计算机体系结构基础

胡伟武、苏孟豪

第06章 计算机总线接口技术

- 总线概述
- 总线分类
- 片上总线
- 内存总线
- IO总线

总线

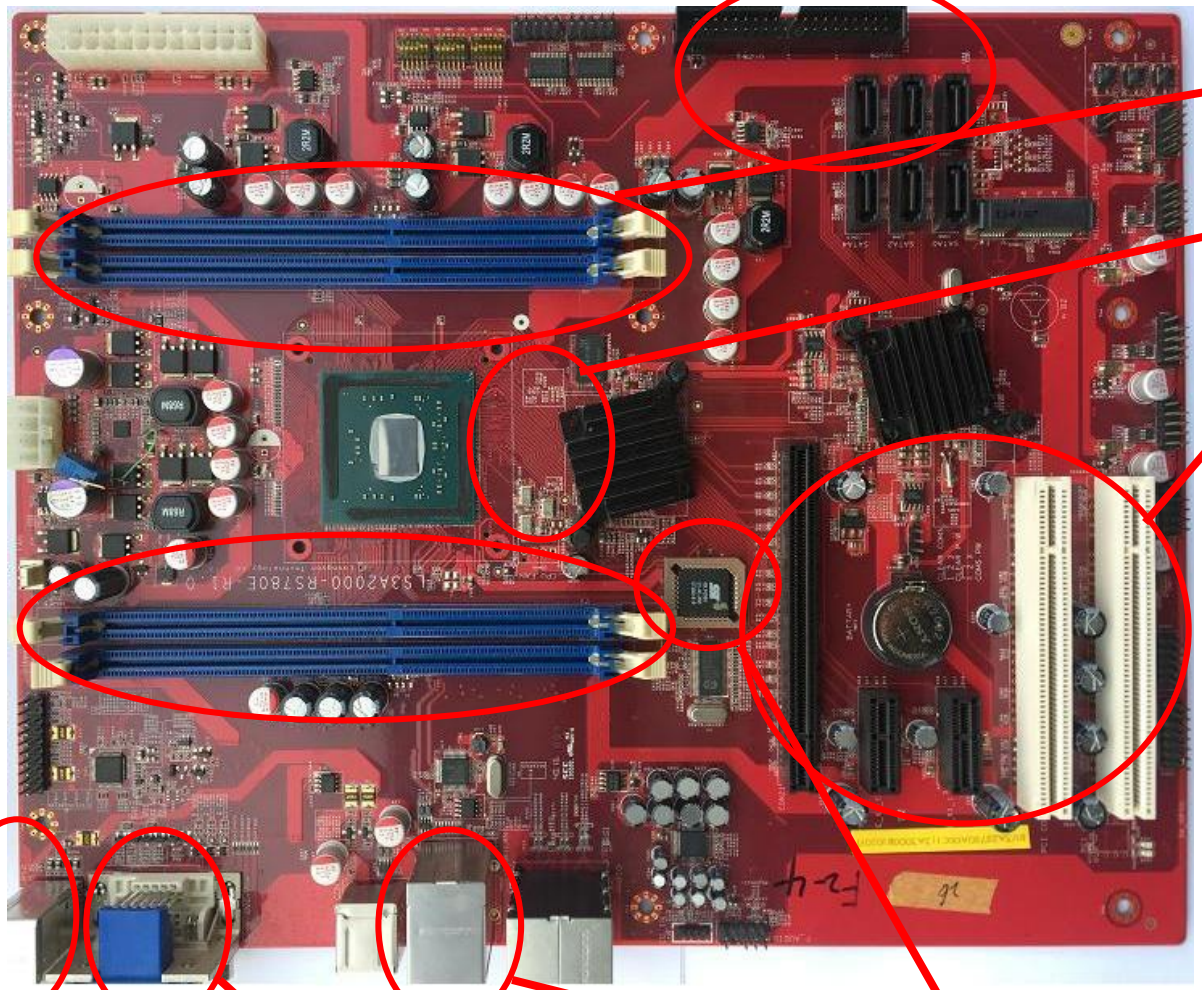
总线形态

- **1974年的S-100总线**
  - **使用100根线，并行连接所有插槽的对应引脚**
  - **包括电源、数据、地址、时钟及控制信号**
  - **提供可扩展能力，开始有“攒机”的概念**

ref: https://en.wikipedia.org/wiki/S-100_bus

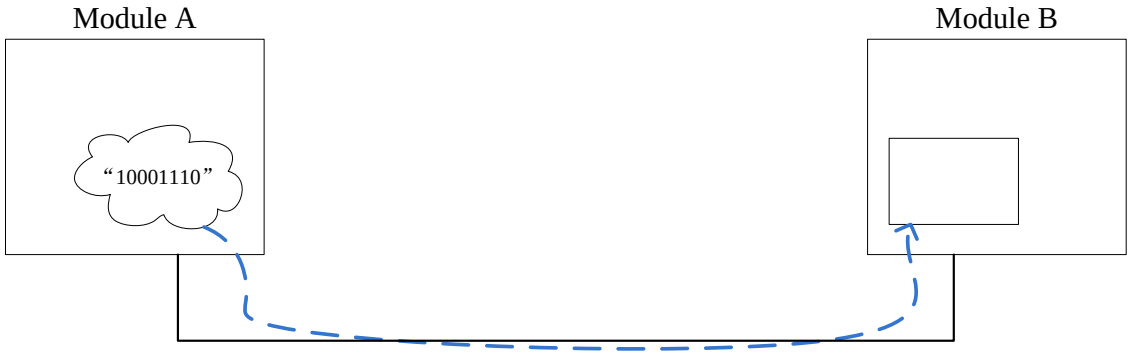
总线形态

- **3A2000主板**



总线概述

- 总线是什么？
  - 连接计算机各部件，用于数据传送的通讯系统
  - 本质作用是进行数据交换



总线概述

- 为了让不同厂家生产的部件相互匹配
  - 形成总线规范
  - 兼容性认证
- 总线规范包括以下层次：
  - 机械层：接口的外形、尺寸、信号排列、连接线的长度等等
  - 电气层：信号描述、电源电压、电平标准、信号质量等等
  - 协议层：信号时序、握手规范、命令格式、出错处理等等
  - 架构层：硬件模型、软件框架等等

