计算机组成原理与接口技术
——基于 MIPS 架构

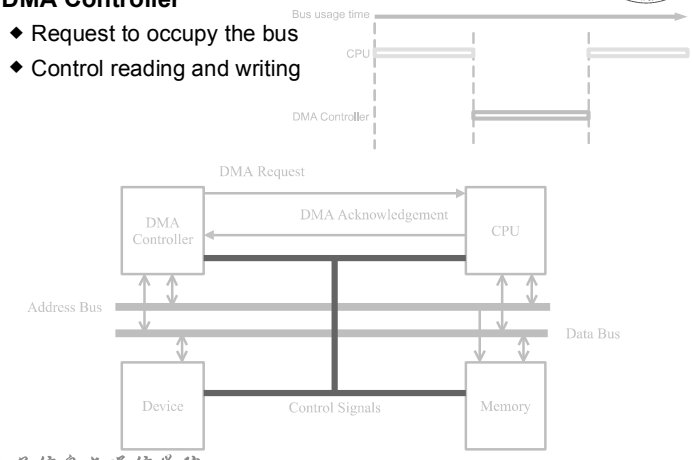
Chapter 9
DMA

张江山
zhangjs@hust.edu.cn
信息工程系

- **Content**
  - The Concept of Direct Memory Access
  - AXI CDMA Controller
- **Objectives**
  - Understanding the concepts of DMA
  - Master the method of using AXI CDMA controller

- **Comparison of 3 data transmission control modes**
  - Program Polling
    - CPU must query the status continuously
  - Interrupt
    - CPU can respond to IRQ only after completing an instruction
    - CPU must backup the current status
  - Direct Memory Access
    - CPU can respond to the DMA request at any time
    - CPU does not have to backup the current status
    - CPU does not access the bus temporarily
    - Transfer data directly between peripheral and memory
      - Memory → Memory: ***srcAddr*** and ***dstAddr*** be incremented
      - Memory → I/O Device: ***srcAddr*** be incremented, ***dstAddr*** be fixed
      - I/O Device → Memory: ***srcAddr*** be fixed, ***dstAddr*** be incremented

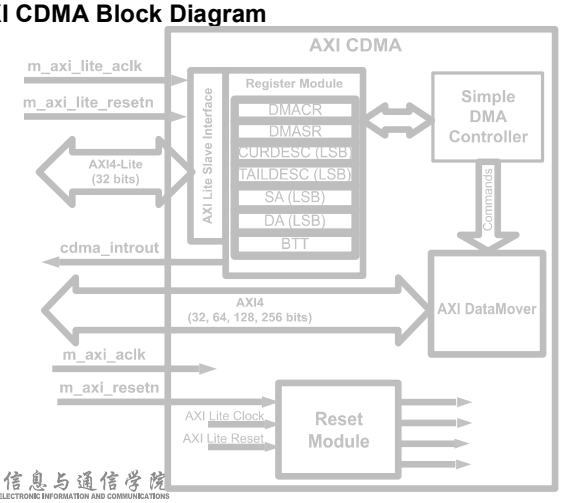
- **DMA Controller**
  - Request to occupy the bus
  - Control reading and writing



- **DMA Operations**
  1. Initialize DMAC
     - Transfer mode
     - Source and Destination address
     - Bytes to transfer
  2. DMAC Request
     - DMA Controller request → ***HOLD*** → CPU
  3. CPU Response
     - CPU response → ***HLDA*** → DMA Controller, CPU release bus
  4. Data Transter
     - DMAC read data from source to DMAC FIFO
     - DMAC write data from DMAC FIFO to dst address
     - DMAC count down
  5. Finish
     - DMAC → ***IRQ*** → CPU, DMAC release the bus

- **AXI CDMA (Central Direct Memory Access)**
  - AXI4 interface for data transfer
  - Independent AXI4-Lite Slave interface for register access
  - Fixed-address and Incrementing-address burst support
    - I/O device side: Fixed-address
    - Memory side: Incrementing-address
  - Default simple DMA mode
  - Optional Scatter-gather DMA mode support

