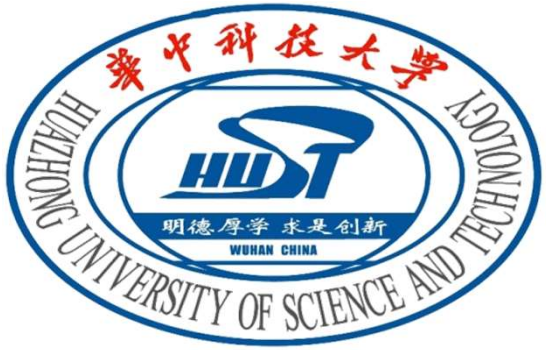
微机原理与接口技术

DMA传输原理及示例

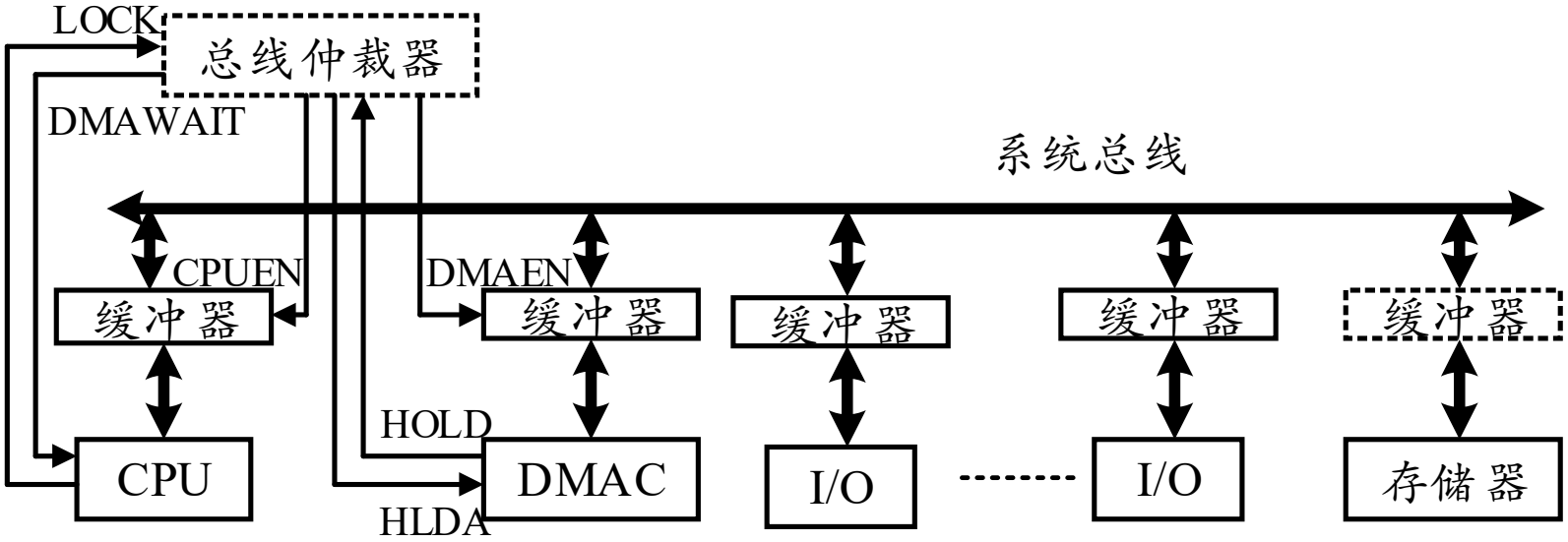
华中科技大学　左冬红



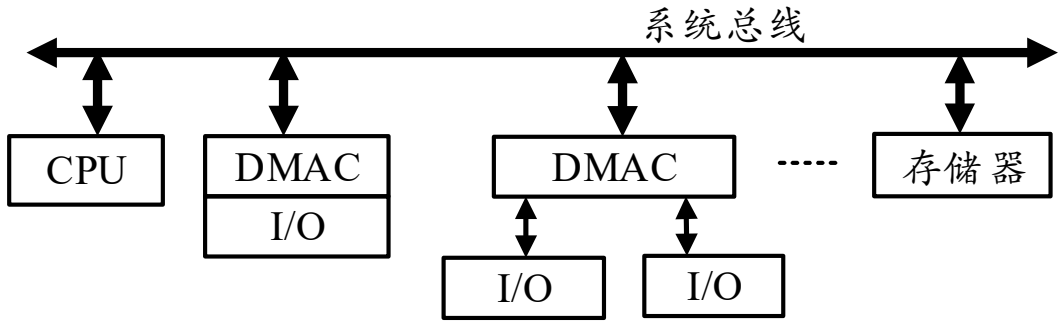
DMA传输

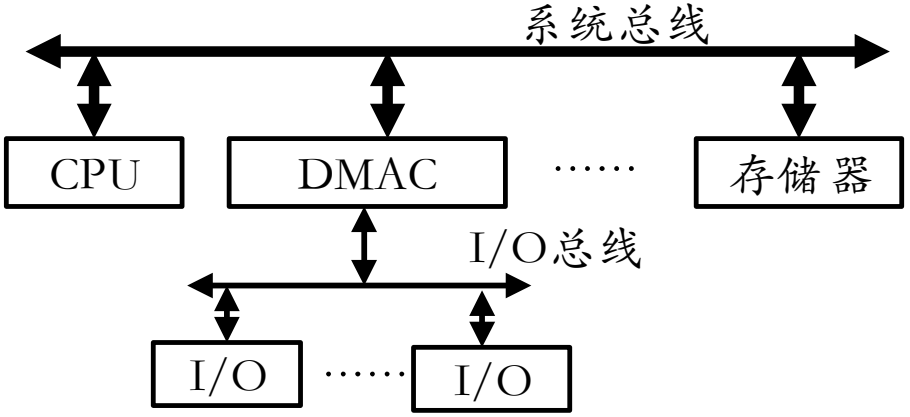
DMA传输是指不在微处理器控制下实现的数据传输

DMA传输系统构成-单总线



DMA传输系统构成





DMA数据传输方向

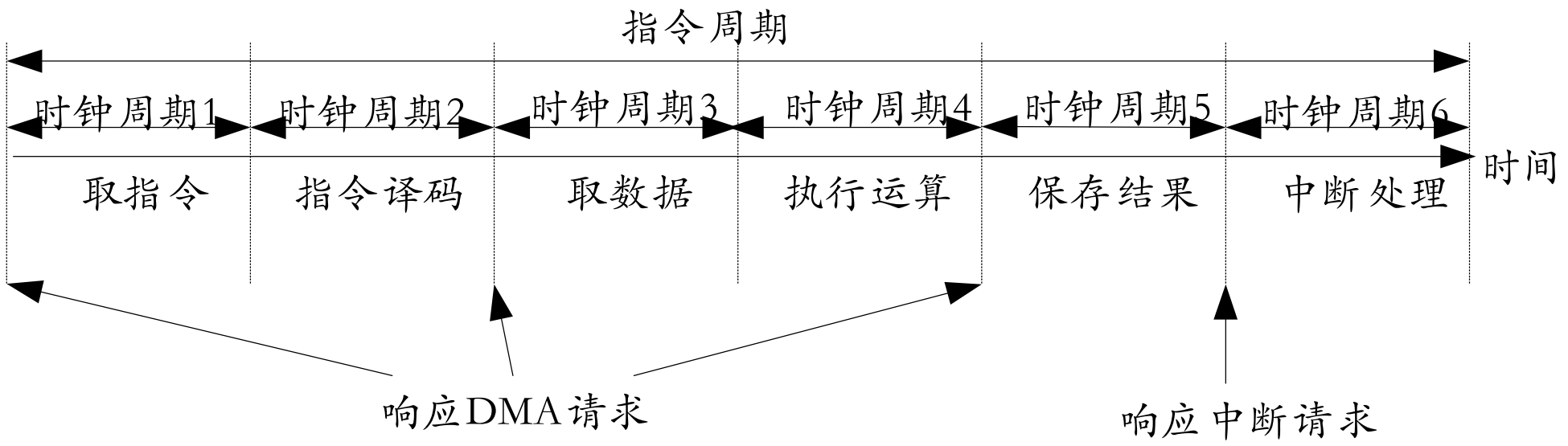