总线分类

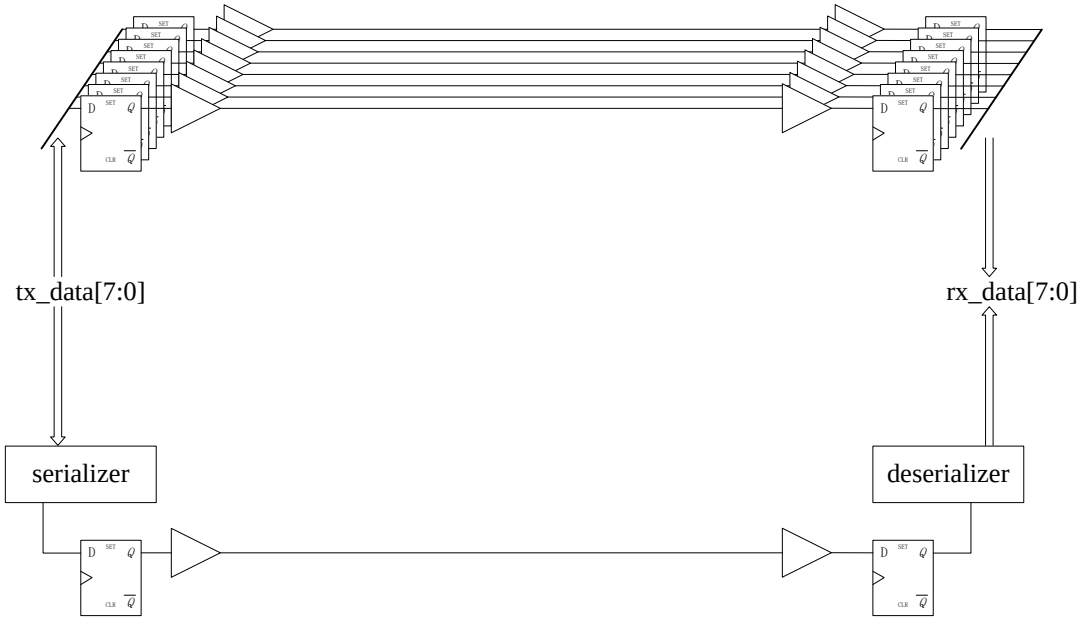
按数据传送方向

- 单向
  - **A→B单向传**
  - **VGA、DVI**
- 双向
  - 半双工（只有一个数据传输通道）
    - 同时只能进行一个方向传送，**A→B或B→A**
    - **USB 2.0**
  - 全双工（有两个数据传输通道）
    - 同时可以进行两个方向传送， **A→B和B→A**
    - **UART**

总线分类

按数据组织方式

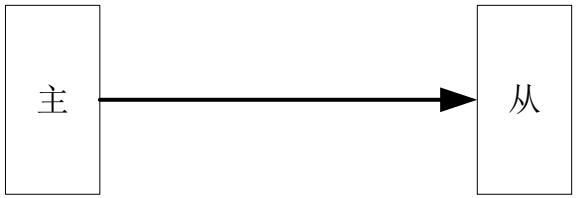
- 并行
  - 多字节位同时传
  - 需控制线间延迟差
  - 线多后提频困难
  - **PCI、HT**
- 串行
  - 字节按位传
  - 多条线(lane)之间无等长要求
  - **PCIe、SATA**



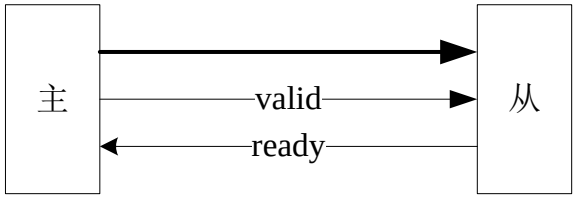
总线分类

按数据握手方式

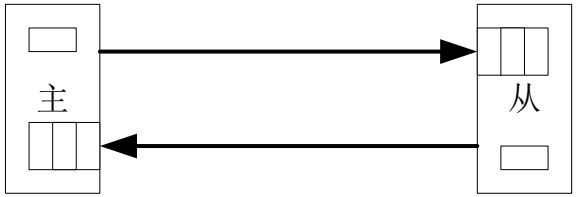
- 无
  - 发送端总在发，接收端无条件接收
  - **DVI、APB**



- **Valid-Ready**
  - 发送端、接收端均有“准备好”标识
  - 两边都准备好才能传输
  - **PCI、AXI**



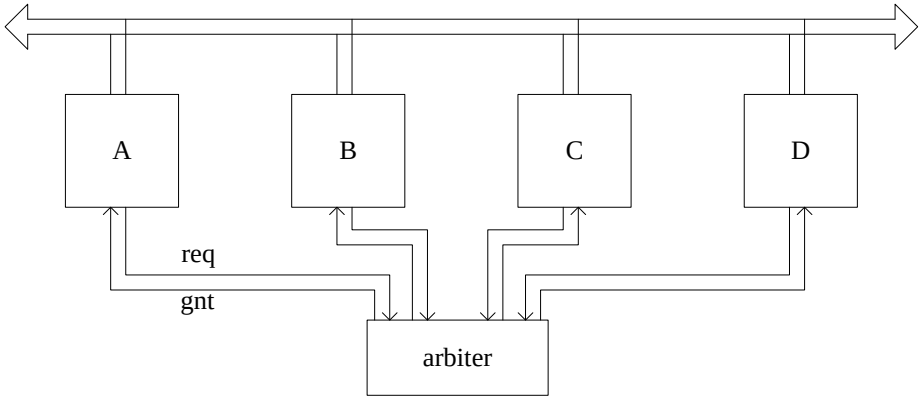
- **Credit**
  - 发送端跟踪接收端的可用缓冲数*(how?)*
  - 发送前先判断是否能够接收
  - **PCIe、HT**

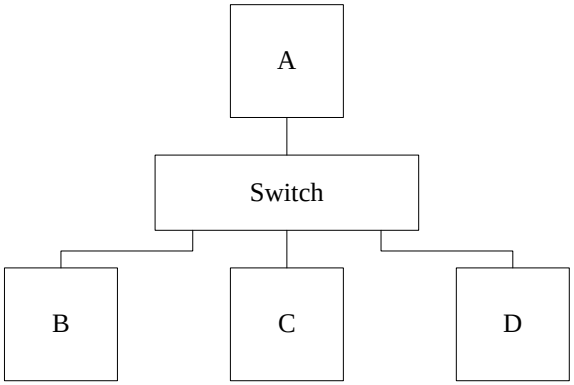


总线分类

按连接方式

- 共享信号
  - 同一组信号线
  - 通过仲裁占有总线
  - 三态输出(高、低、高阻)
  - **PCI**
- 点对点
  - 独占信号线
  - 专用的交换节点
  - **PCIe**

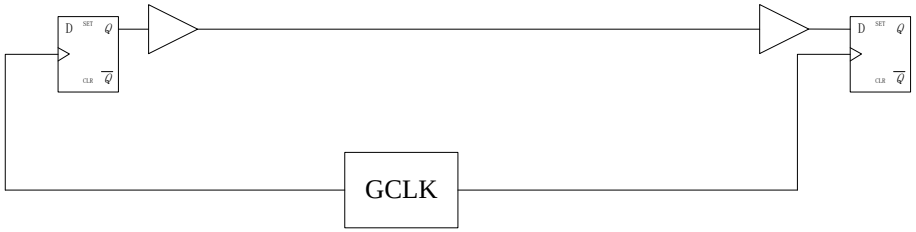


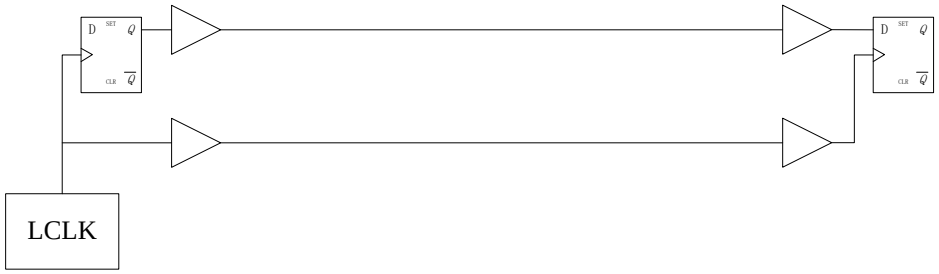


总线分类

按时钟实现方式

- 全局时钟
  - 时钟源到各部件路径等长
  - **PCI**
- 源同步
  - 时钟随数据一起发送
    - 时钟数据传输路径等长
    - **HT**
  - 时钟嵌入到数据中发送
    - 对发送的数据进行编码
    - 从接收数据中恢复时钟
    - **PCIe**