- **AXI CDMA Block Diagram**



- **I/O Signals**

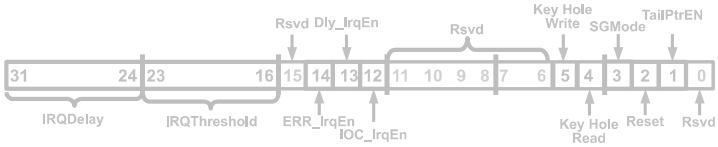
Signal	Type	Init	Description
cdma_introut	O	0	Interrupt output for the AXI CDMA core
m_axi_aclk	I	-	AXI CDMA Synchronization Clock
s_axi_lite_aclk	I	-	Synchronization Clock for the AXI4-Lite interface
s_axi_lite_aresetn	I	-	Active-Low AXI4-Lite Reset, be synchronous to s_axi_lite_aclk
s_axi_lite_*	I/O	-	AXI4-Lite Slave Interface Signals
m_axi_*	I/O	-	CDMA Data AXI4 Read/Write Master Interface Signals

- **Registers**

Offset	Register	Description
0x0	CDMACR	CDMA Control Register
0x4	CDMASR	CDMA Status Register
0x18	SA	Source Address Register
0x20	DA	Destination Address Register
0x28	BTT	Bytes to Transfer Register

- ***CDMACR*** (CDMA Control Register, Offset: **0x0**)

Bits	Field	Init	Description
2	Reset	0	Soft reset control for the AXI CDMA core
3	SGMode	0	0 / 1: Simple / Scatter-gather DMA mode
4	Key Hole Read	0	0 / 1: Incremental / Fixed source address
5	Key Hole Write	0	0 / 1: Incremental / Fixed destination address
12	IOC_IrqEn	0	Complete Interrupt Enable
14	Err_IrqEn	0	Error Interrupt Enable
...	...	...	(Scatter-gather DMA mode and Reserved)

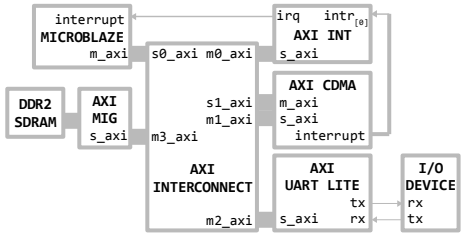


- ***CDMASR*** (CDMA Status Register, Offset: **0x4**)

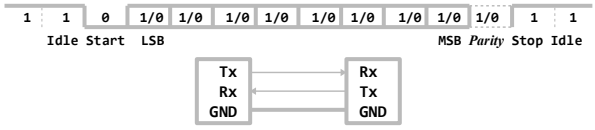
Bits	Field	Init	Access Type	Description
1	Idle	0	RO	CDMA Idle
4	DMAIntErr	0	RO	1: Internal error
5	DMASlvErr	0	RO	1: Slave error
6	DMADecErr	0	R/TOW	1: Decode error
12	IOC_Irq	0	R/TOW	Interrupt on complete
14	Err_Irq	0	R/TOW	Interrupt on error
Others	...	...	...	...

- ***BTT*** (CDMA Bytes to Transfer, Offset: **0x28**)
  - $BTT_{[22:0]}$: Bytes transmitted from *srcAddr* to *dstAddr*
  - $BTT_{[31:23]}$: Reserved
  - Writing to ***BTT*** register also initiates the Simple DMA transfer

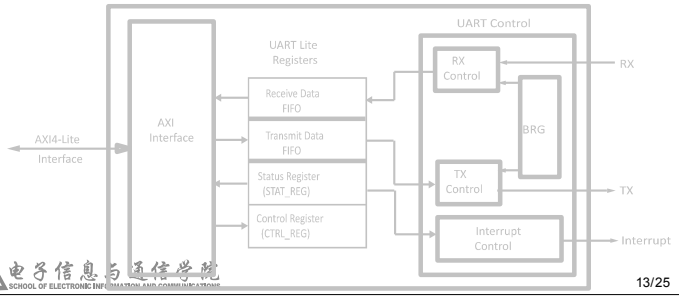
- **Ex.1:** 3 types of DMA transfer between device and memory
  - Memory to memory
  - Memory to UART I/O device
  - UART I/O device to memory