DMA传输方向分为三种：

- I/O设备到存储器
- 存储器到I/O设备
- 存储器到存储器

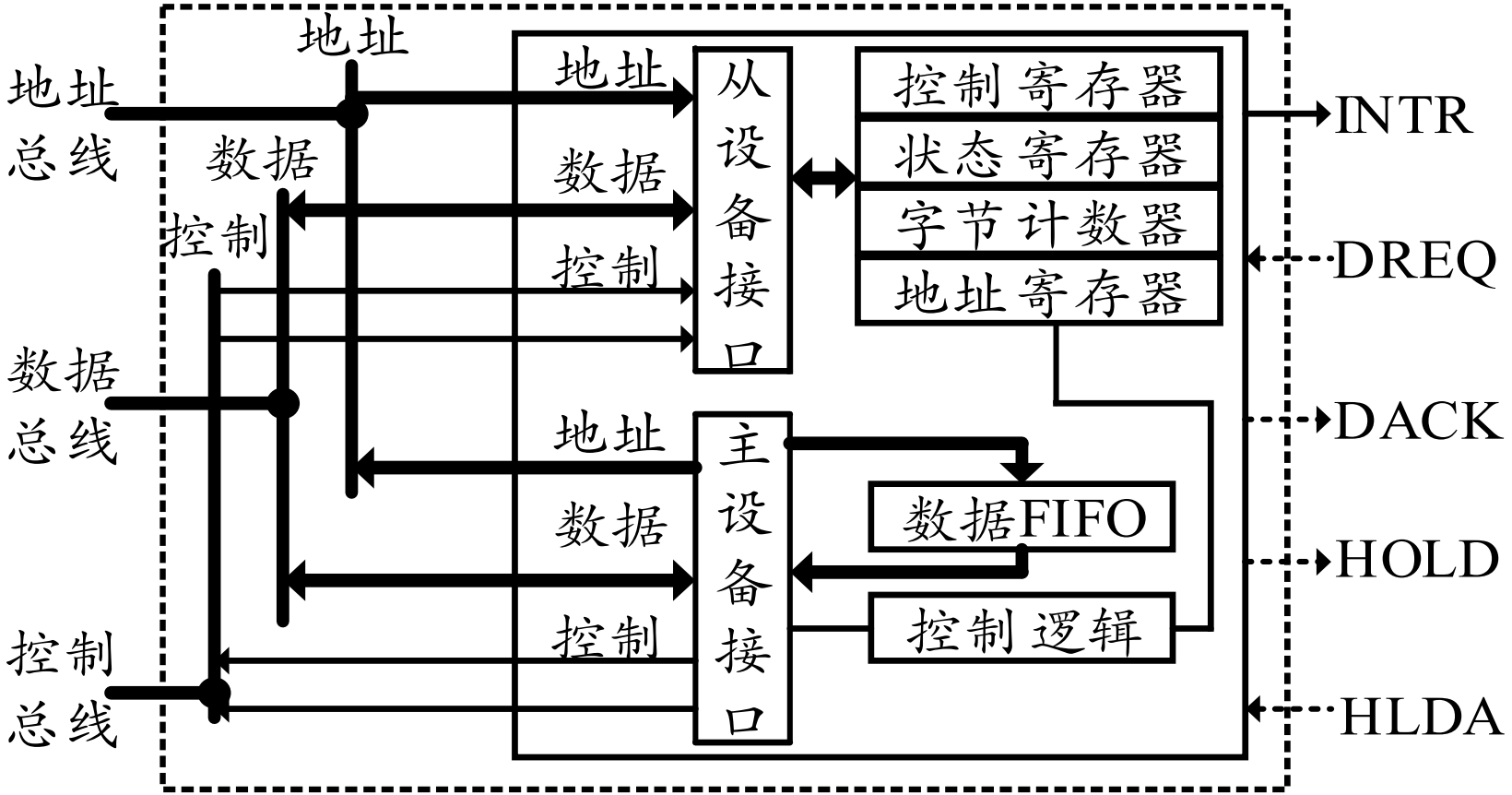
DMA传输流程

- 初始化阶段：
  - DMAC初始化，如配置数据传输模式、数据传输方向、存储器存储区域首地址、存储器地址是递增还是递减、传送字节数等。
- DMA请求阶段：
  - 外设产生DREQ信号或者CPU向DMAC写入启动DMA传输控制字，DMAC向CPU发出总线请求信号(HOLD)。
- DMA响应阶段：
  - 若CPU接收到总线请求信号(HOLD)，并且可以释放总线，则发出HLDA信号。此时，CPU便放弃对总线的控制，DMAC获得总线控制权。
- DMA传输阶段：
  - DMAC获得总线控制权后，向地址总线发出地址信号，指出传输过程需使用的存储单元地址。同时，向外设发出DMA应答信号(DACK)，实现IO设备与存储器之间的DMA传输。DMA传输期间，DMAC发出存储器和IO设备读/写信号。DMA传输过程中，每传送一个字节，字节计数器减1，减为0时，该次DMA传输结束。
- DMA传输结束阶段：
  - 当DMA传输的总字节达到初始化时的字节数或者达到某种终止DMA传输的条件时，DMAC向CPU发出结束信号INT，同时撤消HOLD请求，将总线控制权交还CPU。

