第八章 DMA技术

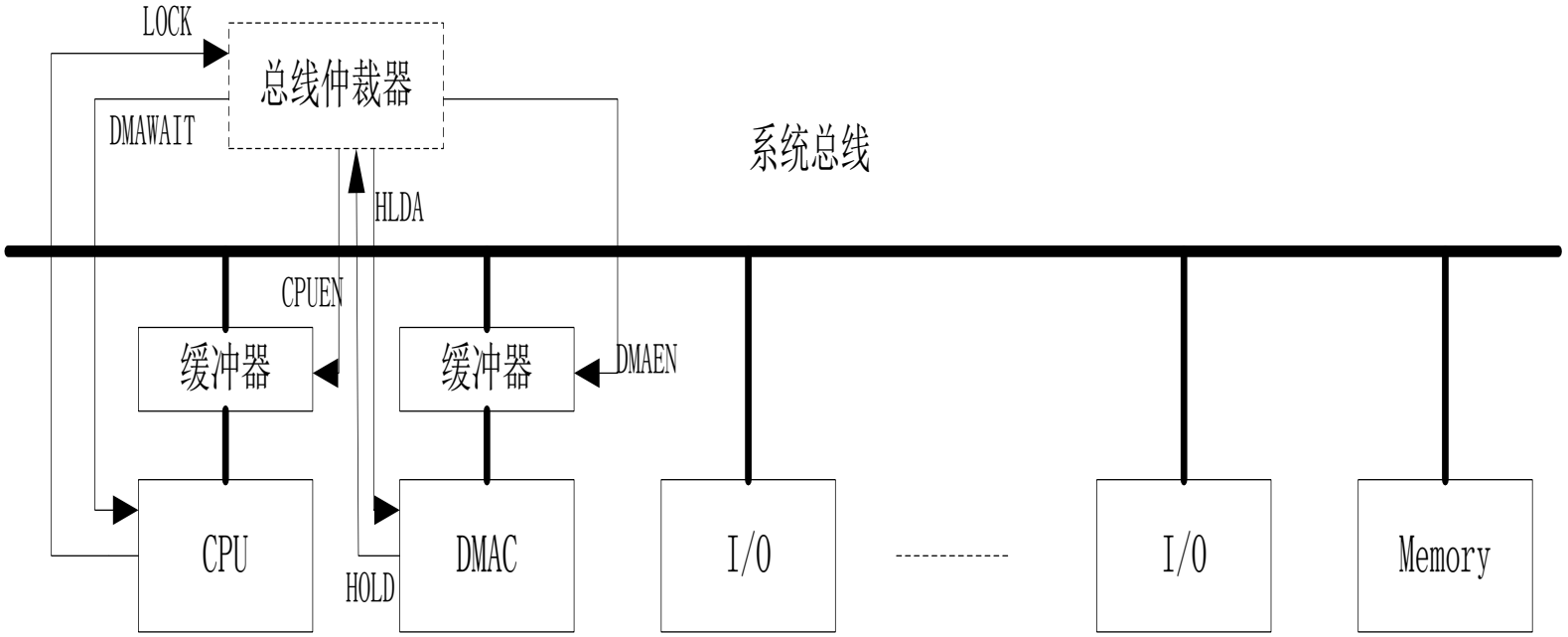
学习目标

- 了解DMA传送的基本原理
- 了解DMA传送流程
- 了解通道的基本原理
- 了解DMAC 8237A的工作原理(适用于通用PC机）
- 掌握Xilinx XPS DMA控制器的使用（适用于嵌入式）
- 掌握DMA传输初始化编程