- **Ex.1:** 3 types of DMA transfer between device and memory
  - UART
    - Serial full-duplex, asynchronous communicates protocol
      - e.g. RS-232, RS-485
    - Four-wire serial bus
      - Vcc, GND, Tx, Rx



- **Ex.1:** 3 types of DMA transfer between device and memory
  - AXI UART Lite
    - AXI4-Lite interface for register access and data transfers
    - 16-character transmit and receive FIFOs
      - Interrupt is generated if transmit / receive FIFOs becomes empty / non-empty
    - Configurable number of data bits (5-8) in a character (frame)
    - Configurable parity bit (odd / even / none) and baud rate



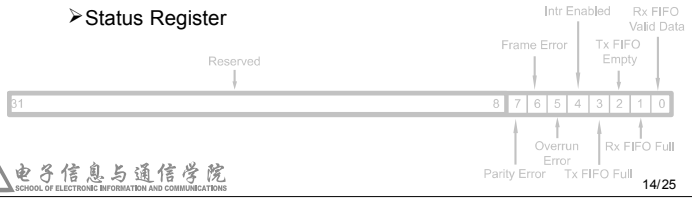
- **Ex.1:** 3 types of DMA transfer between device and memory
  - AXI UART Lite
    - Registers

Offset	Register	Description
0x0	RxFIFO	Receive Data FIFO
0x4	TxFIFO	Transmit Data FIFO
0x8	SR	UART Status Register
0xC	CR	UART Control Register

    - Control Register

Bits	Name	Type	Init	Description
0	Rst_Tx_FIFO	W	0	1: Reset the TxFIFO
1	Rst_Rx_FIFO	W	0	1: Reset the RxFIFO
4	Enable_Intr	W	0	1: Enable interrupt

    - Status Register

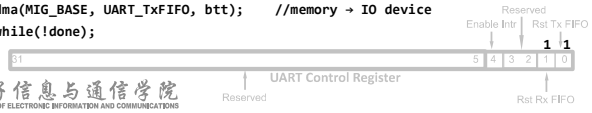


- **Ex.1:** 3 types of DMA transfer between device and memory

```
#include "xil_io.h"
#include "xil_print.h"
#define MIG_BASE     0x80000000
#define UART_BASE    0x41100000
#define INTC_BASE    0x41200000
#define CDMA_BASE    0x44A00000
#define UART_RxFIFO (UART_BASE + 0x0)  //UART Receive Data FIFO
#define UART_TxFIFO (UART_BASE + 0x4)  //UART Transmit Data FIFO
#define UART_SR     (UART_BASE + 0x8)  //UART Status Register
#define UART_CR     (UART_BASE + 0xC)  //UART Control Register
#define INTC_ISR    (INTC_BASE + 0x0)  //INTC Interrupt Status Register
#define INTC_IER    (INTC_BASE + 0x8)  //INTC Interrupt Enable Register
#define INTC_IAR    (INTC_BASE + 0xC)  //INTC Interrupt Acknowledge Register
#define INTC_MER    (INTC_BASE + 0x1C) //INTC Master Enable Register
#define CDMA_CR     (CDMA_BASE + 0x0)  //CDMA Control Register
#define CDMA_SR     (CDMA_BASE + 0x4)  //CDMA Status Register
#define CDMA_SA     (CDMA_BASE + 0x18) //CDMA Source Address Register
#define CDMA_DA     (CDMA_BASE + 0x20) //CDMA Destination Address Register
#define CDMA_BTT    (CDMA_BASE + 0x28) //CDMA Byters to Transfer
```