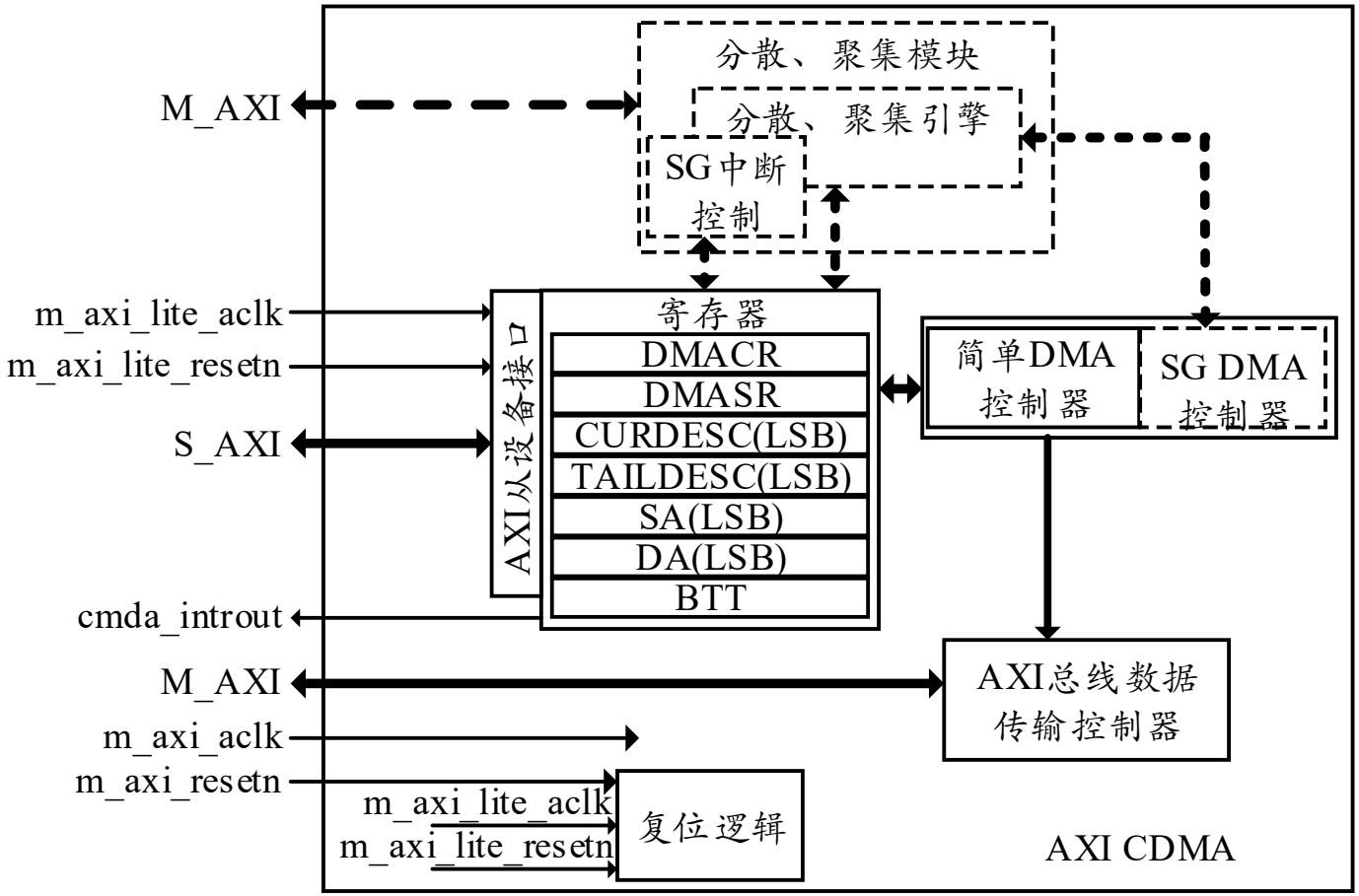
DMA响应时间


指令周期
时钟周期1
时钟周期2
时钟周期3
时钟周期4
时钟周期5
时钟周期6
时间
取指令
指令译码
取数据
执行运算
保存结果
中断处理
响应DMA请求
响应中断请求

DMA控制器结构


地址
地址
总线
数据
控制
数据
总线
控制
总线
地址
数据
控制
从设备接口
控制寄存器
状态寄存器
字节计数器
地址寄存器
INTR
DREQ
DACK
HOLD
HLDA
地址
数据
控制
主设备接口
数据FIFO
控制逻辑

AXI CDMA结构


分散、聚集模块
分散、聚集引擎
SG中断控制
M_AXI
m_axi_lite_aclk
m_axi_lite_resetn
S_AXI
AXI从设备接口
寄存器
DMACR
DMASR
CURDESC(LSB)
TAILDESC(LSB)
SA(LSB)
DA(LSB)
BTT
简单DMA控制器
SG DMA控制器
cmda_introut
M_AXI
AXI总线数据传输控制器
m_axi_aclk
m_axi_resetn
m_axi_lite_aclk
m_axi_lite_resetn
复位逻辑
AXI CDMA

CDMA寄存器存储映像

寄存器名称	偏移地址	含义
DMACR	0x0	控制寄存器
DMASR	0x4	状态寄存器
CURDESC(LSB)	0x8	链表当前指针（低32位）
CURDESC(MSB)	0xc	链表当前指针（高32位）
TAILDESC(LSB)	0x10	链表尾指针（低32位）
TAILDESC(LSB)	0x14	链表尾指针（高32位）
SA(LSB)	0x18	源地址（低32位）
SA(MSB)	0x1c	源地址（高32位）
DA(LSB)	0x20	目标地址（低32位）
DA(MSB)	0x24	目标地址（高32位）
BTT	0x28	字节计数器