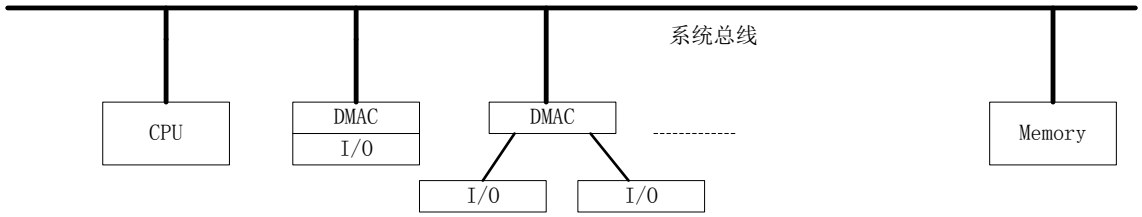
8.1 DMA传送基本原理

1) DMA 传输计算机系统构成
2) DMA 传输步骤
3) DMA 传输方向
4) DMA 传输模式

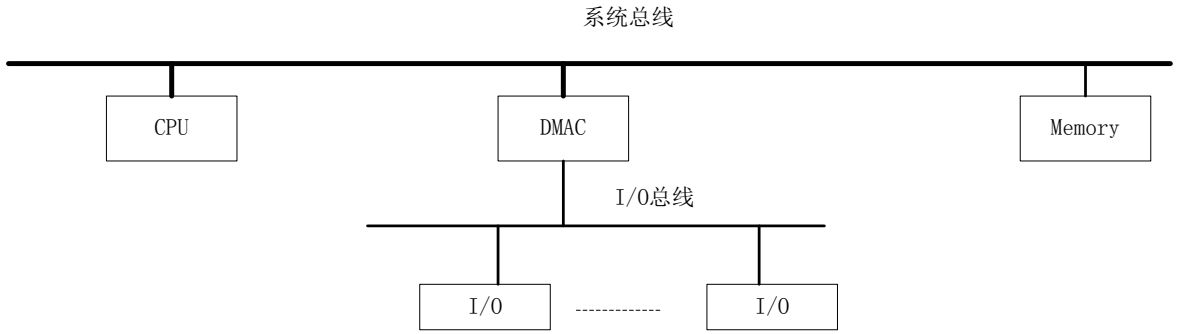
DMA 传输计算机系统构成



DMA控制器与IO接口集成

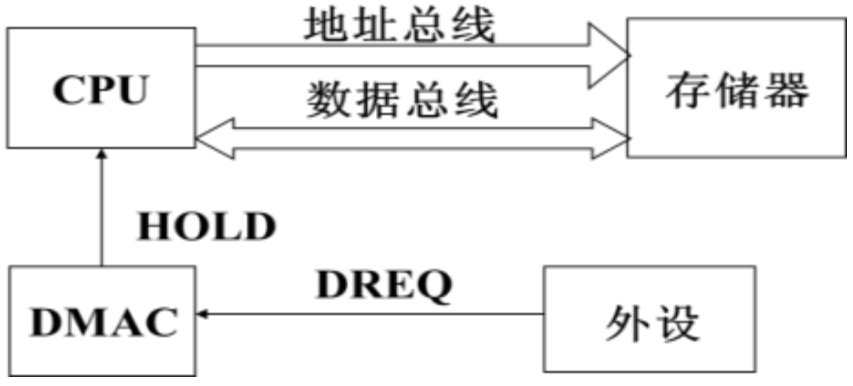


DMA控制器提供专门I/O总线

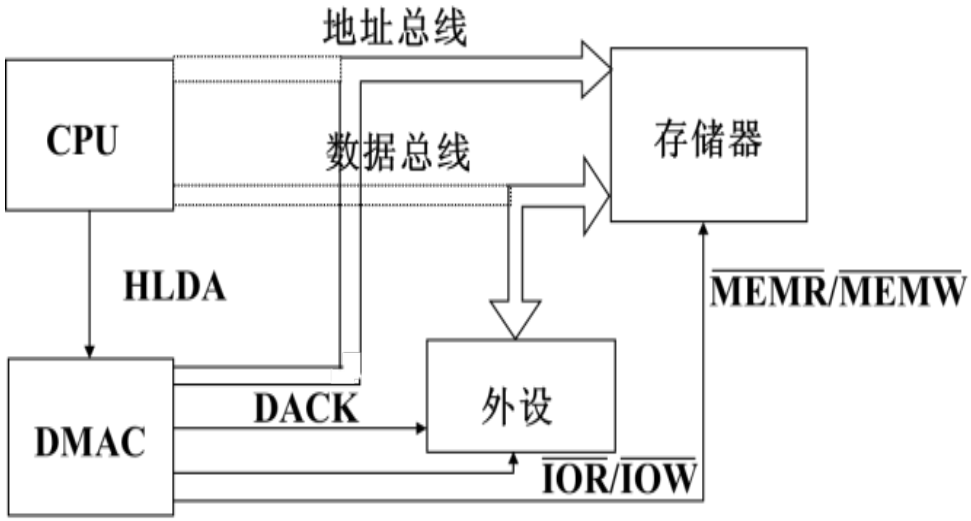


DMA传输步骤

- DMA请求阶段



- DMA响应及传输阶段



- DMA过程结束时，DMAC向CPU发出结束信号INT（撤消HOLD请求），将总线控制权交还CPU。

DMA传输方向

- I/O接口到存储器、
- 存储器到I/O接口、
- 存储器到存储器

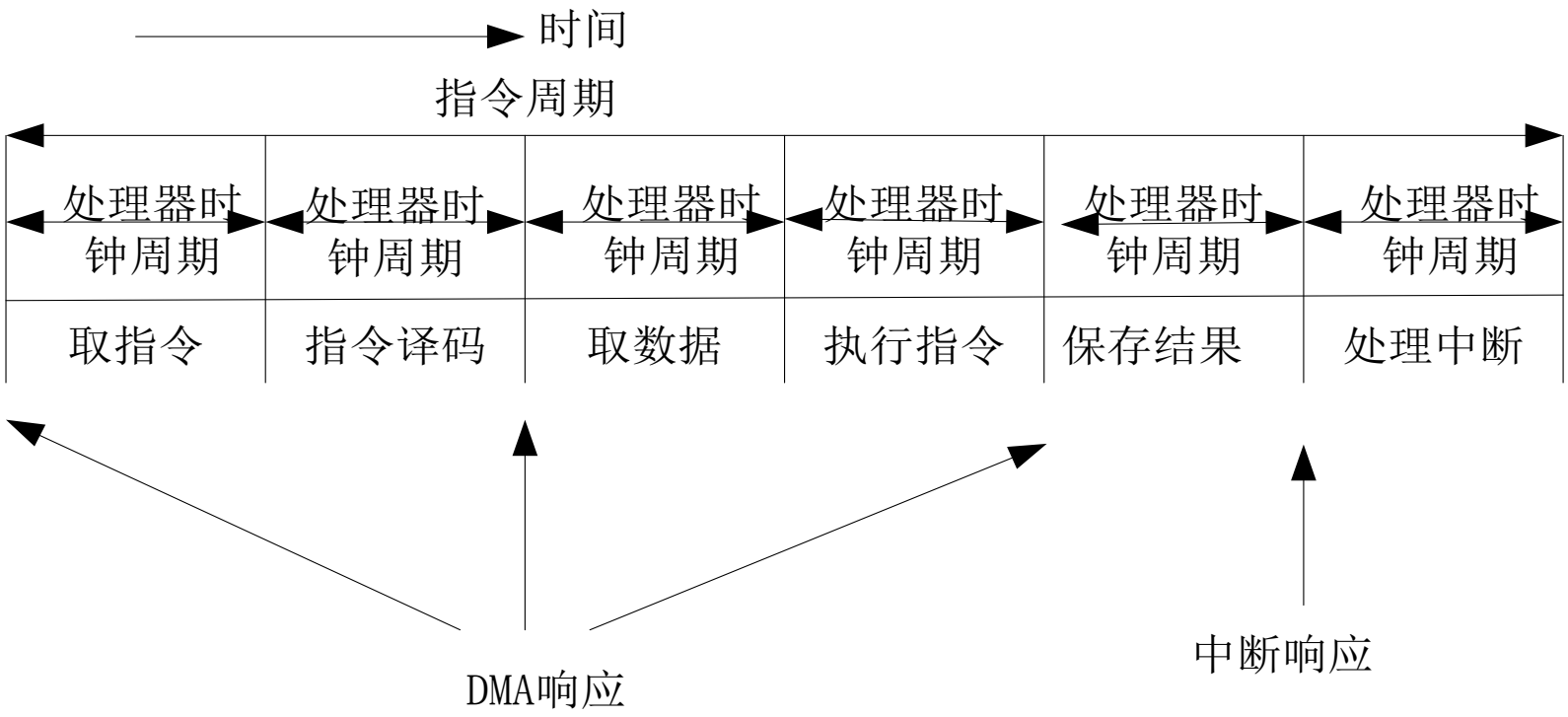
DMA传输模式

- 单字节传输模式：每次DMA操作传送一个字节后，接着释放总线。
- 块传输模式：连续传送多个字节，每传输一个字节，当前字节计数器减1，当前地址寄存器加1或减1，直到所要求的字节数传输完（当前字节计数器减至0），然后释放总线。
- 请求传输模式：DMAC要检测DREQ信号（询问外设），当DREQ为低时，暂停传输（不释放总线），当DREQ再次有效后，继续进行传输。
- 级联传输模式：多片DMAC级联时，可以构成主从式DMA系统。级联的方式是把从片的请求线HOLD连至主片的DREQ引脚，主片的DACK联至从片的HLDA引脚。若主DMAC的某通道（DREQ）连接从DMAC的HOLD，主DMAC的该通道应设置为级联传输模式，但从DMAC不设置级联传输模式，而是设置其它三种模式之一。

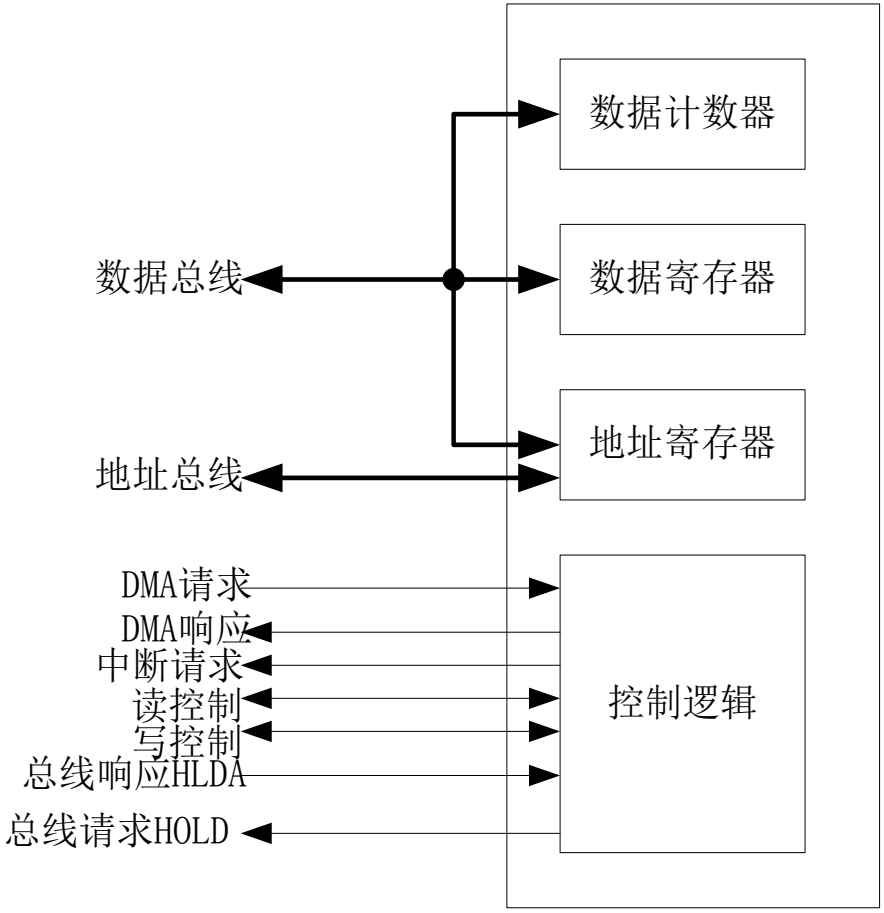
DMA传送基本流程

- 初始化阶段：在此阶段实现对DMA控制器的初始化，如配置数据传送模式，数据传送方向，内存区域的首地址、地址是递增还是递减、传送的总字节数等。
- DMA传送请求阶段：请求DMA传送由外设产生DREQ信号或者由CPU向DMAC写入启动DMA传送控制字，DMAC向CPU发出总线请求信号(HOLD)。
- DMA传送响应阶段：若CPU接收到总线请求信号(HOLD)，并且可以释放系统总线，则发出HLDA信号。
- DMA传送阶段：向地址总线发出地址信号，指出传送过程需使用的内存地址同时，向外设发出DMA应答信号(DACK)，实现该外设与内存之间进行DMA传送。在DMA传送期间，DMAC发出内存和外设的读/写信号。
- DMA传送结束阶段：当DMA传送的总字节达到初始化时的字节数或者达到某种终止DMA传输条件时，DMAC向CPU发出结束信号INT（撤消HOLD请求），将总线控制权交还CPU。