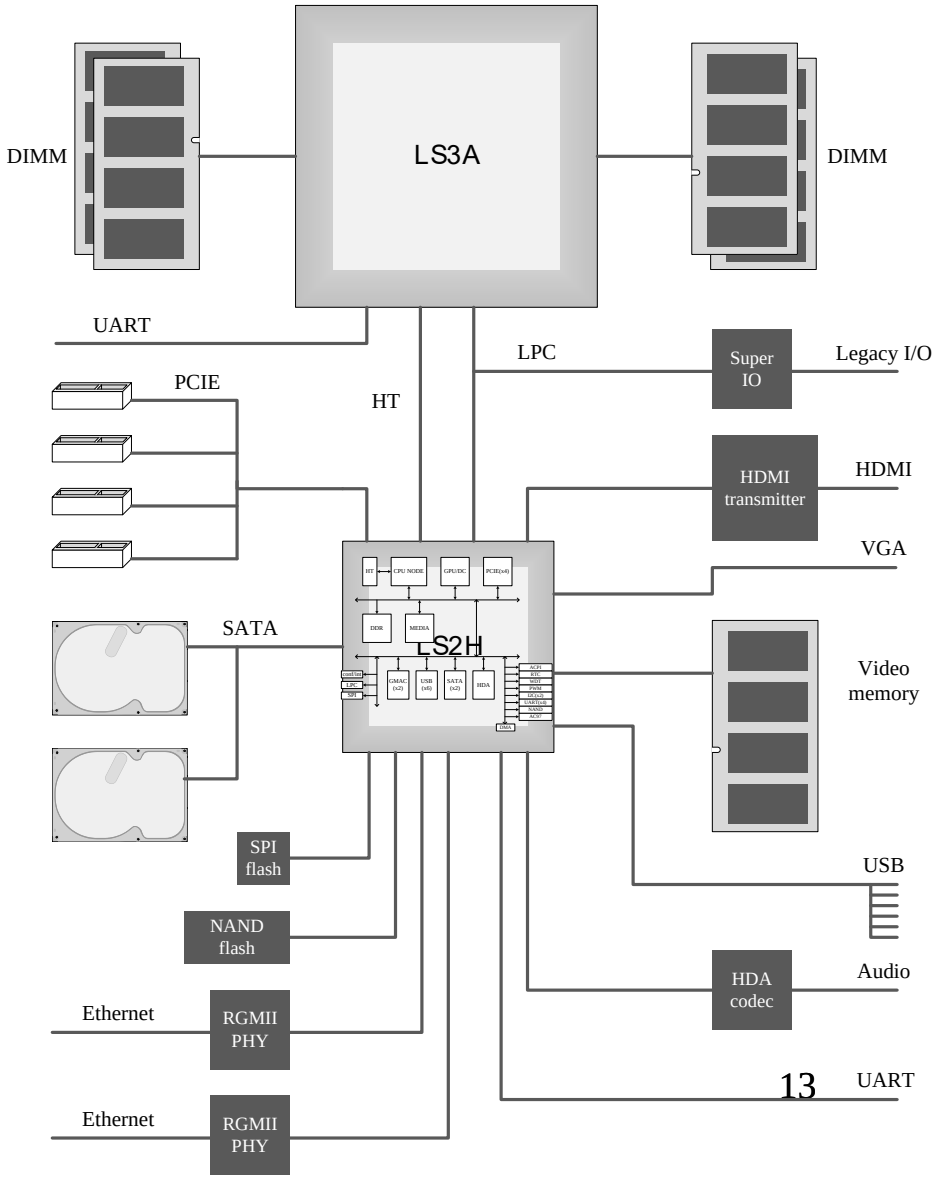




总线分类

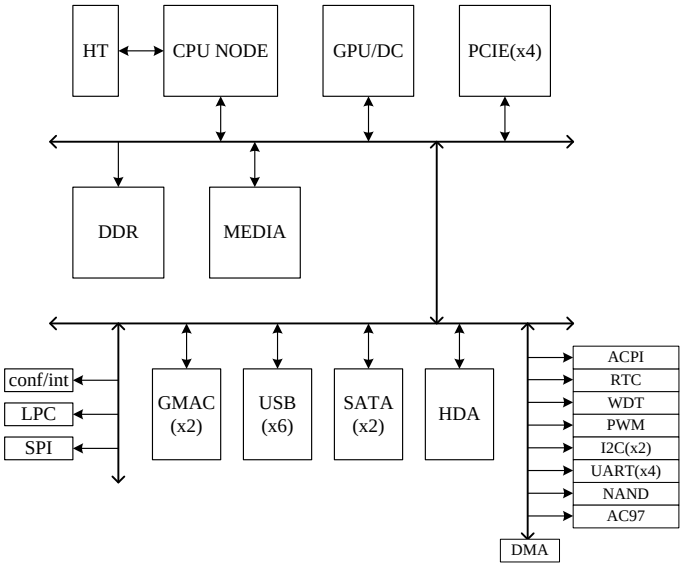
按总线实现位置

- 片上总线
- 内存总线
- **IO总线**



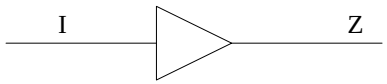
片上总线

- 芯片内部模块互联使用的总线
  - 处理器核
  - 内存控制器
  - IP核
- 支撑基本的读写操作
  - 将读写请求按地址送到目标模块
  - 将读写响应返回到发起模块
- 设计空间
  - 系统性能
  - 实现代价、复杂性

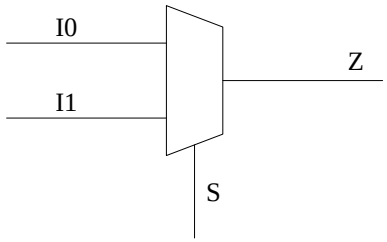


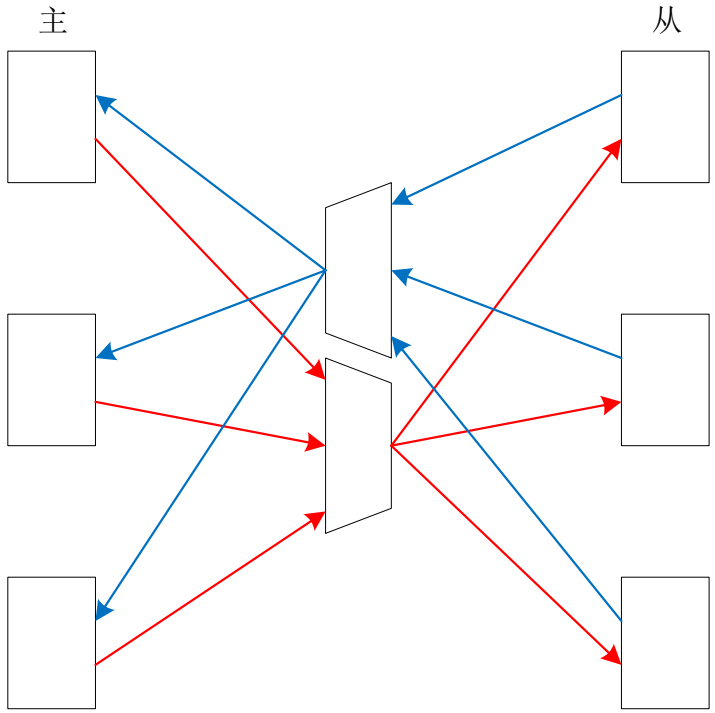
片上总线的特点

- 与片外实现相比
  - 引线资源丰富
  - 全局时钟相对容易实现
  - 不需要复杂的物理层转换
  - 不使用三态信号
- 基本连接单元
  - **buffer**



