| | | | |
|---|---|---|---|
| | | | 当最后的LLI被读取，或者有通道错误，通道有效位被清除，通道也无效。如果通道不得不停止，并且不丢失通道FIFO的数据，则必须设置停止位以屏蔽进一步的DMA请求。必须轮询激活位直到它为0，也就是FIFO中没有剩余数据了。最后可以清除通道有效位了。 |

## 11.8 DMA 控制器应用举例

本节主要针对 DMA 控制器流控制下的内存到内存的处理，做举例说明。通过具体代码的分析，对 DMA 控制器功能的理解有所帮助。

下面是 ARM11 处理器中，内存到内存传输处理的具体代码实现：

DMAT_MemtoMem 功能函数

输入: None

输出: None

```
static void DMAT_MemtoMem(void)
{
    u32 i, csel, usel;
    u32 uTsize, uBurst, uCh;
    // DMA 控制器的四种方式选择
    Disp("\nSelect DMA Controller : 0:DMAC0, 1:DMAC1, 2:SDMAC0, 3:SDMAC1: ");
    csel=GetIntNum();
    Disp("\n");
    switch(csel)
    {
        case 0:                                                 //DMAC0 选择
                Disp("Selected DMAC 0 ..... \n");
                    DMAC_InitCh(DMA0, DMA_ALL, &oDmac0);
```

```
            INTC_SetVectAddr(NUM_DMA0,  Dma0Done);
            INTC_Enable(NUM_DMA0);
                break;
    case 1:                                          //DMAC1 选择
            Disp("Selected DMAC 1 ..... \n");
                DMAC_InitCh(DMA1, DMA_ALL, &oDmac1);
            INTC_SetVectAddr(NUM_DMA1,  Dma1Done);
            INTC_Enable(NUM_DMA1);
                break;
    case 2:                                          //SDMAC0 选择
            Disp("Selected SDMAC 0 ..... \n");
                DMAC_InitCh(SDMA0, DMA_ALL, &oDmac2);
            INTC_SetVectAddr(NUM_SDMA0, Sdma0Done );
            INTC_Enable(NUM_SDMA0);
                break;
    case 3:                                          //SDMAC1 选择
            Disp("Selected SDMAC 1 ..... \n");
                DMAC_InitCh(SDMA1, DMA_ALL, &oDmac3);
            INTC_SetVectAddr(NUM_SDMA1, Sdma1Done);
            INTC_Enable(NUM_SDMA1);
                break;
    default : Assert(0);
}
//通道选择
Disp("\nSelect Channel : 0:CH0, 1:CH1, 2:CH2, 3:CH3, 4:CH4, 5:CH5, 6:CH6, 7:CH7      :
");
usel=GetIntNum();
Disp("\n");
switch(usel)
```

```
    {
        case 0:
                uCh= DMA_A;
                     break;
        case 1:
                uCh= DMA_B;
                     break;
        case 2:
                uCh= DMA_C;
                    break;
        case 3:
                uCh = DMA_D;
                     break;
        case 4:
                uCh = DMA_E;
                     break;
        case 5:
                uCh = DMA_F;
                     break;
        case 6:
                uCh = DMA_G;
                     break;
        case 7:
                uCh = DMA_H;
                     break;
            default : Assert(0);
    }
//传输宽度选择
 Disp("\nSelect Transfer Width : 0:BYTE, 1:HWORD, 2:WORD: ");
```

```
usel=GetIntNum();
Disp("\n");
switch(usel)
{
      case 0:                                          //字节选择
              uTsize = BYTE;
                   break;

      case 1:                                          //半字选择
              uTsize = HWORD;
                   break;
      case 2:                                          //字选择
              uTsize = WORD;
                   break;
      default : Assert(0);
  }
//脉冲大小选择
Disp("\nSelect Burst Size : 0:SINGLE, 1:BURST4, 2:BURST8, 3:BURST16, 4:BURST32, 5:BURST64,
     6:BURST128, 7:BURST256: ");
usel=GetIntNum();
Disp("\n");
switch(usel)
{
      case 0:
              uBurst = SINGLE;
                   break;
      case 1:
              uBurst = BURST4;
                   break;
```

```
        case 2:
                uBurst = BURST8;
                     break;