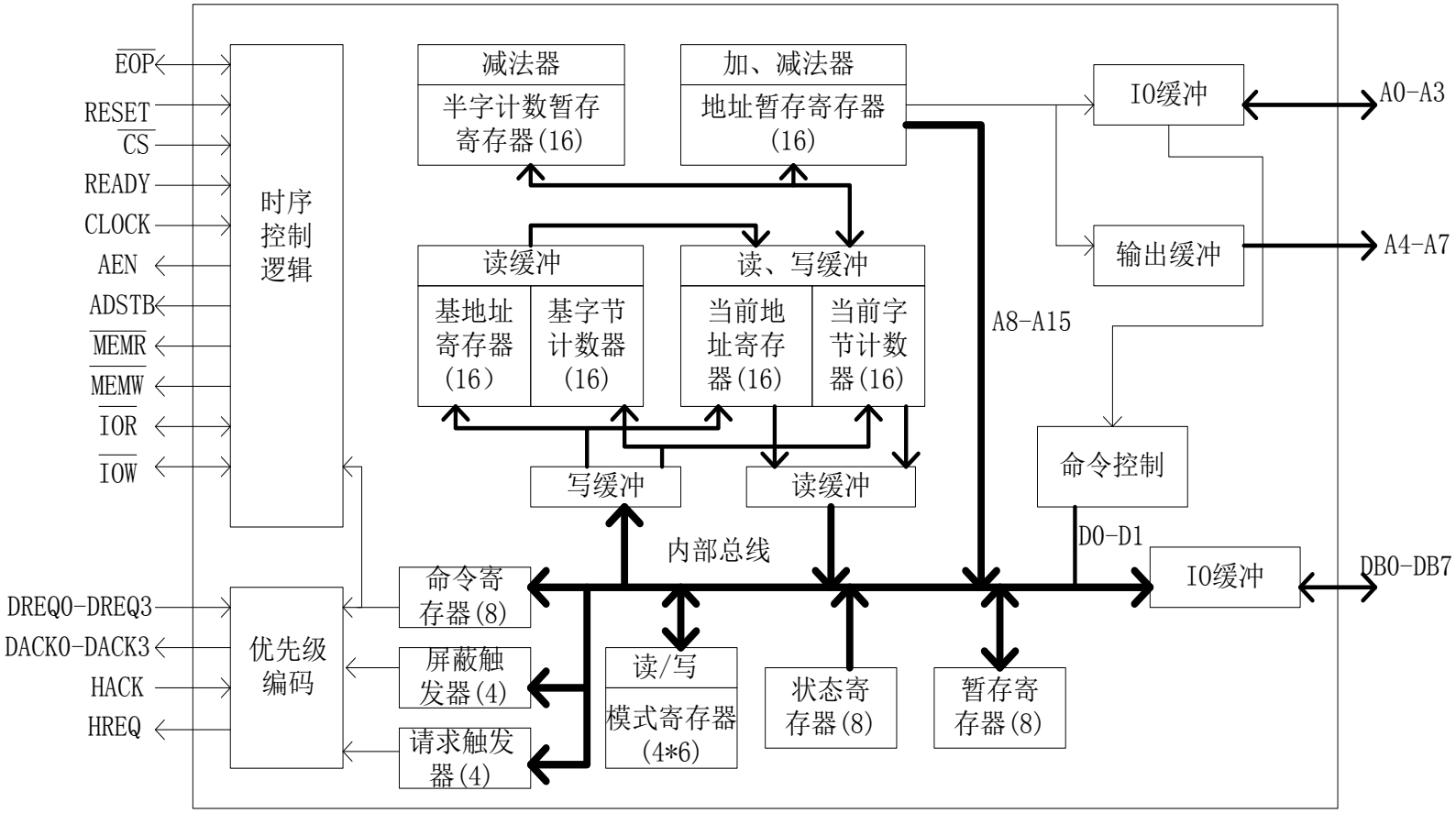
DMA以及中断在指令周期中的响应位置



8.3 DMA控制器


数据计数器
数据寄存器
地址寄存器
控制逻辑
数据总线
地址总线
DMA请求
DMA响应
中断请求
读控制
写控制
总线响应HLDA
总线请求HOLD

可编程DMA控制器8237A



DMAC 8237A内部寄存器

寄存器名称	偏移地址	读操作	写操作
通道0地址寄存器	0x00	当前地址	基地址
通道0字节计数器	0x01	当前字节	基字节
通道1地址寄存器	0x02	当前地址	基地址
通道1字节计数器	0x03	当前字节	基字节
通道2地址寄存器	0x04	当前地址	基地址
通道2字节计数器	0x05	当前字节	基字节
通道3地址寄存器	0x06	当前地址	基地址
通道3字节计数器	0x07	当前字节	基字节
控制状态寄存器	0x08	状态寄存器	控制寄存器
请求寄存器	0x09		请求寄存器
屏蔽寄存器	0x0A		屏蔽寄存器
模式寄存器	0x0B		模式寄存器
先/后触发器	0x0C		清先/后触发器
	0x0D	暂存寄存器	复位寄存器
	0x0E		清除屏蔽寄存器
	0x0F		多通道屏蔽寄存器

控制寄存器

D7：0， DACK低电平有效；1，DACK高电平有效

D6：0， DREQ高电平有效；1，DREQ低电平有效

D5：1，扩展写信号，/比正常时序提前一个周期；0，不扩展写信号

D4：0，固定优先级，通道0优先级最高，通道3优先级最低；1，循环优先级

D3：0，正常时序；1，压缩时序

D2：0，启动8237工作；1，停止8237工作

D1：内存到内存传输时，D1=1，源地址保持不变

D0：0，禁止内存与内存间的传输；1，允许内存与内存间的传输

状态寄存器

- D3~D0：分别对应通道3~0，指出4个通道的DMA传送是否结束，结束为1。
- D7~D4：分别对应通道3~0，表示4个通道是否有DMA请求，有DMA请求为1。

模式寄存器

- D7，D6：模式选择
  - 00，请求传输模式；01，单字节传输模式；10，块传输模式；11，级联传输模式
- D5：存储器地址增减选择，0，地址增1；1，地址减1
- D4：自动预置功能，0，禁止；1，允许
  - 在当前字节计数器到达0时，当前字节计数器和当前地址寄存器从基本字节计数器和基地址寄存器中自动获得新的初值。
- D3，D2：传输类型选择
  - 01，写传输（I/O到内存）；10，读传输（内存到I/O）；00，校验传输；11，无意义
- D1，D0：通道选择
  - 00，通道0；01，通道1；10，通道2；11，通道3

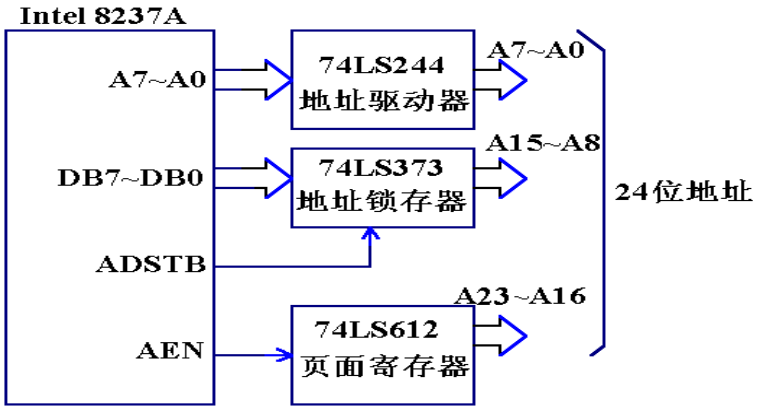
- **请求寄存器**
  - D2=1设置DMA请求；D1~D0：指定软件DMA请求的通道
- **屏蔽寄存器**
  - D2=1，设置屏蔽；D2 =0，清除屏蔽；D1~D0：选通道
- **清除屏蔽寄存器**
  - 写入0，一次清除四个通道的屏蔽触发器
- **多通道屏蔽寄存器**
  - D3~D0：分别对通道3~0设置屏蔽。1，设置屏蔽；0，清除屏蔽；