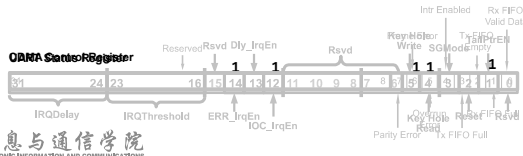

- **Ex.1:** 3 types of DMA transfer between device and memory

```
u32 k, done = 0, btt = 64;
void MyISR(void) __attribute__ ((interrupt_handler));
void dma(u32 src, u32 dst, u32 btt);
void main(void) {
  Xil_Out32(UART_CR, 3);  //set UART Control Register, reset UART TxFIFO and RxFIFO
  Xil_Out32(INTC_IAR, 1); //set INTC Interrupt Register, clear INTC ISR[0]
  Xil_Out32(INTC_IER, 1); //set INTC Interrupt Register, enable INTC intr[0]
  Xil_Out32(INTC_MER, 3); //set INTC Master Enable Register, enable INTC irq & intr
  microblaze_enable_interrupts();
  for(k = 0; k < btt / 4 - 1; k++)        //initialize 16 characters in memory
    Xil_Out8(MIG_BASE + k * 4, 'a' + k);  //"abcdefghijklmnop\n"
  Xil_Out8(MIG_BASE + k * 4, '\n');
  dma(MIG_BASE, MIG_BASE + btt, btt); //memory → memory
  while(!done);
  dma(UART_RxFIFO, MIG_BASE, btt);    //IO device → memory
  while(!done);
  dma(MIG_BASE, UART_TxFIFO, btt);    //memory → IO device
  while(!done);
}
```



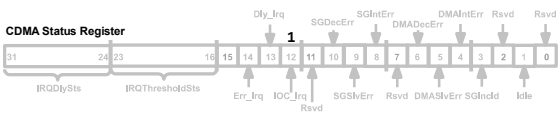
- **Ex.1:** 3 types of DMA transfer between device and memory

```
void dma(u32 src, u32 dst, u32 btt) {
  done = 0;
  if(src == UART_RxFIFO)          //IO device → memory
    Xil_Out32(CDMA_CR, 0x5010);   //enable Error & Complete Irq, fixed src addr
  else if(dst == UART_TxFIFO)     //memory → IO device
    Xil_Out32(CDMA_CR, 0x5020);   //enable Error & Complete Irq, fixed dest addr
  else                            //memory → memory
    Xil_Out32(CDMA_CR, 0x5000);   //enable Error & Complete Irq
  Xil_Out32(CDMA_SA, src);        //set CDMA Source Address Register
  Xil_Out32(CDMA_DA, dst);        //set CDMA Destination Address Register
  //if (IO device → memory) & (RxFIFO is not full), wait
  while((src == UART_RxFIFO) && (!(Xil_In32(UART_SR)&2)));
  Xil_Out32(CDMA_BTT, btt);       //bytes to transmitted, start transmission
}
```



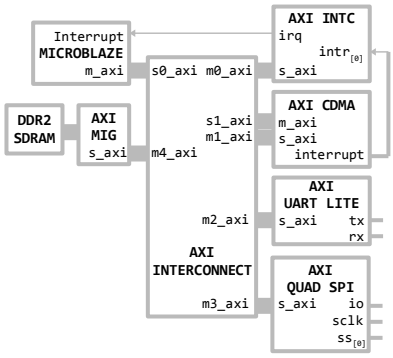
- **Ex.1:** 3 types of DMA transfer between device and memory

```
void MyISR(void) __attribute__ ((interrupt_handler)) {
  u32 state = Xil_In32(INTC_ISR);
  Xil_Out32(INTC_IAR, state);  //clear INTC Status Register
  if((state & 1) {             //if state[0], handle INTC intr[0]
    state = Xil_In32(CDMA_SR);
    Xil_Out32(CDMA_SR, state); //clear CDMA Status Register (TOW)
    if(state & 0x1000) {       //if state[12], handle CDMA Completed Irq
      done = 1;
      xil_prinf("dma done\n");
    }
    else
      xil_prinf("dma error\n");
  }
}
```