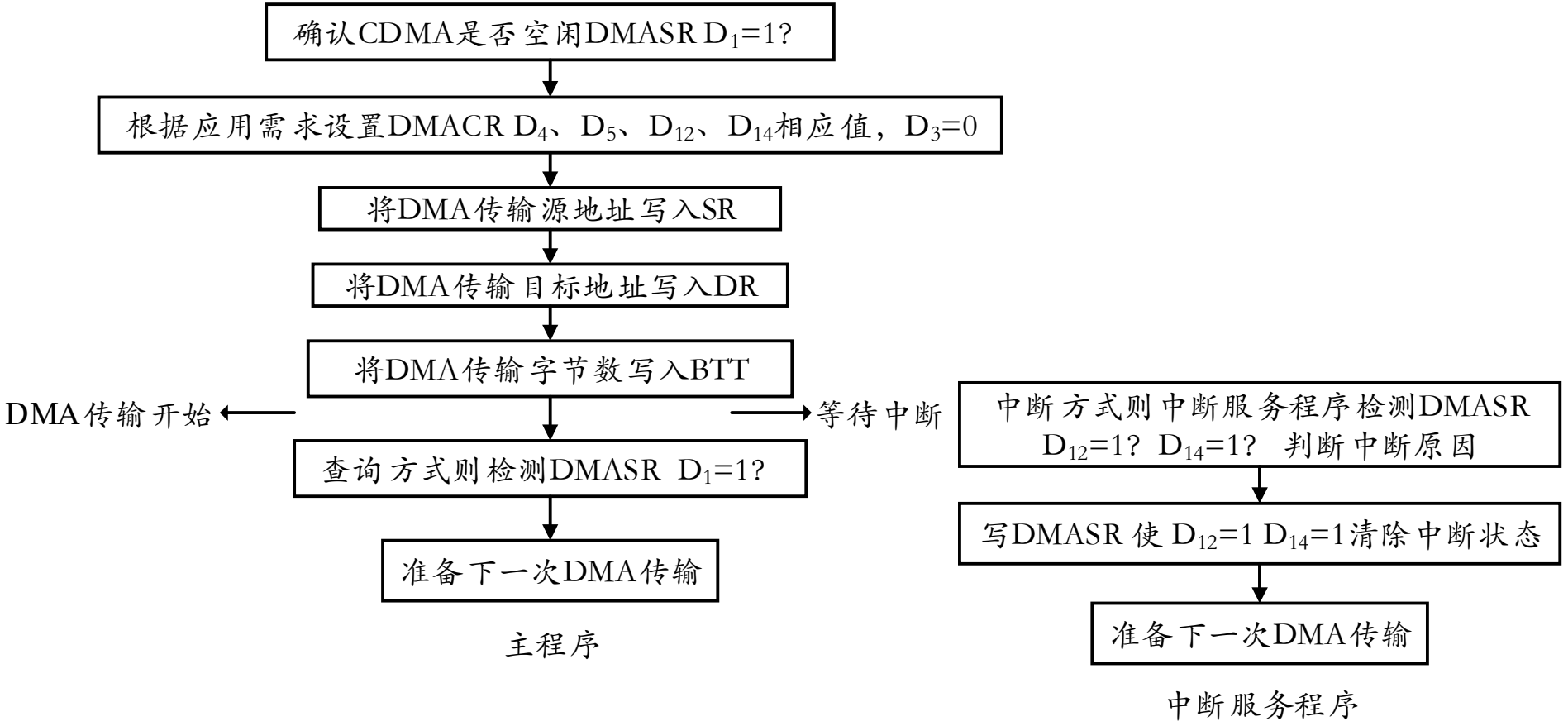
DMACR寄存器含义

位	名称	CDMA工作模式	含义
31~24	IRQ延时时间	分散、聚集	设定产生传输结束中断的延时时间，若所有DMA传输结束，但未达到DMA传输结束次数中断阈值。分散、聚集模块内部计时器开始从IRQ延时时间减计数，当计数到0时，产生中断。若该值为0，则表示不延时直接产生中断。
23~16	IRQ次数阈值	分散、聚集	设定产生传输结束中断请求的DMA传输完成次数，每一个DMA传输完成，该值减1，到达0时产生传输结束中断请求，要求最小设定值为0x1。
15	保留	N/A	写无意义，读为0
14	ERR_IrqEn	所有模式	使能出错中断，0-禁止；1-使能
13	Dly_IrqEn	分散、聚集	使能传输延时中断，0-禁止；1-使能
12	IOC_ IrqEn	所有模式	使能传输结束中断，0-禁止；1-使能
11~7	保留	N/A	写无意义，读为0
6	使能循环链表	分散、聚集	1-采用循环链表，忽略链表描述符中的结束位标志； 0-链表描述符结束位有效。
5	孔洞写	所有模式	1-目标地址固定，适用于存储器到IO设备DMA传输；0-目标地址递增
4	孔洞读	所有模式	1-源地址固定，适用于IO设备到存储器DMA传输；0-源地址递增
3	分散、聚集模式	所有模式	1-分散、聚集DMA模式；0-简单DMA模式
2	复位	所有模式	1-软件复位CDMA; 0-正常工作
1	使能尾指针	分散、聚集	该位只读，1-包含分散、聚集引擎；0-不包含
0	保留	N/A	写无意义，读为0