  - **mux**





“点对点”形式的共享信号连接

片上总线的性能

- 性能相关因素
  - 频率：能跑多快
    - 能否加流水级，加多少
  - 数据位宽：单位周期传送的数据量
    - **8/16/32/64/128/256/512 ...**
  - 带宽利用率：总线事务中数据传输时间占总时间的平均值
    - 读：主设备发出读请求到从设备，准备数据，通过总线返回
    - 写：主设备发出写请求到从设备，接收数据，返回状态
    - 延迟不容易降低，关键是能否流水，支持**outstanding**
    - 同时进行中的总线事务个数
- 性能目标决定了实现代价

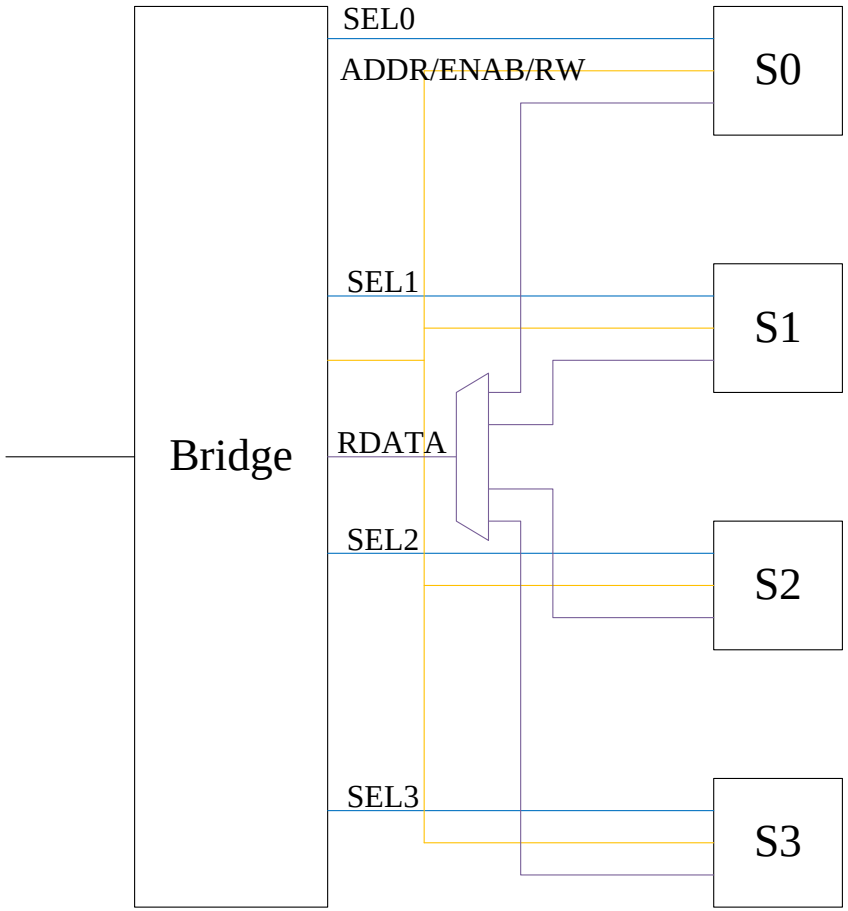
片上总线-AMBA

- **Advanced Microcontroller Bus Architecture**
  - **开放标准，广泛应用于SoC、ASIC、AP**

	AXI	AHB	APB
规范中含流水	未定义，地址与数据完全分离	地址周期比数据周期早一拍	
可增加流水级	√	×	×
常见数据宽度	32/64/128	32/64	8/32
突发传输长度	1～16	1～16～∞	1
支持outstanding	√	× split	×
总线架构	交换	共享	共享
仲裁方式	分布式	集中式	无
应用范围	高性能处理器	中等性能系统	低带宽寄存器类
实现代价	高	中	低

