清華大學
Tsinghua University

软硬件接口

路由器实验团队

2020年11月

主要内容

Contents

- 软硬件接口意义
- 普通MMIO
- DMA
- 轮询与中断
- 转发表软硬件接口
- 网络软硬件接口

软硬件接口意义

- 软硬件各司其职，相互通讯，相互协助
  - 绝大多数IP分组通过转发引擎高效转发（数据平面）
  - 运行在CPU上的软件处理RIP路由协议（控制平面）
- 示例
  - 四千兆线速5.952Mpps，IP分组平均处理时间168ns，CPU及软件性能不足
  - OSPF、BGP或TCP等协议复杂，硬件实现困难
- 为此，转发引擎将部分IP分组传递至软件处理
  - 目标IP地址为路由器自身地址或RIP组播地址等情况
- 此外，软件自身需要访问网络，收发数据

回顾：串口的访问方式

- 某个地址范围背后不是内存，而是串口控制器
  - [0x10000000, 0x10000007]
- 这些地址对应串口控制器的一些“寄存器”
  - LSR (Line Status Register) ：串口控制器状态
  - THR (Transmitter Holding Register) ：串口发送缓冲区

```
WRITE_SERIAL:                          // 写串口：将a0的低八位写入串口
    li t0, COM1
.TESTW:
    lb t1, %lo(COM_LSR_OFFSET)(t0)     // 查看串口状态
    andi t1, t1, COM_LSR_THRE          // 截取写状态位
    bne t1, zero, .WSERIAL             // 状态位非零可写进入写
    j .TESTW                           // 检测验证，忙等待
.WSERIAL:
    sb a0, %lo(COM_THR_OFFSET)(t0)     // 写入寄存器a0中的值
    jr ra
```

回顾：组成原理实验地址映射

起始	结束	说明
0x80000000	0x800FFFFF	监控程序代码
0x80100000	0x803FFFFF	用户程序代码
0x80400000	0x807EFFFF	用户程序数据
0x807F0000	0x807FFFFF	监控程序数据
0x10000000	0x10000007	串口控制器
?	*?*	*网络*
?	*?*	*转发表*

- 复位后，CPU从0x80000000开始执行指令

回顾：地址映射的硬件支持

- 读（lw等指令）

```
BaseRAM_Addr = addr, ExtRAM_Addr = addr
if addr ∈ 基础SRAM地址:
  BaseRAM_OE_n = 0, data = BaseRAM_Data
elif addr ∈ 扩展SRAM地址:
  ExtRAM_OE_n = 0, data = ExtRAM_Data
elif addr == 串口状态寄存器:
  data = {CPLD_tbre & CPLD_tsre, 4'b0,
          CPLD_data_ready}
elif addr == 串口数据寄存器:
  CPLD_rdn = 0, data = BaseRAM_Data
```

回顾：地址映射的硬件支持

- 写（sw等指令）

```
BaseRAM_Addr = addr, ExtRAM_Addr = addr
BaseRAM_Data = data, ExtRAM_Data = data
if addr ∈ 基础SRAM地址:
  BaseRAM_WE_n = 0
elif addr ∈ 扩展SRAM地址:
  ExtRAM_WE_n = 0
elif addr ∈ 串口数据寄存器:
  CPLD_wrn = 0
```

普通MMIO

- MMIO全称Memory-Mapped I/O
- 与“串口”非常类似，软件通过统一的内存地址访问硬件接口，包括硬件提供的一些寄存器
- 优点
  - ✓编程简单
  - ✓与内存访问相似
  - ✓易于移植
- 局限
  - – 不方便快速传输大量数据

MMIO接口设计

- 将硬件映射至某段内存地址，并提供一些寄存器
- 寄存器设计示例
  - 状态寄存器：硬件当前状态，如串口发送端是否空闲
  - 数据寄存器：用于保存操作数，如串口需要发送的数据
  - 命令寄存器：保存操作命令，其被写入时硬件执行命令
  - 注：串口功能简单，对“数据寄存器”写入后，串口控制器即可开始发送，故无需命令寄存器。
  - 串口控制器寄存器示例

地址	位	说明
0x10000000	[7:0]	串口数据，读、写地址分别表示串口接收、发送一个字节
0x10000005	[5]	只读，为1时表示串口空闲，可发送数据
0x10000005	[0]	只读，为1时表示串口收到数据