        case 3:
                uBurst = BURST16;
                     break;
        case 4:
                uBurst = BURST32;
                     break;
        case 5:
                uBurst = BURST64;
                     break;
        case 6:
                uBurst = BURST128;
                     break;
        case 7:
                uBurst = BURST256;
                     break;
        default : Assert(0);
        }
// 传输大小选择
Disp("\nSelect Transfer Size [1～0x200_0000] :  ");
uDataCnts=GetIntNum();
Disp("\n");

// 0.清除 rx/tx 缓冲器
   for (i = 0; i<(uDataCnts*uTsize+16); i++)
   {
       *(u8 *)(uRxBuffAddr+i) = 0;
```

```
        *(u8 *)(uTxBuffAddr+i) = 0;
    }

 // 1.建立 tx 缓冲器
    for (i = 0; i<uDataCnts*uTsize; i++)
        *(u8 *)(uTxBuffAddr+i) = (u8)(i+2)%0xff;
        switch(csel)
        {
           case 0 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
   OpMode(DEMAND), Src Req, Dst Req, Burst
              DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                        uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac0);
              DMACH_Start(&oDmac0);
                    break;
          case 1 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
   OpMode(DEMAND), Src Req, Dst Req, Burst
             DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                       uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac1);
              DMACH_Start(&oDmac1);
                    break;
         case 2 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
   OpMode(DEMAND), Src Req, Dst Req, Burst
              DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                        uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac2);
              DMACH_Start(&oDmac2);
                    break;
```

```
        case 3 :
// Channel, LLI_Address, SrcAddr, Src Type, DstAddr, Dst Type, Transfer Width, Transfer Size,
   OpMode(DEMAND), Src Req, Dst Req, Burst
              DMACH_Setup((DMA_CH)uCh, 0x0, uTxBuffAddr, 0, uRxBuffAddr, 0, (DATA_SIZE)uTsize,
                        uDataCnts, DEMAND, MEM, MEM, (BURST_MODE)uBurst, &oDmac3);
              DMACH_Start(&oDmac3);
                   break;
          default :
              Assert(0);
      }

   while(g_DmaDone==0);                              // Int.
   //while (!DMAC_IsTransferDone(&oDmac0));          // Polling
    if (CompareDMA(uTxBuffAddr, uRxBuffAddr, (DATA_SIZE)uTsize, uDataCnts))
        Disp(" >> Test Tx&Rx -> Ok << \n");
    else
    {
        Disp(" >>*** Tx-data & Rx-data mismatch ***<< \n");
     }

    switch(csel)
    {
        case 0:
                INTC_Disable(NUM_DMA0);
                DMAC_Close(DMA0, DMA_ALL, &oDmac0);
                    break;

        case 1:
                INTC_Disable(NUM_DMA1);
```

```
            DMAC_Close(DMA1, DMA_ALL, &oDmac1);
                 break;
        case 2:
            INTC_Disable(NUM_SDMA0);
            DMAC_Close(SDMA0, DMA_ALL, &oDmac2);
                 break;
        case 3:
            INTC_Disable(NUM_SDMA1);
            DMAC_Close(SDMA1, DMA_ALL, &oDmac3);
                 break;
        default : Assert(0);
        }
}
```

# 12 矢量中断控制器

本节主要描述 S3C6410X　RISC 微处理器内的矢量中断控制器的功能及用途。

## 12.1 概述

S3C6410X 内的中断控制器由 2 个 VIC(矢量中断控制器，ARM PrimeCell PL192)和 2 个 TZIC(TrustZone 中断控制器，sp890)组成。两个矢量中断控制器和两个 TrustZone 中断控制器链接在一起支持 64 位中断源。