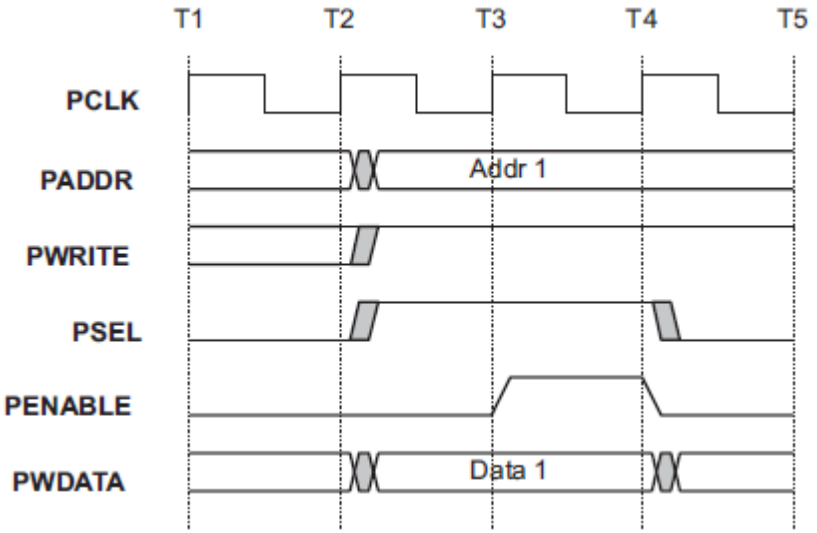
APB总线

- 单主设备
- 共享式
- 片选

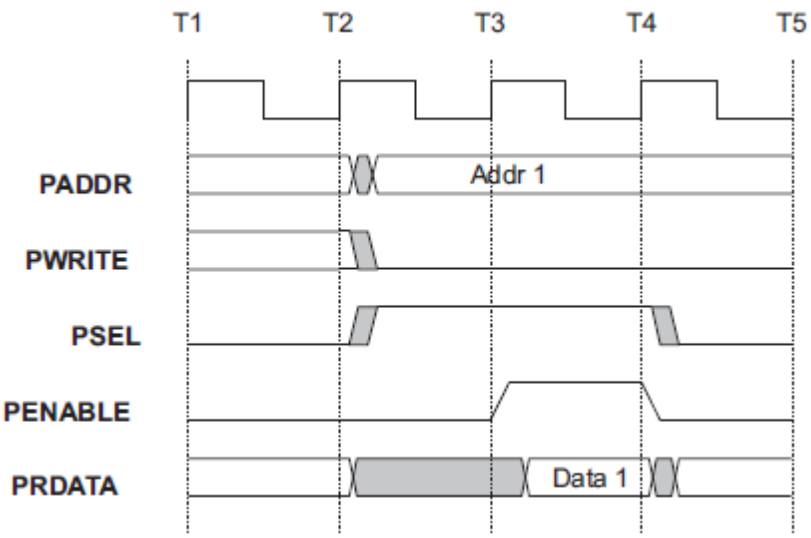


APB总线

- 固定的时序，不支持突发
- 没有数据握手，无法插入等待周期



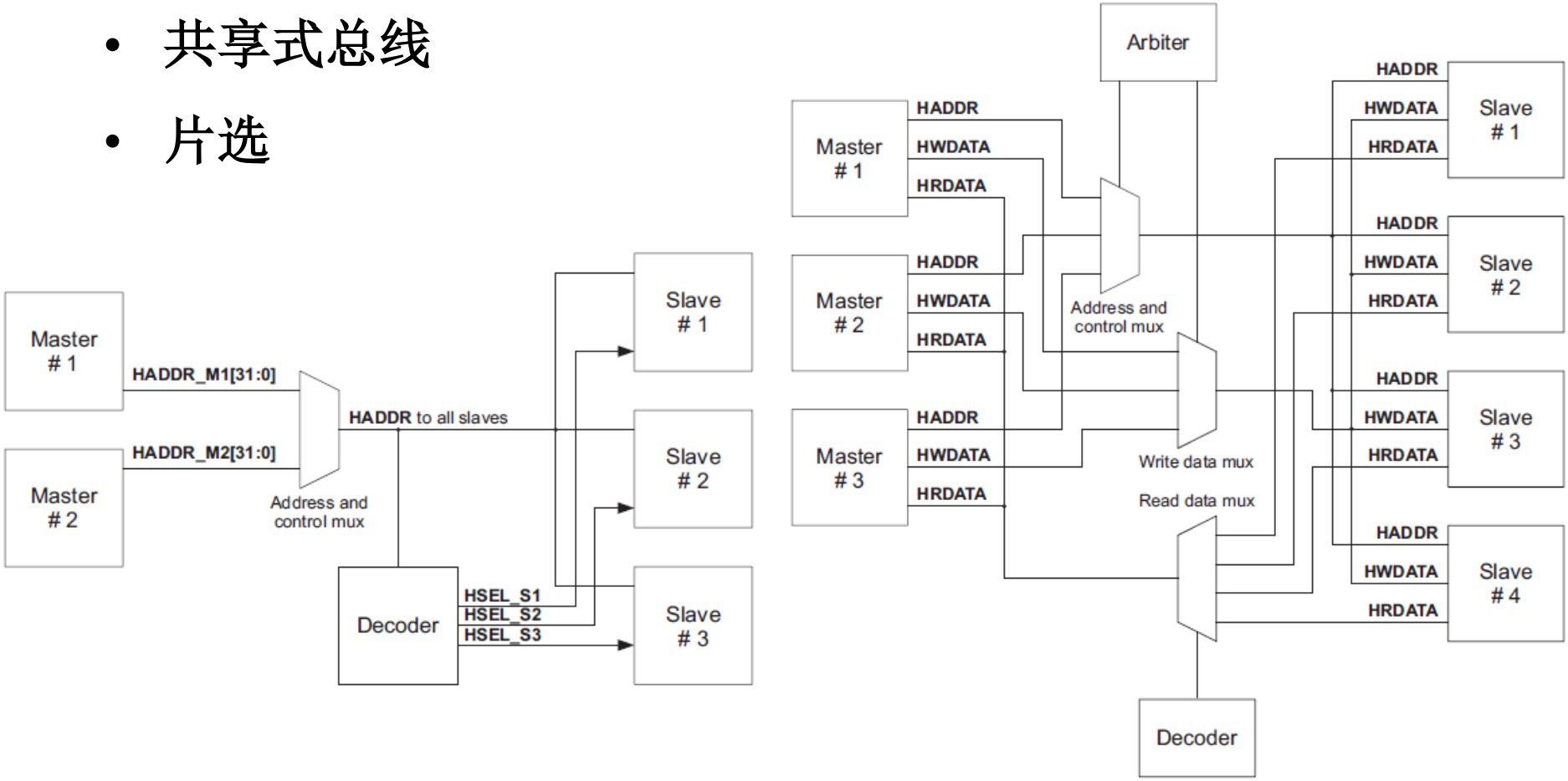
写



读

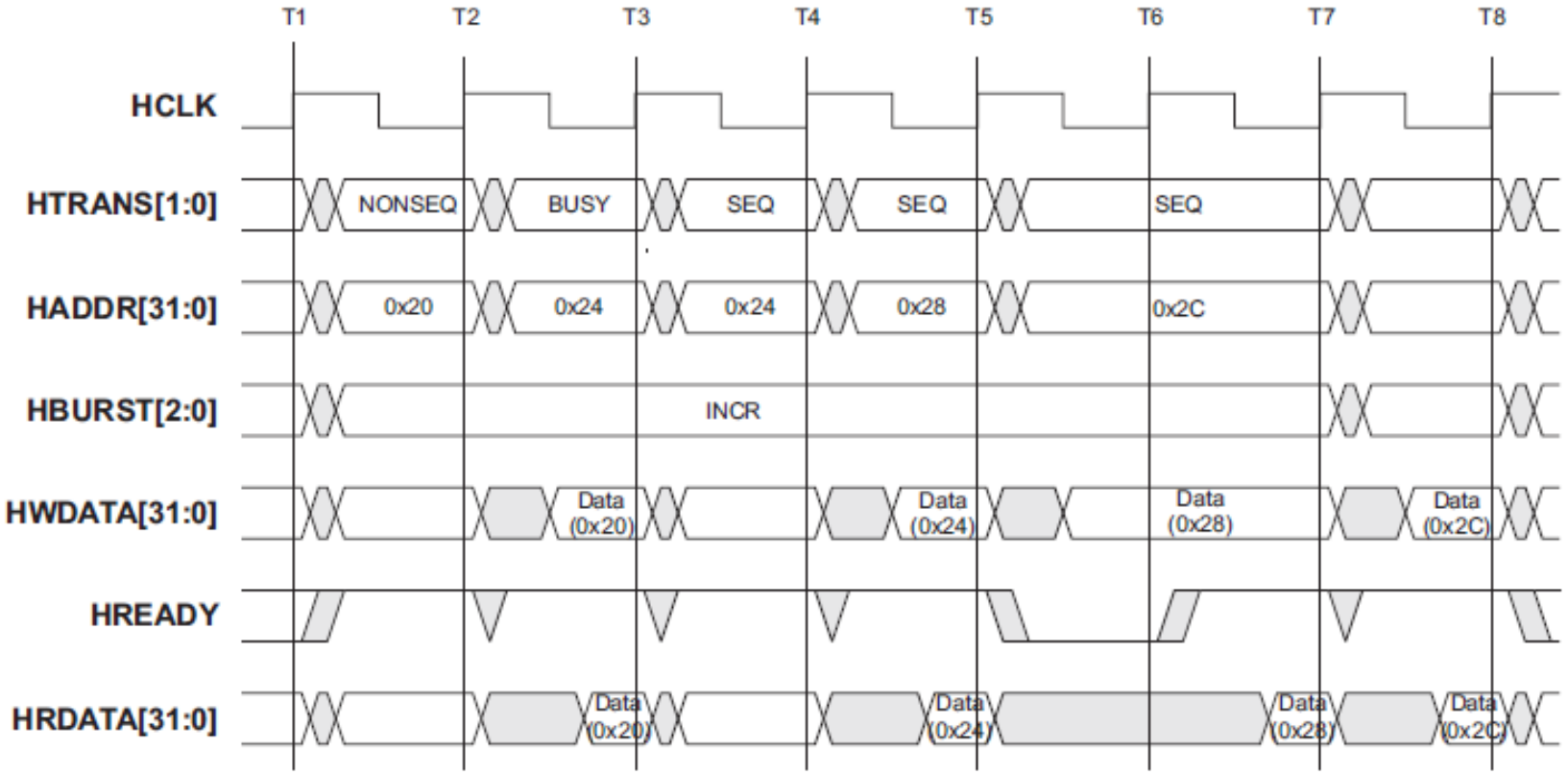
AHB总线

- 中央仲裁器、译码器
- 共享式总线
- 片选



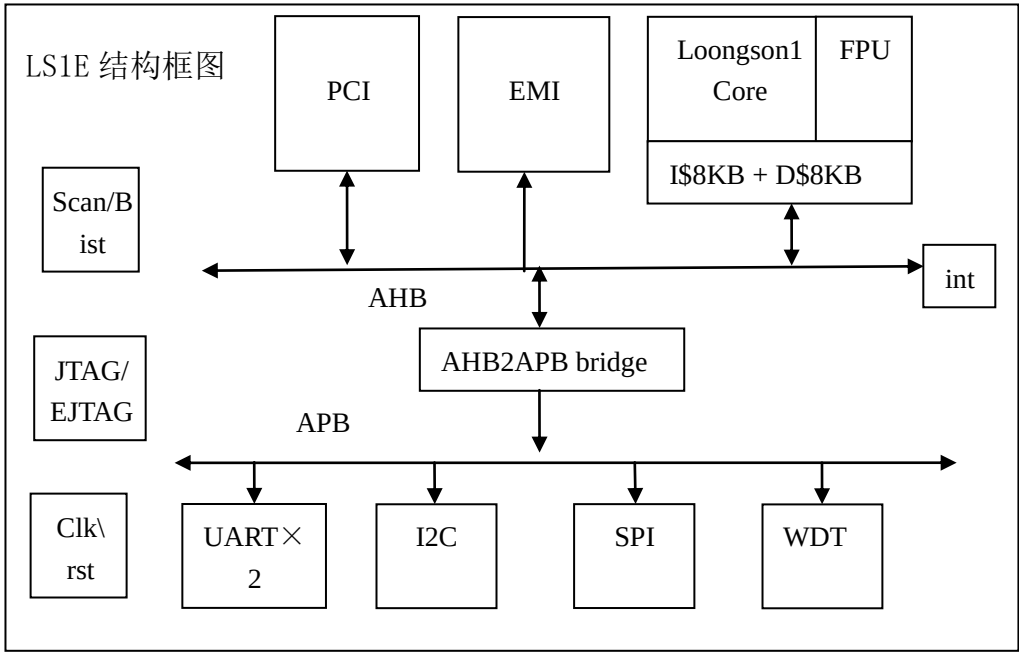
AHB总线