MMIO接口驱动程序

- `volatile uint32_t *POINTER_TO_REGISTER;`
- 写寄存器
  - `*POINTER_TO_REGISTER = data;`
- 读寄存器
  - `data = *POINTER_TO_REGISTER;`
- 提示
  - 确定数据类型：`uint8_t`、`uint16_t`或`uint32_t`等
  - 代码需要使用`volatile`阻止编译器对该地址访问的优化
  - 若CPU可能乱序访问内存，代码需要使用`fence`指令（本实验的CPU一般不需要）

DMA

- DMA全称Direct Memory Access，**允许硬件直接访问内存**，不经过CPU中转，提高效率

✓ 节约CPU时间，CPU可同时执行其他任务

✓ 对于大量数据的传输（如网卡收发以太网帧），DMA效率更高

  – 硬件接收数据后可直接写入内存；硬件可直接读取内存数据并发送

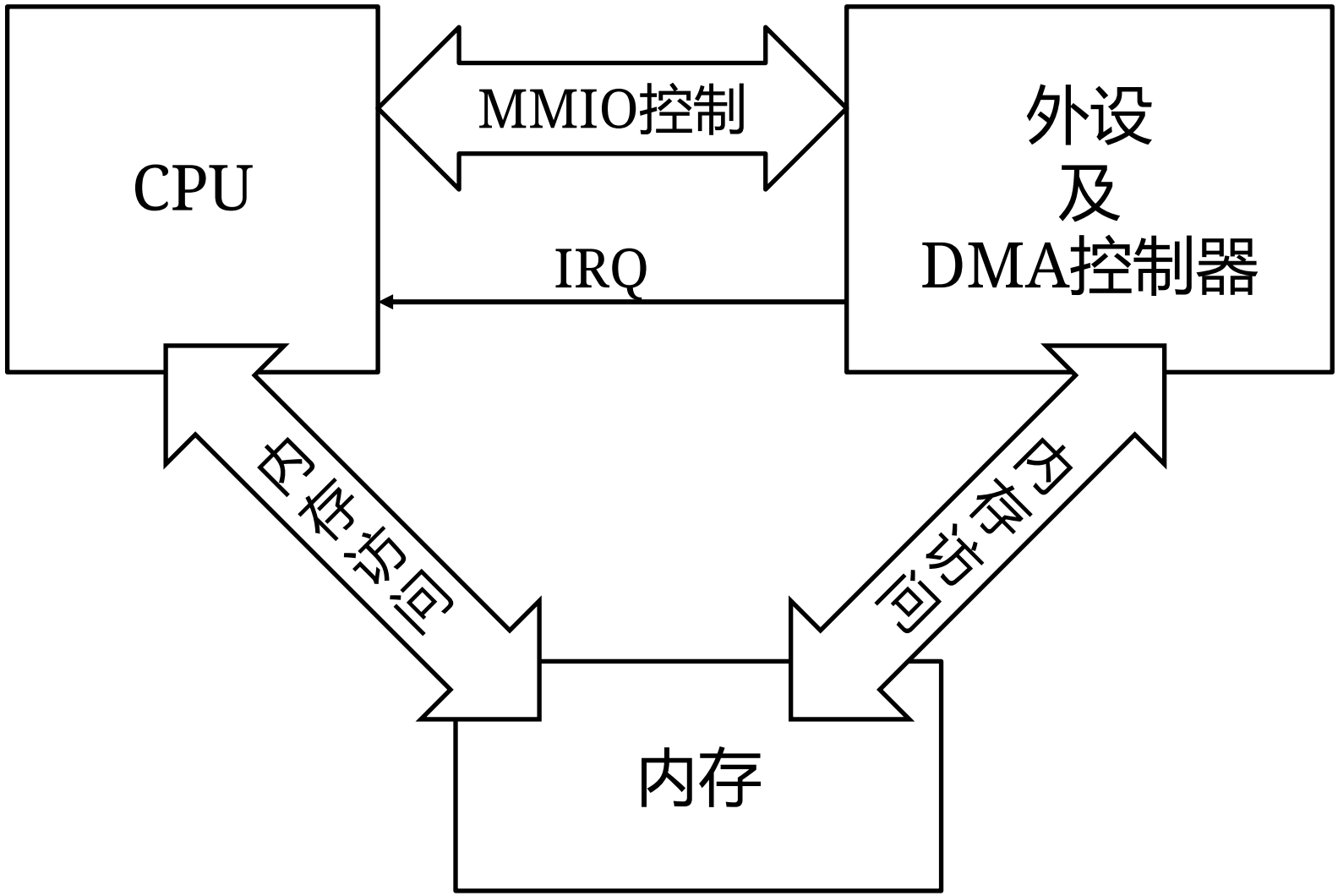
  – 普通MMIO一次读写一个字，效率欠佳

✓ 软件通过DMA接收数据后，可就地处理，然后再次通过DMA发送，减少拷贝次数，提高效率

DMA接口设计

- 基于MMIO，硬件提供寄存器用于控制DMA执行
- 寄存器设计示例
  - 状态寄存器：DMA当前状态，如是否完成数据传输
  - 缓冲区长度寄存器：接收或发送缓冲区的大小
  - 缓冲区基地址寄存器：软件提供的缓冲区的起始地址
- 若CPU实现了缓存，硬件访问内存时需注意缓存一致性问题（cache coherence）

DMA结构示意图



DMA工作流程

- 接收数据流程示例
  - 软件在内存中开辟一块缓冲区，并把起始地址写入基地址寄存器，同时把缓冲区大小写入长度寄存器
  - 硬件自动开始接收数据，并写入内存
  - 软件等待硬件DMA执行完成（等待方法后续介绍）
    - ✓等待的同时可执行其他任务
  - 软件读取长度寄存器获得数据长度，并进一步处理

DMA工作流程

- 发送数据流程示例
  - 软件将数据保存在内存中的缓冲区，并把起始地址写入基地址寄存器，同时把数据长度写入长度寄存器