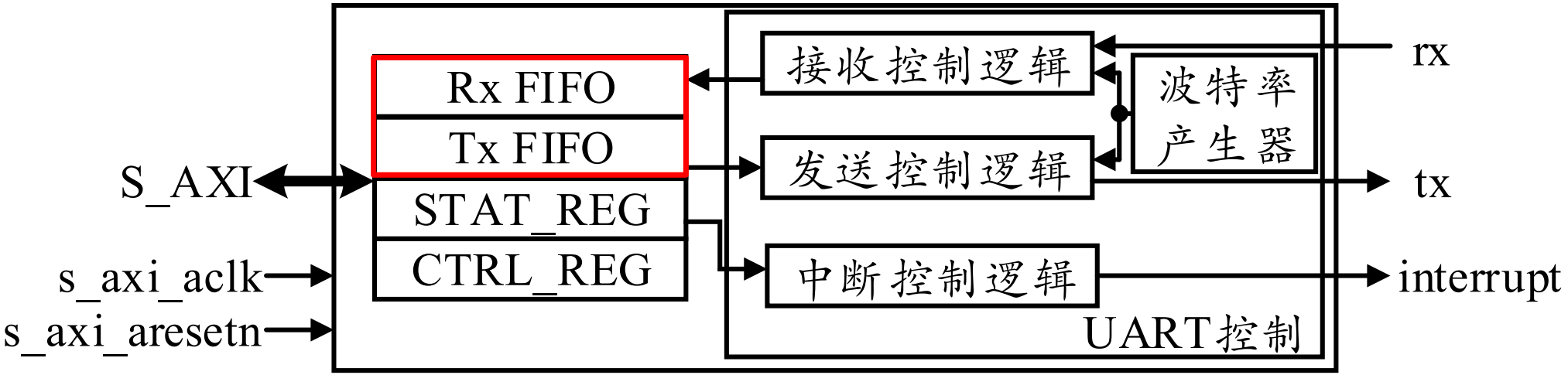
DMASR寄存器含义

位	名称	CDMA工作模式	含义
31~24	IRQ Delay	分散、聚集	DMA传输结束中断延时计数器当前计数值
23~16	IRQThreshold	分散、聚集	DMA传输结束中断次数计数器当前计数值
15	保留	N/A	写无意义，读为0
14	ERR_Irq	所有模式	出错中断状态，0-无，1-有；写1清除中断状态
13	Dly_Irq	分散、聚集	传输延时中断状态，0-无，1-有；写1清除中断状态
12	IOC_IrqEn	所有模式	传输结束中断，0-无，1-有；写1清除中断状态
11	保留	N/A	写无意义，读为0
10	SGDecErr	分散、聚集	分散/聚集译码错误，0-无，1-有；写1清除中断状态
9	SGSlvErr	分散、聚集	分散/聚集从设备错误，0-无，1-有；写1清除中断状态
8	SGIntErr	分散、聚集	分散/聚集内部错误，0-无，1-有；写1清除中断状态
7	保留	N/A	写无意义，读为0
6	DMADecErr	所有模式	DMA译码错误，0-无，1-有；写1清除中断状态
5	DMASlvErr	所有模式	DMA从设备错误，0-无，1-有；写1清除中断状态
4	DMAIntErr	所有模式	DMA内部错误，0-无，1-有；写1清除中断状态
3	SGIncld	所有模式	1-包含分散、聚集模块；0-不包含
2	保留	N/A	写无意义，读为0
1	IDLE	所有模式	1-空闲；0-忙
0	保留	N/A	写无意义，读为0

