA = (unsigned int*)&DMA0DA;
            DMAxSZ = (unsigned int*)&DMA0SZ;
          break;                                    //指针 = 0通道控制
       case1:
            DMAxCTL = (unsigned int*)&DMA1CTL;
            DMAxSA = (unsigned int*)&DMA1SA;
            DMAxDA = (unsigned int*)&DMA1DA;
            DMAxSZ = (unsigned int*)&DMA1SZ;
          break;                                    //指针 = 1通道控制
       case2:
            DMAxCTL = (unsigned int*)&DMA2CTL;
            DMAxSA = (unsigned int*)&DMA2SA;
            DMAxDA = (unsigned int*)&DMA2DA;
            DMAxSZ = (unsigned int*)&DMA2SZ;
          break;                                    //指针 = 2通道控制
    }

  switch(transMode)        //设置 DMA 通道的传输模式
    {
      case'S': *DMAxCTL = DMADT_0;    break;          //单次传输
      case's': *DMAxCTL = DMADT_4;    break;          //重复单次传输
      case'B': *DMAxCTL = DMADT_1;    break;          //块传输
      case'b': *DMAxCTL = DMADT_5;    break;          //重复块传输
      case'I': *DMAxCTL = DMADT_2;    break;          //突发块传输 交错
      case'i': *DMAxCTL = DMADT_6;    break;          //重复突发块传输 交
错
    }

  *DMAxCTL |= (srcMode & 0x04) << 2;                  //源 字或字节
```

```
  *DMAxCTL |= (srcMode & 0x03) << 8;                    //源 地址改变方式

  *DMAxCTL |= (dstMode & 0x04) << 3;                    //目的 字或字节
  *DMAxCTL |= (dstMode & 0x03) << 10;                   //目的 地址改变方式

    *DMAxSA = src;
     *DMAxDA = dst;
     *DMAxSZ = size;

  *DMAxCTL |= DMAEN;                                    //DMA 使能
}
```

函数比较麻烦，函数内容按参数设置每个寄存器。DMACTL0 = trigger << (channel << 2)；这个是设置对应 channel 通道的的参考源，不大明白的可以看下 DMACTL0的寄存器内容；switch (channel)语句则根据通道设置对应指针指向的寄存器；然后对应设置参数即可。

当设置成非重复模式时，需要重新置位 DMAEN，本程序就函数 DMAReEnable 实现：

```
voidDMAReEnable(charchannel)
{
  switch(channel)          //使能对应通道
    {
      case0: DMA0CTL |= DMAEN;    break;       //0通道
     case1: DMA1CTL |= DMAEN;    break;       //1通道
     case2: DMA2CTL |= DMAEN;    break;       //2通道
    }
}
```

这个函数比较简单，只是根据传入参数设置对应通道的 DMAEN 位。

当设置为软件触发时，需要软件启动 DMA 程序如下：

```
voidDMAStart(charchannel)
{
  switch(channel)          //使能对应通道
    {
      case0: DMA0CTL |= DMAREQ;   break;       //0通道
     case1: DMA1CTL |= DMAREQ;   break;       //1通道
     case2: DMA2CTL |= DMAREQ;   break;       //2通道
    }
}
```

这个和上个函数类似：仅仅设置一个控制位，函数很简单，不再解释啦。

程序实现就这么多了，有关详细内容可以下载附件里的程序库，程序的注释很详细。

3.使用示例：

使用这个程序时，步骤和原来的相同：工程中加入 DMA.c 文件，然后源文件中包含 DMA.h 头文件即可。

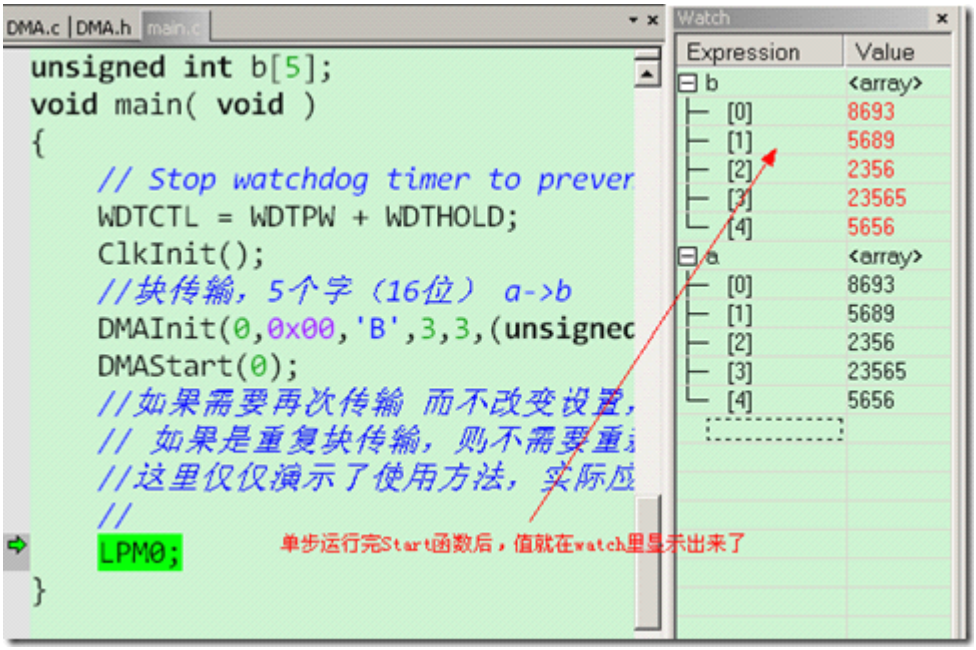
示例程序主要如下：

```
#include<msp430x16x.h>
#include"DMA.h"

unsigned inta[5] = {8693,5689,2356,23565,5656};
unsigned intb[5];
voidmain( void)
{
 // Stop watchdog timer to prevent time out reset
   WDTCTL = WDTPW + WDTHOLD;
    ClkInit();
 //块传输，5个字（16位） a->b
   DMAInit(0,0x00,'B',3,3,(unsigned int)a,(unsigned int)b,5);
    DMAStart(0);
 //如果需要再次传输 而不改变设置，只需调用 DMAReEnable 再次启动传输即可
 // 如果是重复块传输，则不需要重新使能 DMAReEnable 直接启动即可
 //这里仅仅演示了使用方法，实际应用中，应根据需要选择适当的触发源。
 //
   LPM0;
}
```

示例程序完成功能很简单，仅仅把一个数组的值赋给另外一个数组。数组地址即是数组名强制转换为所需类型(无符号16位)，传入函数初始化设置。这里为了简便，设置为软件启动。

运行效果如下：



单步运行完启动 DMA 传输后，结果即出来了；说明 DMA 传输数据的速度是很快的。

DMA 可以用于对速度要求比较高的程序中。例如：DMA 配合硬件乘法器和 ADC12模块，可以很容易的实现比较高频率的数字滤波方案。

相关文章及附件下载：http://www.ideyi.org/bbs/article_1077_374378.html