初始化编程

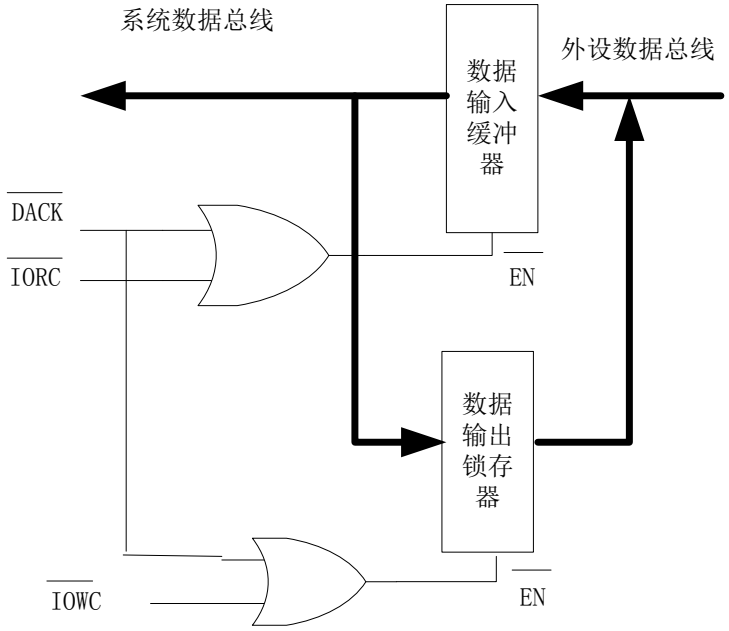
- 写入控制寄存器：设定各个通道DREQ/DACK信号的信号形式、DMA传输工作时序以及传输方向。
- 写入屏蔽寄存器——屏蔽要初始化的通道：禁止该通道的硬件DMA请求和软件DMA请求。
- 写入模式寄存器：设定指定DMA通道的传输模式。
- 先后触发器置0：为写入基地址和基本字节寄存器做准备。
- 写入基地址和基本字节寄存器：分别两次写入基地址寄存器和基本字节寄存器，初始化16位数据
- 解除屏蔽：允许指定通道的硬件DMA请求和软件DMA请求。
- 如果是软件请求DMA传送，则写入请求寄存器；如果是由硬件请求DMA传送，则不需要写入请求寄存器，初始化结束。