简单DMA传输控制流程



CDMA应用示例-UART DMA输入输出

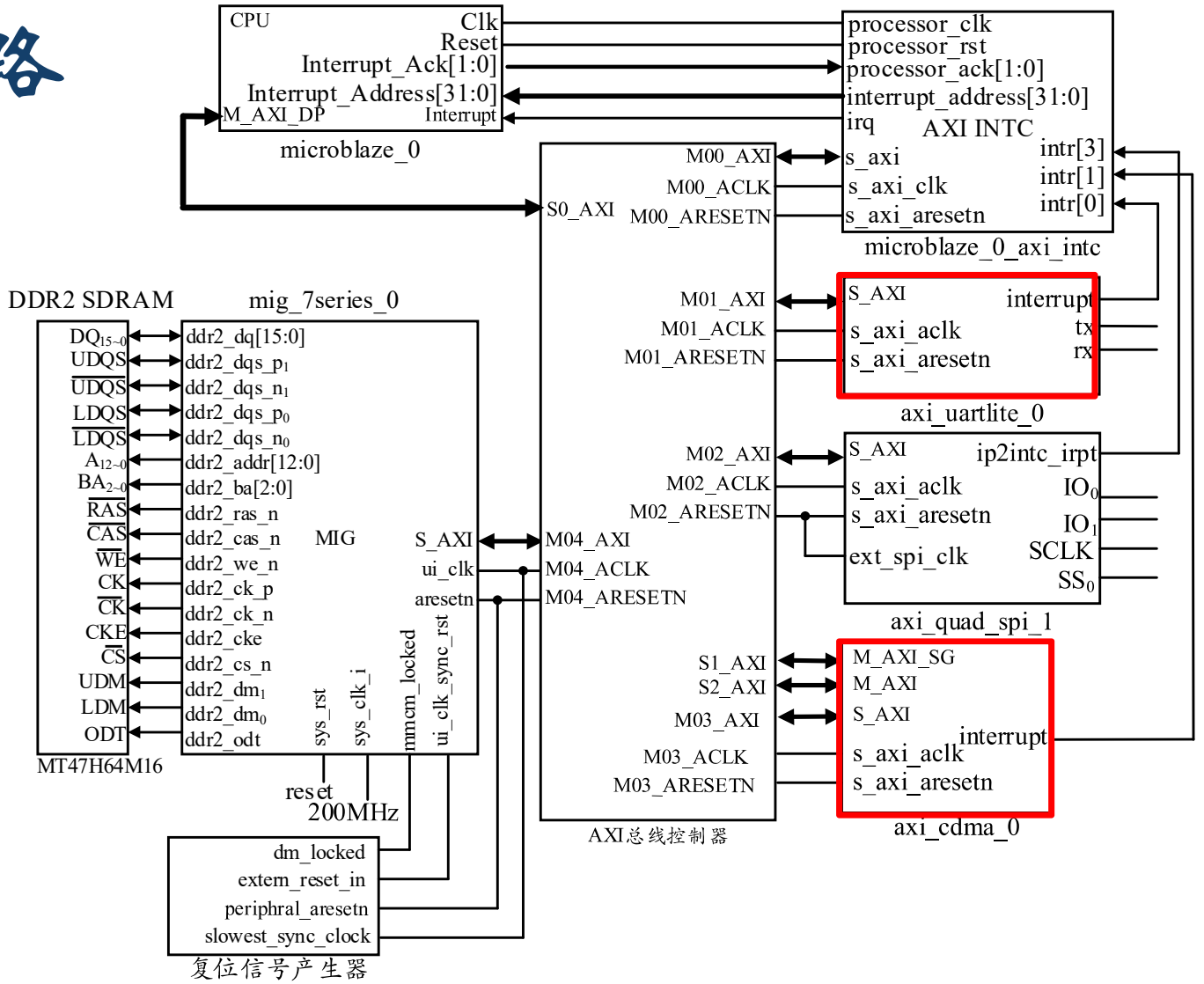


名称	偏移地址	含义
Rx FIFO	0x0	接收数据FIFO
Tx FIFO	0x4	发送数据FIFO
STAT_REG	0x8	状态寄存器
CTRL_REG	0xc	控制寄存器

UART STAT_REG含义

位	名称	含义
31~8	保留	无意义，全0
7	奇偶校验错	奇偶校验出错，0-无错；1-错
6	帧错误	停止位非0出错，0-无错；1-错
5	溢出错误	接收FIFO溢出，0-无溢出；1-溢出
4	中断使能状态	0-禁止；1-使能
3	Tx FIFO满	0-未满；1-满
2	Tx FIFO空	0-非空；1-空
1	Rx FIFO满	0-未满；1-满
0	Rx FIFO数据有效	0-空；1-有数据