- 无独立的请求/数据通道
- 支持突发传输，数据周期落后地址周期一拍
- 主设备、从设备均可插入等待周期



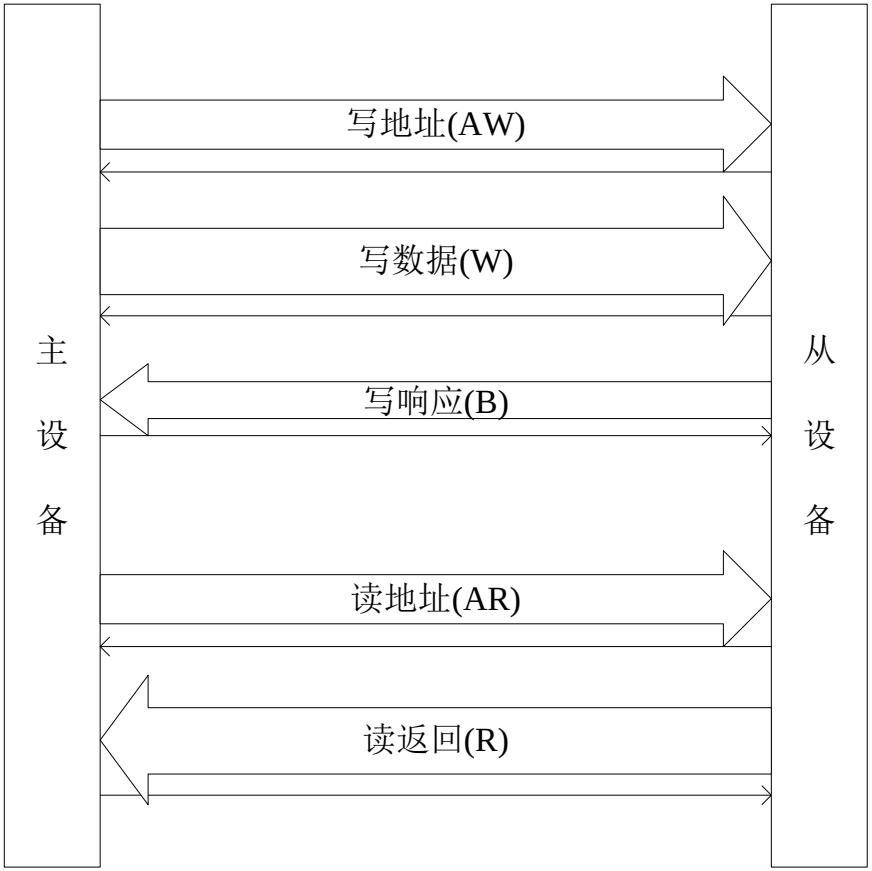
AHB+APB系统

- **龙芯1E**

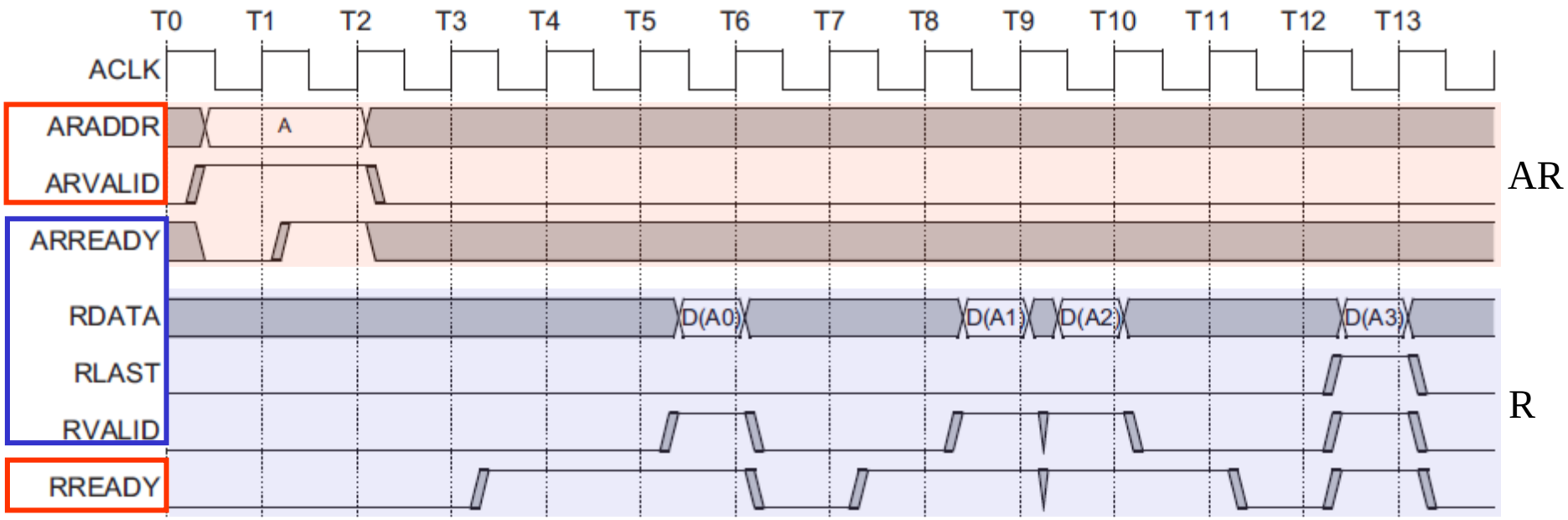


AXI总线架构

- 由五个通道组成
  - 写地址、写数据、写响应
  - 读地址、读返回
- 基本定义
  - 总线事务(transaction):
    - 一次完整读写过程
  - 传输(transfer):
    - 传输一个周期的数据
    - 在VALID、READY同时为高的周期完成一次传输