  - 硬件自动开始读取内存，并发送数据
  - 软件等待硬件DMA执行完成（等待方法后续介绍）
    - ✓等待的同时可执行其他任务

轮询与中断

- 如何等待硬件DMA执行完成？
- 轮询
  - 软件定期读取状态寄存器，直到某些标志位满足条件
- 中断
  - 事件到来时，硬件主动通知CPU
- 普通MMIO也可用类似方法等待硬件操作完成或数据到来

中断

- 中断是硬件传递给CPU的信号
  - 边沿触发：出现一次高到低或低到高的跳变触发一次
  - 电平触发：特定电平（高或低）时不断触发，直到电平变为其他
- 常用于硬件通知CPU处理外部事件
  - 例如时钟、串口、键盘、鼠标、网络或存储的事件等
  - 触发频率高低不等
  - ✓CPU上的软件无需轮询，可以同时处理其他任务
- 底层代码设置中断处理程序
  - 可在《操作系统》课上进一步了解☺

转发表软硬件接口

- 普通MMIO
  - 参数寄存器：保存写入或读取的前缀及下一跳等信息
  - 命令寄存器：触发插入、更新、删除或查询操作，其被写入时硬件执行操作
- 共享内存
  - 暴露底层存储（RAM），将其映射至一段内存地址
  - 软件直接操作数据结构

网络软硬件接口

- 普通MMIO
  - 与串口非常类似
  - 软件每次收发一个字，硬件提供状态位标识帧的边界
- DMA
  - 提供状态寄存器、缓冲区长度寄存器、基地址寄存器
  - 硬件自动收发以太网帧，并访问内存
- 高级DMA
  - 实际网卡的接口
  - 实现复杂，性能更好

实际网卡的接口

- **前述DMA存在不足**：软件处理数据时，占用缓冲区，硬件不应访问相同缓冲区
- **解决方案**：多个缓冲区组成Ring Buffer，软硬件共同维护一个循环队列
  - 队列中每个元素为一个缓冲区的地址和大小
  - 软硬件分别维护一个指针，标识当前处理位置
  - 指针寄存器通过MMIO方法更新
  - 硬件可通过中断通知软件处理新数据

网络软硬件接口

- 思考：本实验中，将以太网帧传递给软件或软件发送以太网帧时，如何标识接收或发送接口？

本周任务

- 加油

清華大學
Tsinghua University

谢谢