- **Scatter-gather Mode**
  - For off-loading CPU management tasks to hardware automation
  - AXI4 read/write master interface I/O signals: ***m_axi_sg_**
    - Fetches and updates DMA control transfer descriptors from memory
  - Provides internal descriptor queuing
    - Tansfer Descriptor Word (16 32-bit words)
      - $NXTDESC\_PNTR_{[31:6]}$: Next Descriptor Pointer (**0x0**)
        Descriptor address must be aligned to 64-byte boundaries (16 32-bit words)
        e.g. 0x00, 0x40, 0x80, 0xC0 ...
      - $SA_{[31:0]}$: Source Address (**0x8**)
      - $DA_{[31:0]}$: Destination Address (**0x10**)
      - $CONTROL_{[22:0]}$: Bytes to Transfer (**0x18**)
      - $STATUS_{[31:28]}$: *Cmplt*, *DMADecErr*, *DMASlvErr*, *DMAIntErr* (**0x1C**)
  - Registers
    - $CURDESC\_PNTR_{[31:6]}$: Current Descriptor Pointer (**0x8**)
    - $TAILDESC\_PNTR_{[31:6]}$: Tail Descriptor Pointer (**0x10**)
      - Start the channel fetching and processing descriptors after writing

- **Ex.2:** DMA transfer 64 bytes from DDR2-SDRAM to the TxFIFOs of UARTLite and SPI with Scatter-gather Mode

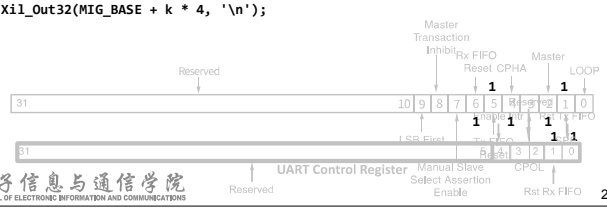


- **Ex.2:** DMA transfer 64 bytes from DDR2-SDRAM to the TxFIFOs of UARTLite and SPI with Scatter-gather Mode

```
#include "xil_io.h"
#include "xil_print.h"
#define MIG_BASE    0x80000000
#define DESC_BASE   (MIG_BASE + 0x03000000) //Destination Base Address
#define UART_BASE   0x41100000
#define SPI_BASE    0x41200000
#define INTC_BASE   0x41300000
#define CDMA_BASE   0x44A00000
#define UART_TxFIFO (UART_BASE + 0x4)  //UART Transmit Data FIFO Register
#define UART_SR     (UART_BASE + 0x8)  //UART Status Register
#define UART_CR     (UART_BASE + 0xC)  //UART Control Register
#define SPI_CR      (SPI_BASE + 0x60)  //SPI Control Register
#define SPI_DTR     (SPI_BASE + 0x68)  //SPI Data Transmit Register
#define SPI_SSR     (SPI_BASE + 0x70)  //SPI Slave Select Register
#define INTC_ISR    (INTC_BASE + 0x0)  //INTC Interrupt Status Register
#define INTC_IER    (INTC_BASE + 0x8)  //INTC Interrupt Enable Register
#define INTC_IAR    (INTC_BASE + 0xC)  //INTC Interrupt Acknowledge Register
#define INTC_MER    (INTC_BASE + 0x1C) //INTC Master Enable Register
#define CDMA_CR     (CDMA_BASE + 0x0)  //CDMA Control Register
#define CDMA_SR     (CDMA_BASE + 0x4)  //CDMA Status Register
#define CDMA_SA     (CDMA_BASE + 0x18) //CDMA Source Address Register
#define CDMA_DA     (CDMA_BASE + 0x20) //CDMA Destination Address Register
#define CDMA_BTT    (CDMA_BASE + 0x28) //CDMA Byters to Transfer Register
#define CDMA_CDESC  (CDMA_BASE + 0x8)  //CDMA Current Descriptor Pointer Register
#define CDMA_TDESC  (CDMA_BASE + 0x10) //CDMA Tail Descriptor Pointer Register
```