S3C6410X 内的矢量中断控制器的性能如下：

（1）每个 VIC 支持 32 位的矢量 IRP 中断

（2）支持固定硬件中断优先级和可编程中断优先级

（3）支持硬件中断优先级屏蔽和可编程中断优先级屏蔽

（4）产生 IRQ 和 FIQ 中断

（5）产生软件中断

（6）raw 中断状态

（7）中断请求状态

（8）支持限制访问的特权模式

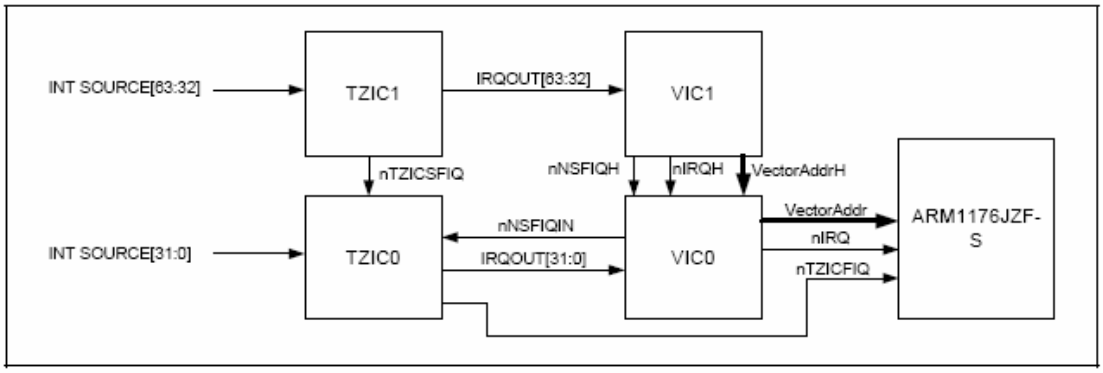


图 12-1　S3C6410X 的中断控制器

## 12.2 中断源

S3C6410X 支持 64 位中断源，不支持 ARM1176HZF-S 镜像中断运行.

| 中断号 | 中断源 | 描述 | 组 |
|---|---|---|---|
| 63 | INT_ADC | ADC EOC 中断 | VIC1 |
| 62 | INT_PENDNUP | ADC 笔向下/向上中断 中断 | VIC1 |
| 61 | INT_SEC | 安全中断 | VIC1 |
| 60 | INT_RTC_ALARM | RTC 警告中断 | VIC1 |
| 59 | INT_IrDA | IrDA 中断 | VIC1 |
| 58 | INT_OTG | USB OTG 中断 | VIC1 |
| 57 | INT_HSMMC1 | HSMMC1 中断 | VIC1 |
| 56 | INT_HSMMC0 | HSMMC0 中断 | VIC1 |
| 55 | INT_HOSTIF | 主机接口中断 | VIC1 |
| 54 | INT_MSM | MSM 调制解调器 I/F 中断 | VIC1 |
| 53 | INT_EINT4 | 外部中断组 1~组 9 | VIC1 |
| 52 | INT_HSIrx | HS Rx 中断 | VIC1 |
| 51 | INT_HSItx | HS Tx 中断 | VIC1 |
| 50 | INT_I2C0 | I2C 0 中断 | VIC1 |
| 49 | INT_SPI/INT_HSMMC2 | SPI 中断或 HSMMC2 中断 | VIC1 |
| 48 | INT_SPI0 | SPI0 中断 | VIC1 |
| 47 | INT_UHOST | USB 主机中断 | VIC1 |
| 46 | INT_CFC | CFCON 中断 | VIC1 |
| 45 | INT_NFC | NFCON 中断 | VIC1 |
| 44 | INT_ONENAND1 | 板块 1 的 ONENANE 中断 | VIC1 |
| 43 | INT_ONENAND0 | 板块 0 的 ONENAND 中断 | VIC1 |
| 42 | INT_DMA1 | DMA1 中断 | VIC1 |
| 41 | INT_DMA0 | DMA0 中断 | VIC1 |
| 40 | INT_UART3 | UART3 中断 | VIC1 |