内存和I/O寻址

- 8237实现24位地址寻址存储器电路原理



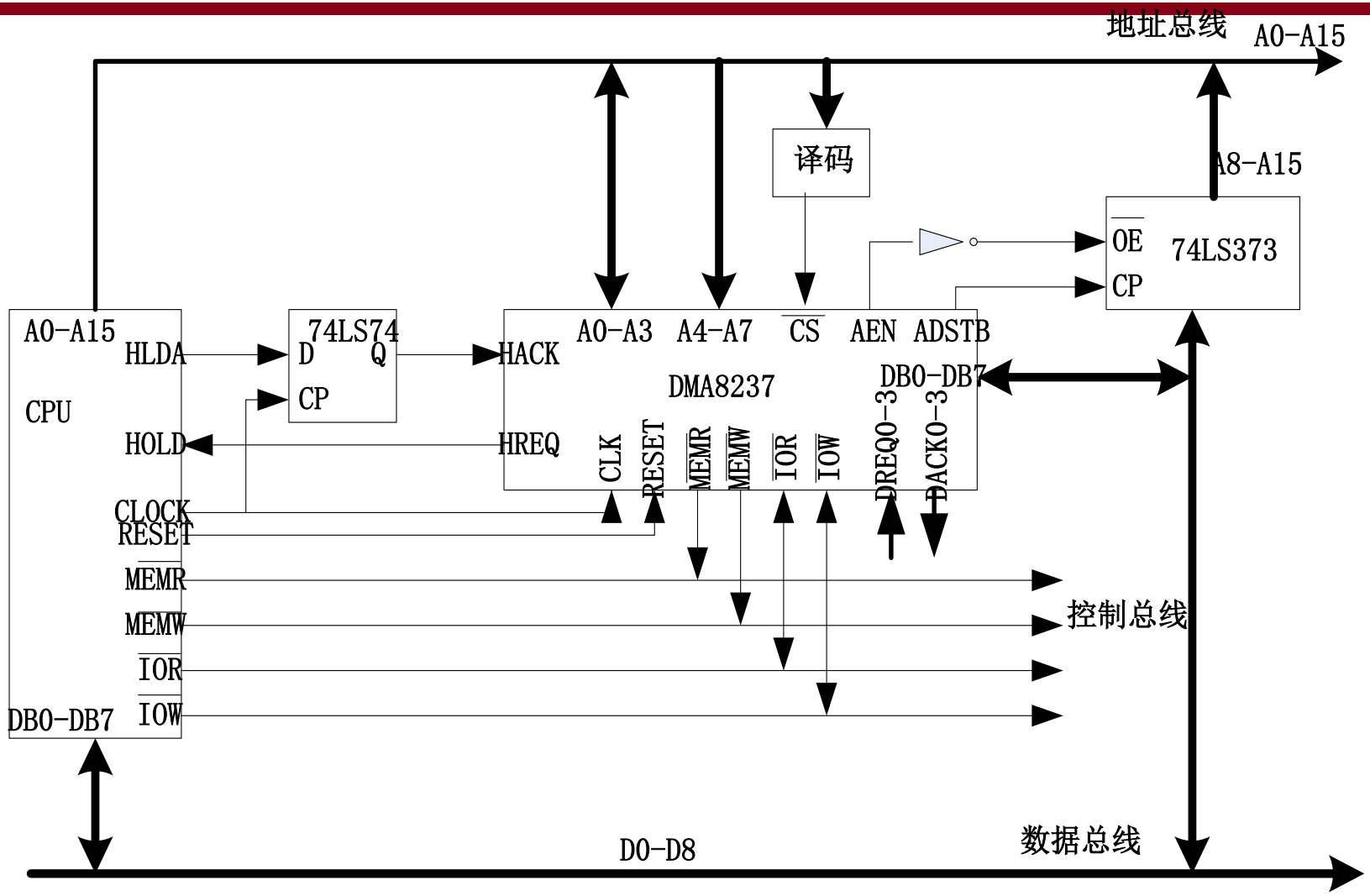
- DMA IO寻址原理



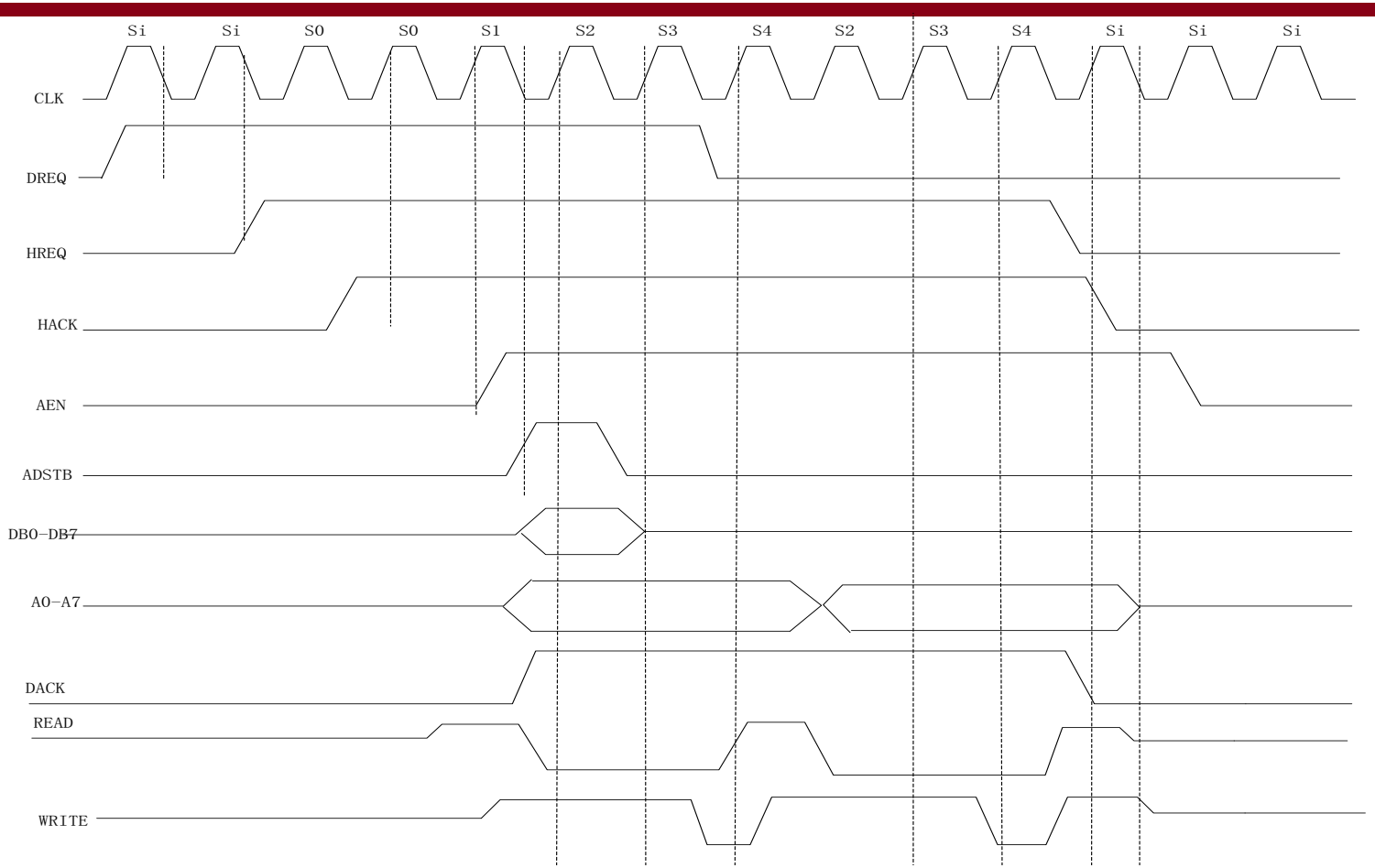
存储器与存储器间的数据传送

- 存储器与外设数据传送方式可以在任一个DMA 通道与存储器之间进行，
- 而存储器与存储器数据传送方式只能在通道0和通道1之间进行，而且只能是通道0寻址源存储区，通道1寻址目的存储区。
- 先将通道0当前地址寄存器寻址的源字节数据读入DMA内的数据暂存器内，然后再转送到通道1当前地址寄存器寻址的目的数据区去

典型应用电路

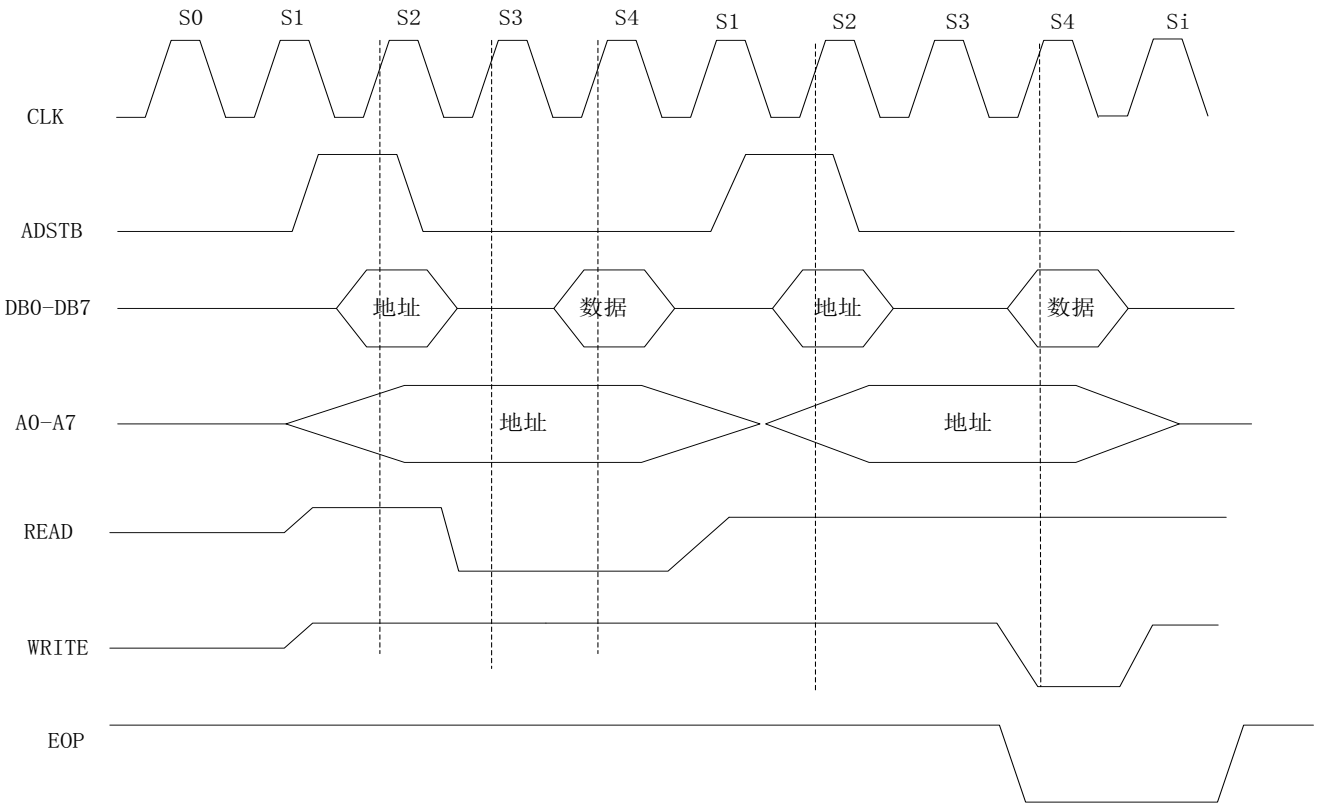

地址总线
A0-A15
译码
A8-A15
OE
74LS373
CP
A0-A15
HLDA
74LS74
D
Q
CP
HACK
A0-A3
A4-A7
CS
AEN
ADSTB
DMA8237
DB0-DB7
CPU
HOLD
HREQ
CLK
RESET
MEMR
MEMW
IOR
IOW
DREQ0-3
DACK0-3
CLOCK
RESET
MEMR
MEMW
IOR
IOW
控制总线
DB0-DB7
D0-D8
数据总线