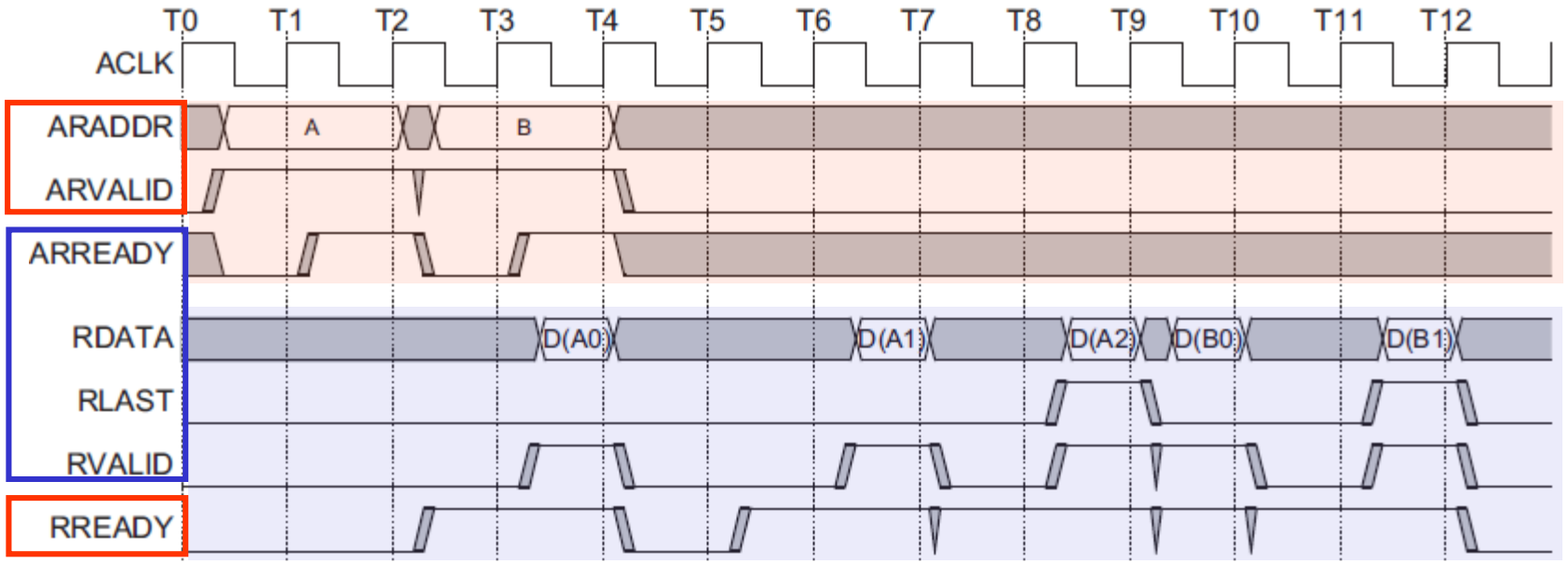
AXI事务示例——读



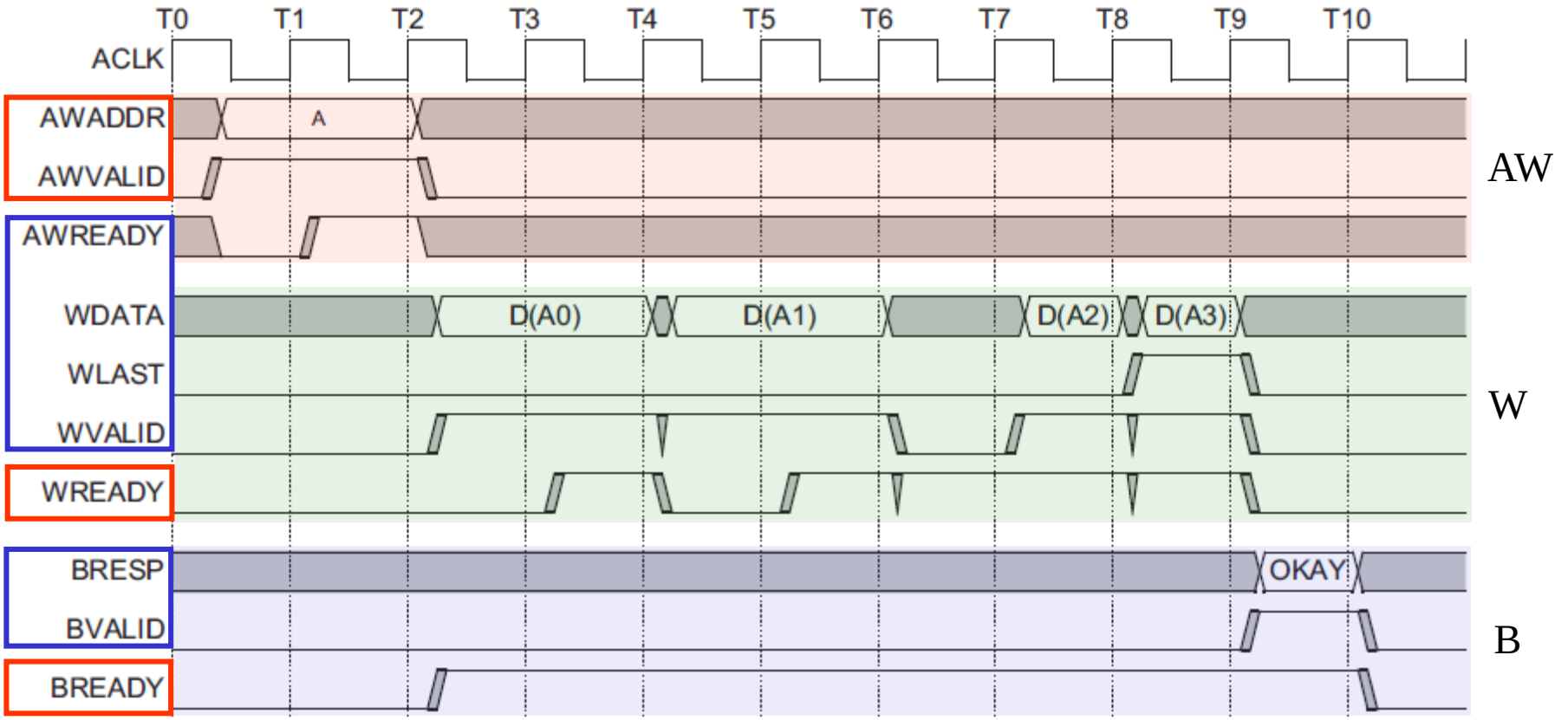
master driven

slave driven

AXI事务示例——重叠读


T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
ACLK
ARADDR
A
B
ARVALID
ARREADY
RDATA
D(A0)
D(A1)
D(A2)
D(B0)
D(B1)
RLAST
RVALID
RREADY
AR
R