应用示例电路


CPU
Clk
Reset
Interrupt_Ack[1:0]
Interrupt_Address[31:0]
M_AXI_DP
Interrupt
microblaze_0
processor_clk
processor_rst
processor_ack[1:0]
interrupt_address[31:0]
irq
AXI INTC
intr[3]
intr[1]
intr[0]
microblaze_0_axi_intc
M00_AXI
M00_ACLK
S0_AXI
M00_ARESETN
s_axi
s_axi_clk
s_axi_aresetn
DDR2 SDRAM
mig_7series_0
DQ15-0
UDQS
UDQS
LDQS
LDQS
A12-0
BA2-0
RAS
CAS
WE
CK
CK
CKE
CS
UDM
LDM
ODT
ddr2_dq[15:0]
ddr2_dqs_p1
ddr2_dqs_n1
ddr2_dqs_p0
ddr2_dqs_n0
ddr2_addr[12:0]
ddr2_ba[2:0]
ddr2_ras_n
ddr2_cas_n
ddr2_we_n
ddr2_ck_p
ddr2_ck_n
ddr2_cke
ddr2_cs_n
ddr2_dm1
ddr2_dm0
ddr2_odt
MIG
S_AXI
ui_clk
aresetn
sys_rst
sys_clk_i
mmcm_locked
ui_clk_sync_rst
MT47H64M16
reset
200MHz
M01_AXI
M01_ACLK
M01_ARESETN
S_AXI
s_axi_aclk
s_axi_aresetn
interrupt
tx
rx
axi_uartlite_0
M02_AXI
M02_ACLK
M02_ARESETN
S_AXI
s_axi_aclk
s_axi_aresetn
ext_spi_clk
ip2intc_irpt
IO0
IO1
SCLK
SS0
axi_quad_spi_1
M04_AXI
M04_ACLK
M04_ARESETN
S1_AXI
S2_AXI
M03_AXI
M03_ACLK
M03_ARESETN
M_AXI_SG
M_AXI
S_AXI
interrupt
s_axi_aclk
s_axi_aresetn
axi_cdma_0
AXI总线控制器
dm_locked
extern_reset_in
peripheral_aresetn
slowest_sync_clock
复位信号产生器

存储器到UART DMA传输

```
void Mem2UartDma(unsigned int Src,unsigned int btt)
{
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_CR_OFFSET,
        XAXICDMA_CR_KHOLE_WR_MASK|XAXICDMA_XR_IRQ_IOC_MASK
                        |XAXICDMA_XR_IRQ_ERROR_MASK);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_SRCADDR_OFFSET,Src);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_DSTADDR_OFFSET,
                        XPAR_UARTLITE_0_BASEADDR+XUL_TX_FIFO_OFFSET);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_BTT_OFFSET,btt);
}
```

UART到存储器DMA传输

```
void Uart2MemDma(unsigned int Dst)
{
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_CR_OFFSET,
                        XAXICDMA_CR_KHOLE_RD_MASK|XAXICDMA_XR_IRQ_IOC_MASK
                        |XAXICDMA_XR_IRQ_ERROR_MASK);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_DSTADDR_OFFSET,Dst);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_SRCADDR_OFFSET,
                        XPAR_UARTLITE_0_BASEADDR);
        while((Xil_In32(XPAR_UARTLITE_0_BASEADDR+XUL_STATUS_REG_OFFSET)
                        &XUL_SR_RX_FIFO_FULL)!=XUL_SR_RX_FIFO_FULL);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_BTT_OFFSET,0x40);
}
```

中断系统初始化

```
int main()
{       xil_printf("simple dma test\n");
        Xil_Out32(XPAR_UARTLITE_0_BASEADDR+XUL_CONTROL_REG_OFFSET,
                        XUL_CR_FIFO_RX_RESET|XUL_CR_FIFO_TX_RESET);
        Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IER_OFFSET,
                        XPAR_AXI_CDMA_0_CDMA_INTROUT_MASK);
        Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_MER_OFFSET,
                        XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
        microblaze_enable_interrupts();
        for(int i=0;i<15;i++)
                Xil_Out32(XPAR_MIG_7SERIES_0_BASEADDR+i*4,'a'+i);
        Xil_Out32(XPAR_MIG_7SERIES_0_BASEADDR+i*4,'\n');//初始化存储区数据
        Mem2UartDma(XPAR_MIG_7SERIES_0_BASEADDR,0x40);//存储器到IO DMA
        while(!DMADone);//等待DMA传输结束
        xil_printf("mem2uartdma done\n");
        DMADone=0;
        Uart2MemDma(XPAR_MIG_7SERIES_0_BASEADDR);// IO到存储器DMA
        while(!DMADone); //等待DMA传输结束
        xil_printf("uart2memdma done\n");
}
```

中断服务程序

```
void My_ISR(){
   int status;
        status=Xil_In32(XPAR_INTC_0_BASEADDR+XIN_ISR_OFFSET);
        if((status&XPAR_AXI_CDMA_0_CDMA_INTROUT_MASK)==
                        XPAR_AXI_CDMA_0_CDMA_INTROUT_MASK)
            DMAHandler();
        Xil_Out32(XPAR_INTC_0_BASEADDR+XIN_IAR_OFFSET,status);
}
void DMAHandler()
{
        int status;
        status=Xil_In32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_SR_OFFSET);
        Xil_Out32(XPAR_AXICDMA_0_BASEADDR+XAXICDMA_SR_OFFSET,status);
        if((status&XAXICDMA_XR_IRQ_IOC_MASK)==XAXICDMA_XR_IRQ_IOC_MASK)
        {
            DMADone=1;
        }
}
```

小结

- DMA传输基本原理
- CDMA传输控制流程——存储器映像IO
  - IO接口区分与存储器：数据端口地址不变