存储器与IO接口DMA字符传输模式时序



半字类型数据的传输

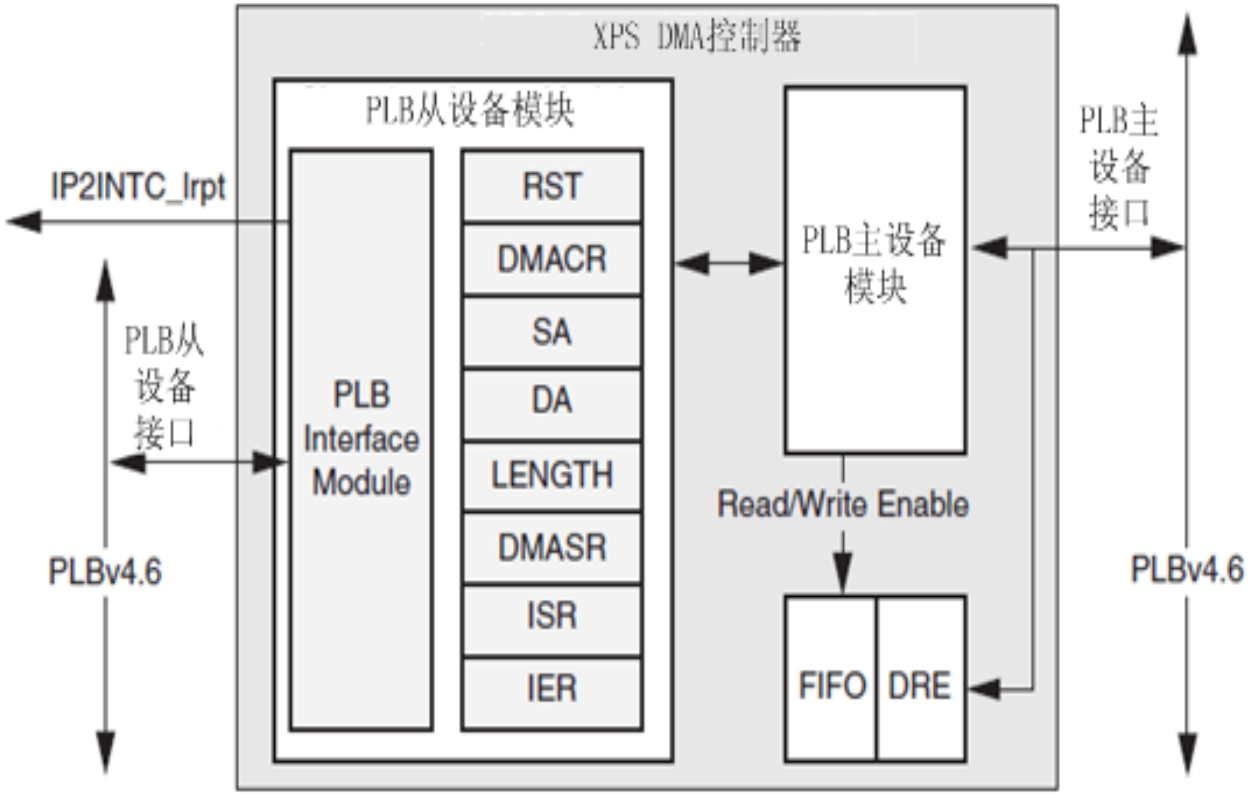
存储器与存储器间DMA字符传输模式时序



- 微型计算机系统使用3片8237，第一片8237，基地址为0x0000，使用I/O地址0x0000~0x000F。第二片8237，基地址为0x0080，使用I/O地址0x0080~0x008F。第三片8237，基地址为0x00C0H，使用I/O地址0x00C0H~0x00CF。若采用第一片8237通道1，将内存起始地址为0x80000的0x300个字节直接输出给外设接口。试撰写该DMA控制器初始化程序段。

```
PORTWrite(0x08,0);//对控制寄存器进行初始化
PORTWrite(0x0A,5);//屏蔽通道1
PORTWrite(0x0C,0);//清除先/后寄存器
PORTWrite(0x02,0);
PORTWrite(0x02,0);//写基地址寄存器
PORTWrite(0x83,8);//写页面寄存器（地址为0x83），保存高位地址
PORTWrite(0x03,0x02);
PORTWrite(0x03,0xFF);//写基本字节寄存器
PORTWrite(0x0B,0x49);//设置模式寄存器,单字节传输，
PORTWrite(0x0A,1);//解除通道1屏蔽
```

Xilinx XPS DMA控制器简介



PLB主设备以不同的地址来区分IO接口和存储器的。DMA传输时在嵌入式系统内都可以认为是存储器到存储器的传输方式，存储器与IO接口之间DMA传输不同于存储器之间的DMA传输主要表现在IO接口不需要修改端口地址，而存储器需要修改地址。

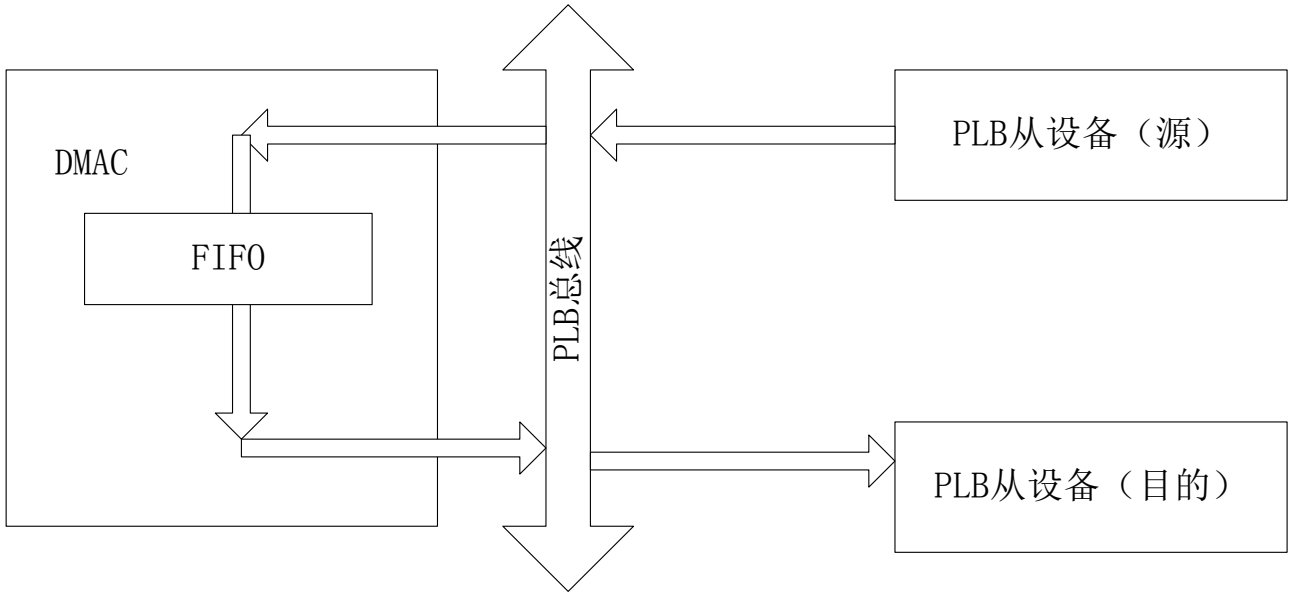
Xilinx XPS DMA控制器内部寄存器

寄存器名称	偏移地址	功能描述
RST	0X0	向该寄存器写0x0000000A，将软件复位DMA控制器
DMACR	0X4	Bit0为SINC:1,源地址增加；0,源地址不变；bit1为DINC：1,目的地址增加；0,目的地址不变
SA	0X8	保存源地址
DA	0XC	保存目的地址
LENGH	0X10	传送数据长度，一旦往该寄存器写入字节长度，DMA控制器就开始进行DMA传输
DMASR	0X14	Bit0，DMA传输忙闲状态：0没有DMA传输，1正在进行DMA传输；bit1，DMA总线出错状态：1有错，0没有错误
ISR	0X2C	Bit30，DMA出错中断；bit31，DMA传输结束中断，1表示产生了中断，向相应的位写1将清除该中断状态
IER	0X30	Bit30，DMA出错中断使能；bit31，DMA传输结束中断使能，写1允许中断，写0不允许产生中断