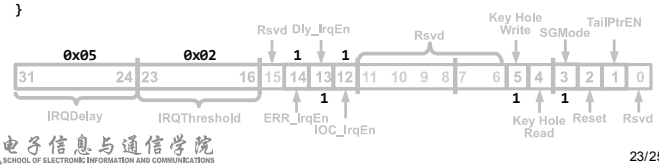

- **Ex.2:** DMA transfer 64 bytes from DDR2-SDRAM to the TxFIFOs of UARTLite and SPI with Scatter-gather Mode

```
u32 k, done = 0, btt = 64;
void MyISR(void) __attribute__ ((interrupt_handler));
void main(void) {
  Xil_Out32(UART_CR, 3);  //set UART Control Register, reset UART TxFIFO and RxFIFO
  Xil_Out32(SPI_CR, 0x76); // CPHA=1, CPOL=0, master, SPI enable, reset Tx/RxFIFO
  Xil_Out32(SPI_SSR, 0);   // set slave select register
  Xil_Out32(INTC_IAR, 1); //set INTC Interrupt Register, clear INTC ISR[0]
  Xil_Out32(INTC_IER, 1); //set INTC Interrupt Register, enable INTC intr[0]
  Xil_Out32(INTC_MER, 3); //set INTC Master Enable Register, enable INTC irq & intr
  microblaze_enable_interrupts();
  for(k = 0; k < btt / 4 - 1; k++)        //initialize 16 characters in memory
    Xil_Out32(MIG_BASE + k * 4, 'a' + k);  //"abcdefghijklmnop\n"
  Xil_Out32(MIG_BASE + k * 4, '\n');
```



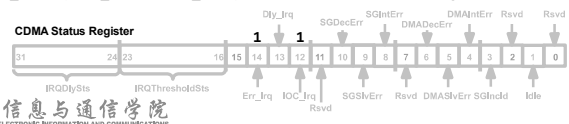
- **Ex.2:** DMA transfer 64 bytes from DDR2-SDRAM to the TxFIFOs of UARTLite and SPI with Scatter-gather Mode

```
  Xil_Out32(DESC_BASE, DESC_BASE + 0x40);   //next Desc. Pointer in the 1st Desc.
  Xil_Out32(DESC_BASE + 0x8, MIG_BASE);     //source Address in the 1st Desc.
  Xil_Out32(DESC_BASE + 0x10, UART_TxFIFO);//destication Address in the 1st Desc.
  Xil_Out32(DESC_BASE + 0x18, btt);         //byte to tran. in the 1st Desc.
  Xil_Out32(DESC_BASE + 0x1C, 0);           //status in the 1st Desc.
  Xil_Out32(DESC_BASE + 40, DESC_BASE + 0x80); //next desc. Pointer of the 2nd Desc.
  Xil_Out32(DESC_BASE + 0x48, MIG_BASE);       //source address of the 2nd Desc.
  Xil_Out32(DESC_BASE + 0x50, SPI_DTR);        //dest. address of the 2nd Desc.
  Xil_Out32(DESC_BASE + 0x58, btt);            //control of the 2nd Desc.
  Xil_Out32(DESC_BASE + 0x5C, 0);              //status of the 2nd Desc.
  Xil_Out32(CDMA_CR, 0x05027028);   //enable Error & Complete Irq, fixed dest addr
  Xil_Out32(CDMA_CDESC, DESC_BASE);   //set CDMA current desc. pointer register
  Xil_Out32(CDMA_TDESC, DESC_BASE + 0x40);//set CDMA tail desc. Pointer register
  while((done < 2) && !Error);//
}
```



- **Ex.2:** DMA transfer 64 bytes from DDR2-SDRAM to the TxFIFOs of UARTLite and SPI with Scatter-gather Mode

```
void MyISR(void) __attribute__ ((interrupt_handler)) {
  u32 state_INTC = Xil_In32(INTC_ISR);
  if((state_INTC & 1) {              //if state[0], handle INTC intr[0]
    u32 state_CDMA = Xil_In32(CDMA_SR);
    Xil_Out32(CDMA_SR, state_CDMA); //clear CDMA Status Register (TOW)
    if(state_CDMA & 0x1000) {       //if state[12], handle CDMA Completed Irq
      for(int i = 0; i < 2; i++)
        if(Xil_In32(DESC_BASE + 0x40 * i + 0x1C) & 0x80000000)
          done++;  //check Desc., if DESC.status[31], Complete a transfer
      xil_prinf("dma done numbers is %d\n", done);
    }
    else if(state_CDMA & 0x4000)
      xil_prinf("dma error\n");
  }
  Xil_Out32(INTC_IAR, state_INTC);  //clear INTC Status Register
}
```



Homework

- P392
  - 10
  - 11