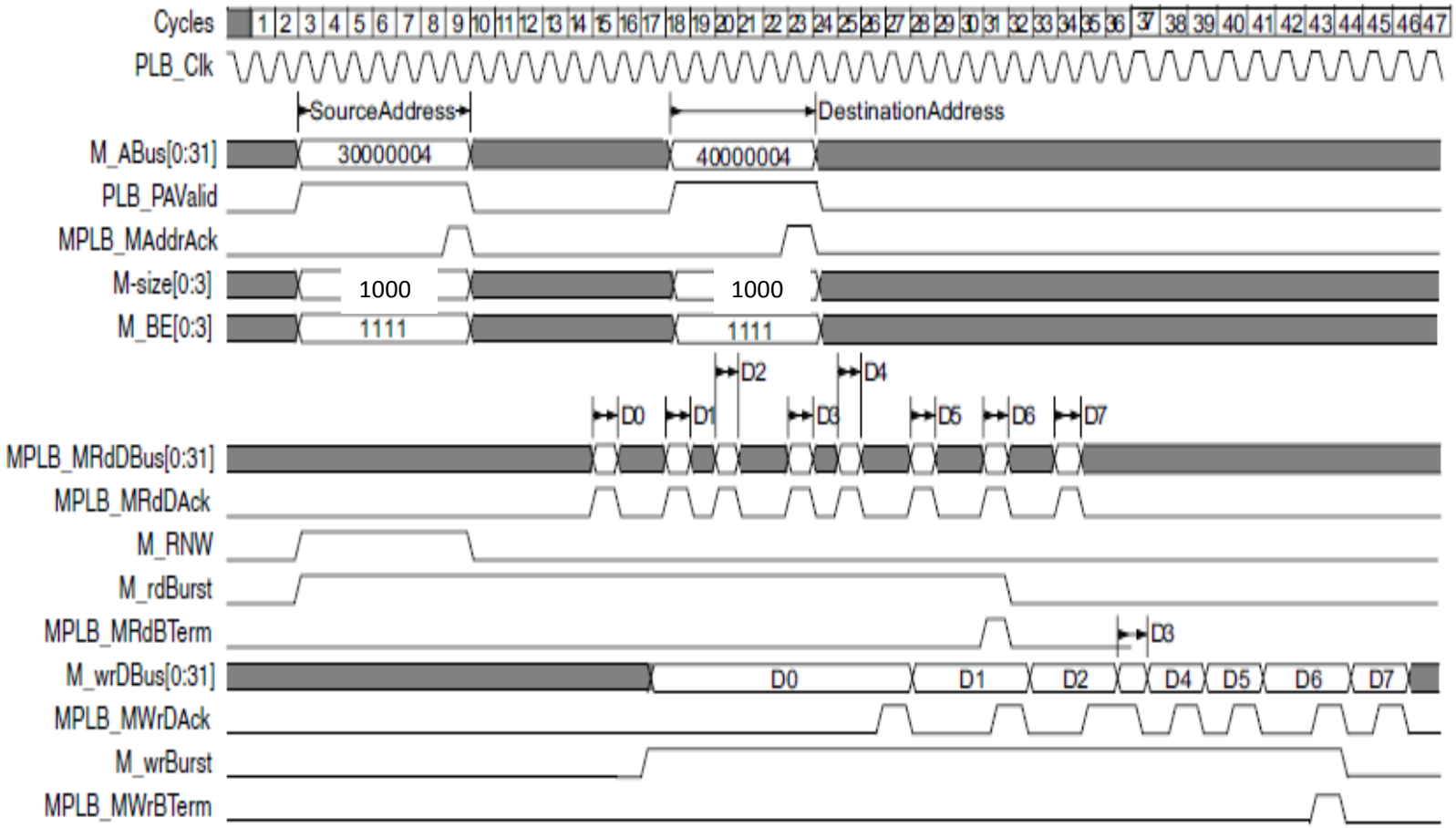
控制Xilinx XPS DMA控制器进行DMA数据传输的流程

- 写源地址寄存器SA，配置DMA传输源端起始地址；
- 写目的地址寄存器DA，配置DMA传输目的端起始地址；
- 写控制寄存器DMACR，通过写SINC和DINC控制源和目的端地址是否发生改变。
  - 若是存储器到IO接口的传输，则SINC=1，DINC=0；
  - 若是IO接口到存储器的传输，则SINC=0，DINC=1；
  - 若是存储器到存储器的传输，则SINC=1，DINC=1；
- 写传输字节长度寄存器LENGTH，将要传输的字节数写入该寄存器就启动了DMA传输。

Xilinx XPS DMA控制器DMA数据传输通路

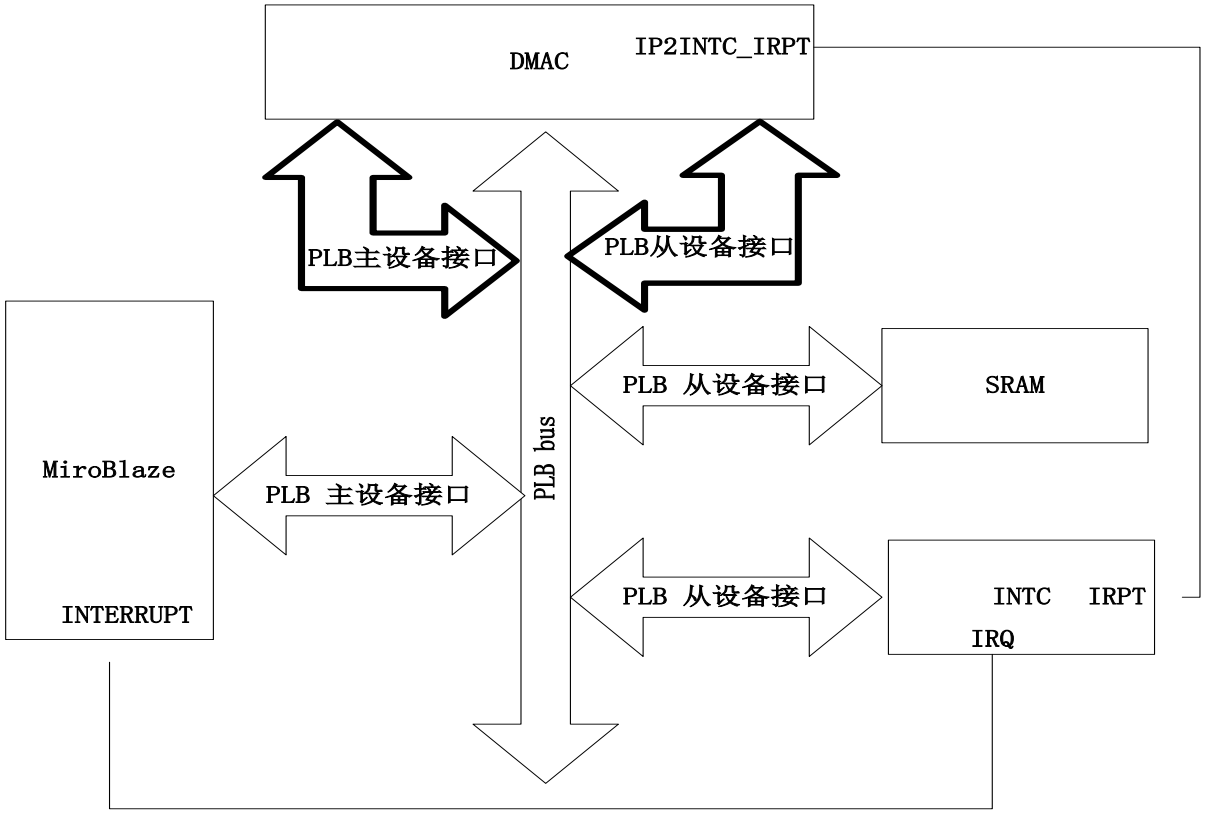


Xilinx XPS DMA控制器突发模式DMA数据传输时序



- 已知某基于PLB总线的计算机系统采用MicroBlaze微处理器，并带有SRAM存储器，存储器地址范围为0x83000000~0x83ffffff，要求采用DMA方式将起始地址为0x83200000的1K字节的内存数据传输到地址为0x83100000的内存存储区间，传输结束时产生中断信号，当CPU接收到该中断信号后，程序退出。试设计接口电路并编写控制程序(检测数据传输的正确性)。

基于PLB总线支持DMA传输的计算机系统硬件电路连接框图



DMA API

Int XDmaCentral_CfgInitialize(XDmaCentral *InstancePtr,XDmaCentral_Config *DmaCentralCfgPtr,u32 EffectiveAddr);
//初始化配置参数
Void XDmaCentral_Reset(XDmaCentral *InstancePtr);
//复位DMA控制器
Void XDmaCentral_SetControl(XDmaCentral *InstancePtr, u32 Value);
//设置DMA控制寄存器为Value
u32 XDmaCentral_GetStatus(XDmaCentral *InstancePtr);
//读取DMA状态寄存器
voidXDmaCentral_Transfer(XDmaCentral *InstancePtr,void *SourcePtr, void *DestinationPtr, u32 ByteCount);
//设置源地址寄存器为SourcePtr，目的地址寄存器为DestinationPtr，长度寄存器为ByteCount，并启动DMA传输
XDmaCentral_Config *XDmaCentral_LookupConfig(u16 DeviceId);
//操作系统系统的统一设备ID查找DMA配置参数数据结构
Int XDmaCentral_Initialize(XDmaCentral *InstancePtr, u16 DeviceId);
//根据操作系统系统的统一设备ID，初始化DMA控制器实例
void XDmaCentral_InterruptEnableSet(XDmaCentral *InstancePtr, u32 Mask);
//使能Mask对应的DMA中断源
u32 XDmaCentral_InterruptStatusGet(XDmaCentral *InstancePtr);
//获取DMA中断请求源
void XDmaCentral_InterruptClear(XDmaCentral *InstancePtr, u32 Mask);
//清除Mask对应的DMA中断源的中断请求状态

DMA中断处理函数

```
Void XDmaHandler(void * CallBackRef){
        Xuint32 status;
//读取中断状态寄存器
status=XDmaCentral_ReadReg(XPAR_XPS_CENTRAL_DMA_0_BASEADDR,
XDMC_ISR_OFFSET);
//判断是否是DMA传输结束中断
if((status&XDMC_IXR_DMA_DONE_MASK)==XDMC_IXR_DMA_DONE_MASK )
        {dma_done=1;
//清除中断标志
        XDmaCentral_WriteReg(XPAR_XPS_CENTRAL_DMA_0_BASEADDR,

XDMC_ISR_OFFSET,XDMC_IXR_DMA_DONE_MASK);
        }
}
```

注册DMA中断处理函数到中断系统中

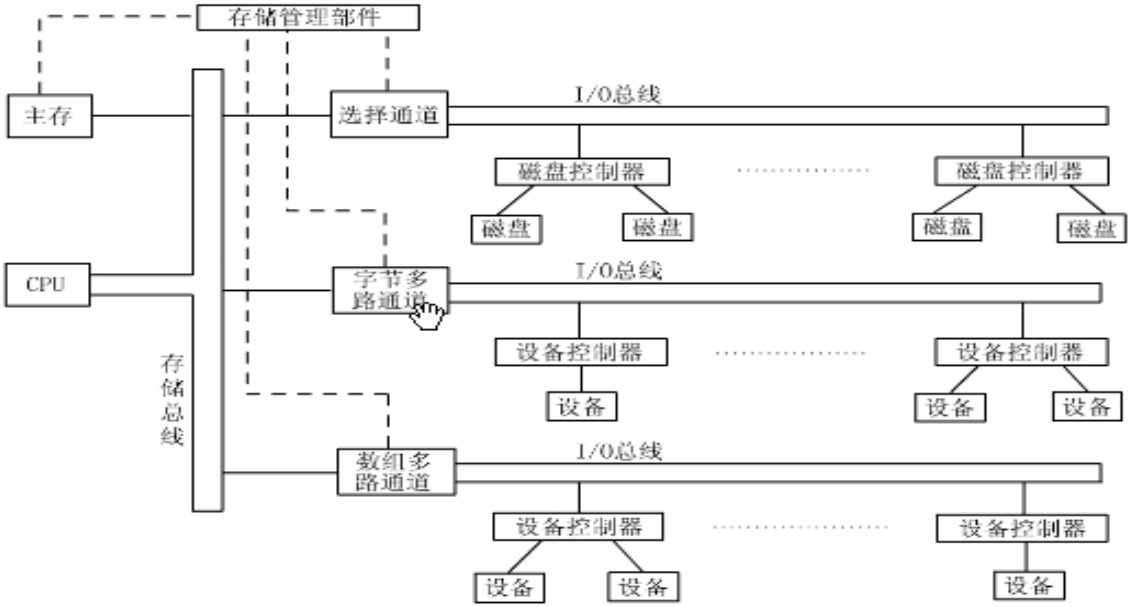
```
intXDmaCentral_SetupIntrSystem(XIntc*IntcInstancePtr, XDmaCentral
*DmaCentralInstancePtr, u16 IntrId)
{
        int Status;
        Status = XIntc_Initialize(IntcInstancePtr, INTC_DEVICE_ID);
        Status = XIntc_Connect(IntcInstancePtr, IntrId,
                                (XInterruptHandler) XDmaHandler,
                                (void *)DmaCentralInstancePtr);
        Status = XIntc_Start(IntcInstancePtr, XIN_REAL_MODE);
        XIntc_Enable(IntcInstancePtr, IntrId);
microblaze_register_handler((XInterruptHandler)XIntc_InterruptHandler, IntcInstancePtr);
        microblaze_enable_interrupts();
        return XST_SUCCESS;
}
```

DMA控制主函数

```
int main()
{
/查找DMA驱动配置项
        ConfigPtr= XDmaCentral_LookupConfig(XPAR_XPS_CENTRAL_DMA_0_DEVICE_ID);
        //初始化驱动参数
Status = XDmaCentral_CfgInitialize(&DmaCentral, ConfigPtr,ConfigPtr->BaseAddress);
//复位DMA控制器
                XDmaCentral_Reset(&DmaCentral);
                SrcPtr = 0x83200000;//初始化源地址指针
                DestPtr = 0x83100000;//初始化目的指针
//初始化源内存区域数据为0~1024模256的余数，目的内存区域数据为0
                for (Index = 0; Index < BUFFER_BYTESIZE; Index++) {
                                SrcPtr[Index] = Index;
                                DestPtr[Index] = 0;
                }
//设置DMA控制寄存器使得SINC=DINC=1
XDmaCentral_SetControl(&DmaCentral, XDMC_DMACR_SOURCE_INCR_MASK |XDMC_DMACR_DEST_INCR_MASK);
//使能DMA控制器当传输完毕时产生中断
XDmaCentral_InterruptEnableSet(&DmaCentral,XDMC_IXR_DMA_DONE_MASK);
//注册DMA中断服务程序
XDmaCentral_SetupIntrSystem(&IntcInstance, &DmaCentral, XPAR_XPS_INTC_0_XPS_CENTRAL_DMA_0_IP2INTC_IRPT_INTR);
//启动DMA传输
XDmaCentral_Transfer(&DmaCentral, SrcPtr,       DestPtr, BUFFER_BYTESIZE);
while (dma_done==0);//等待DMA传输结束中断
                //检验两块数据区的数据是否相等
for (Index = 0; Index < BUFFER_BYTESIZE; Index++) {
                if ( DestPtr[Index] != SrcPtr[Index]) {
                                return XST_FAILURE;
                                }
                }
        return XST_SUCCESS;
}
```

8.4通道

- 通道是一个具有输入输出控制处理器的输入输出部件。通道控制器有自己的指令，即通道命令
- 能够根据程序控制多个外部设备并提供了DMA共享的功能，而DMA只能进行固定数据传输操作



通道类型

- 选择通道
  - 一次对一个设备进行操作。这种通道称为选择通道，它与设备之间的传输一直维持到设备请求的传输完成为止，然后为其他外围设备传输数据。
- 数组多路通道
  - 数组多路通道以数组(数据块)为单位在若干高速传输操作之间进行交叉复用。设备的数据传输以块为单位。通道用块交叉的方法，轮流为多个外设服务。当同时为多台外设传送数据时，每传送完一块数据后选择下一个外设进行数据传送，使多路传输并行进行。
- 字节多路通道
  - 通道以字节交叉方式轮流为多个外设服务，以提高通道的利用率。它的操作模式有两种：字节交叉模式和突发模式。

通道的性能指标

- 衡量通道性能的指标是通道的流量，它指通道在传送数据时，1秒钟内传送的数据位数(bps)。
- 通道所能达到的最大流量称为通道的极限流量。
- 字节多路通道，通道的极限流量应大于所接外设的字节传送速率之和，
- 其他两种方式的通道，通道的极限流量应大于所接外设中字节传送速率最大的设备，因为数组多路通道和选择通道是轮流为外设传输数据的。

通道的功能

（1）接受CPU的输入输出操作指令，按指令要求控制外围设备。

（2）从内存中读取通道程序，并执行，即向设备控制器发送各种命令。

（3）组织和控制数据在内存与外设之间的传送操作。根据需要提供数据中间缓存空间以及提供数据存入内存的地址和传送的数据量。

（4）读取外设的状态信息，形成整个通道的状态信息，提供给CPU或保存在内存中。

（5）向CPU发出输入输出操作中断请求，将外围设备的中断请求和通道本身的中断请求按次序报告CPU。CPU通过执行输入输出指令以及处理来自通道的中断，实现对通道的管理。来自通道的中断有两种：一种是数据传输结束中断；另一种是故障中断。

●通道指令也叫通道控制字(CCW)，它是通道用于放行输入输出操作的指令，由CPU存放在内存中，由通道处理器从内存中取出并执行。通道执行通道指令以完成输入输出传输。通道程序由一条或几条通道指令组成，也称通道指令链。

设备控制器具体任务

（1）从通道接受通道指令，控制外围设备完成指定的操作；

（2）向通道提供外围设备的状态；

（3）将各种外围设备的不同信号转换成通道能够识别的标准信号。

通道工作过程

- 通道中包括通道控制器、状态寄存器、中断机构、通道地址寄存器、通道指令寄存器等。
  - 通道地址寄存器相当于一般CPU中的程序计数器。
  - 通道状态字类似于CPU内部的程序状态字，用于记录输入输出操作结束的原因，以及输入输出操作结束时通道和设备的状态。

1. CPU在进行一个输入输出操作之前，首先准备好通道程序，然后安排好数据缓冲区，再给通道和设备发启动命令。CPU准备好的通道程序存放在内存中，由通道控制器读取并执行。
2. 通道接到启动信号后，首先到指定的内存单元中取通道地址字，放在通道地址寄存器中。
3. 然后根据通道地址寄存器中的值到内存中去取第一条通道指令，并放在通道指令寄存器中。
4. 通道程序执行时，通过对通道指令寄存器中的相应位进行设置，来告诉通道指令执行机构在执行完成当前指令后，自动转入下一条指令或者结束数据传送过程。
5. 通道程序的最后一条指令是一条结束指令，通道在执行到这条结束指令时就不再取下一条指令，而是通知设备结束操作。
6. 在通道程序执行完毕后，由通道向CPU发中断信号，并将通道状态字写入内存专用单元，
7. CPU根据通道状态字CSW（channel status word）分析这次输入输出操作的执行情况。

作业

- 6,8