AXI事务示例——写


T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
ACLK
AWADDR
A
AWVALID
AWREADY
WDATA
D(A0)
D(A1)
D(A2)
D(A3)
WLAST
WVALID
WREADY
BRESP
OKAY
BVALID
BREADY
AW
W
B

AXI关键信号

- **ID: 支持读写乱序**
  - **每个事务都有ID标签(ARID/RID/AWID/WID/BID)**
  - **同ID的事务必须按序**
  - **不同ID的事务可以乱序**
- **BURST/LEN/SIZE: 突发传输控制**
  - **突发传输类型: FIXED, INCR, WRAP**
  - **突发传输长度: 1~16**
  - **突发传输单位: 1,2,4~BW**
- **WSTRB: 写掩码**
  - **为高的位对应的字节写有效**
- **RESP: 读/写响应状态**

AXI关键信号

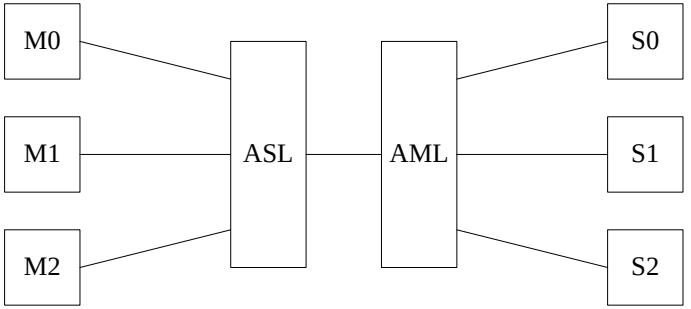
- **ARLOCK/AWLOCK: 原子性操作**
  - **00: 普通**
  - **01: 独占**
    - **独占地对某个地址先读后写，若期间无其它主设备写，则成功写入，否则失败，通过RESP返回状态。类似LL/SC**
  - **10: 加锁**
    - **从第一个加锁事务开始，到第一个普通事务结束**
    - **将所访问的从设备区域加锁，阻止其它主设备访问**
- **ARPROT/AWPROT: 访问保护**
  - **[0]: 普通/特权**
  - **[1]: 安全/非安全**
  - **[2]: 数据/指令**

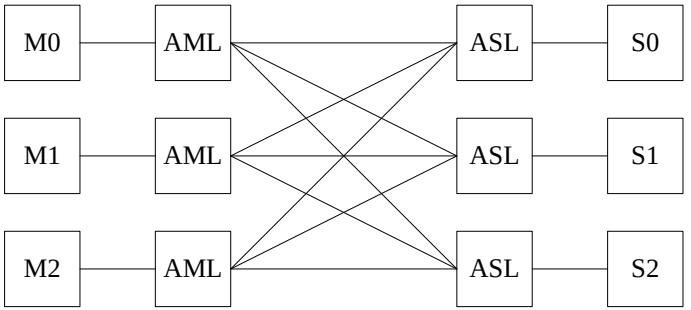
AXI关键信号

- **ARCACHE/AWCACHE: 缓存控制**
  - **[0]: 可写缓冲，未到达目的即回响应**
  - **[1]: 可缓存，cached/uncached，读预取，写合并**
  - **[2]: 读分配，如果读发生缓存缺失则分配一个缓存块**
  - **[3]: 写分配，如果写发生缓存缺失则分配一个缓存块**

AXI互连

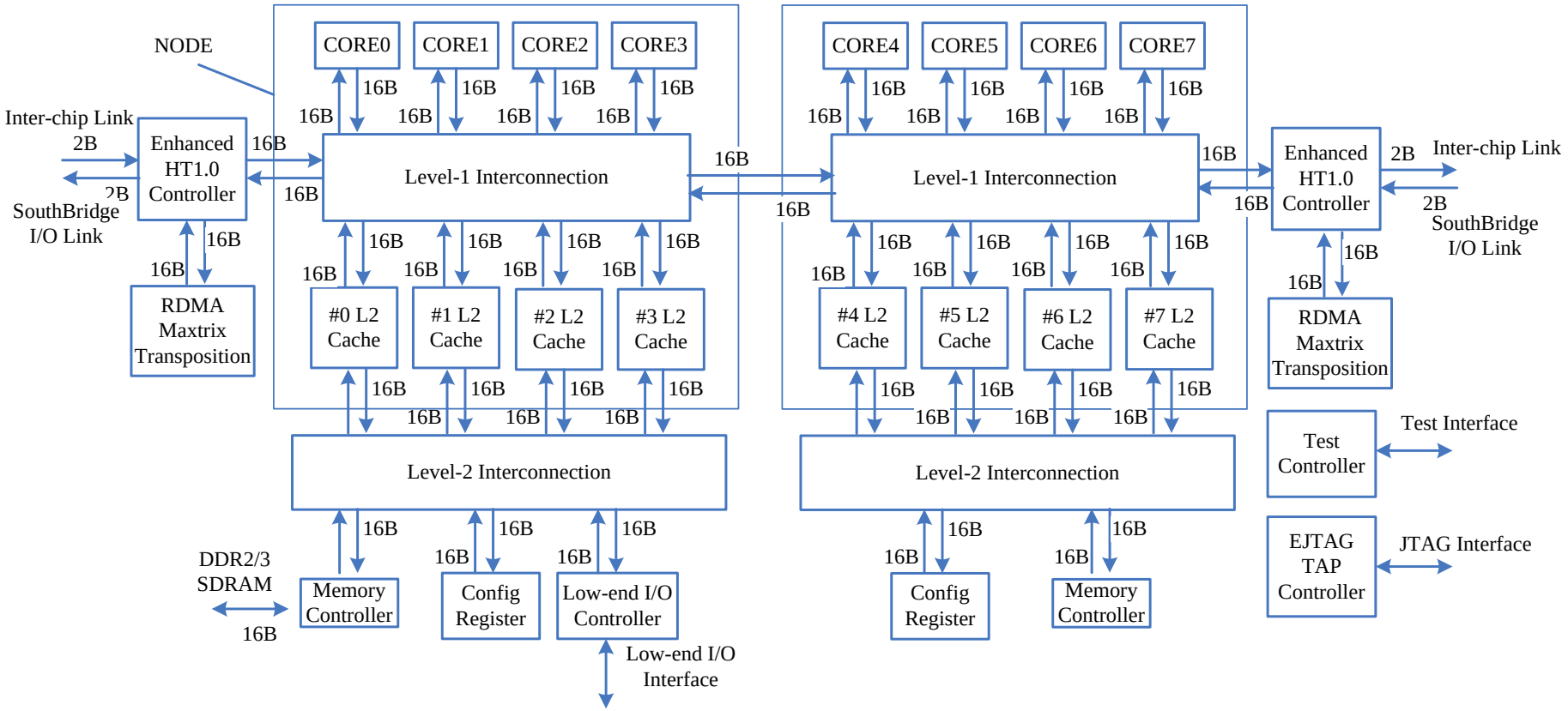
- 将多个主设备、从设备相连
  - **AML:** 单主接多从
    - 根据地址路由
  - **ASL:** 多主接单从
    - **ID**域扩展，加上主端口号
    - 返回时选择路由
  - 共享式
    - 互连线少
  - 交换式
    - 带宽高
  - *怎么加流水？*





AXI系统

- 龙芯3B
- 扩展了Cache一致性支持



AXI实例

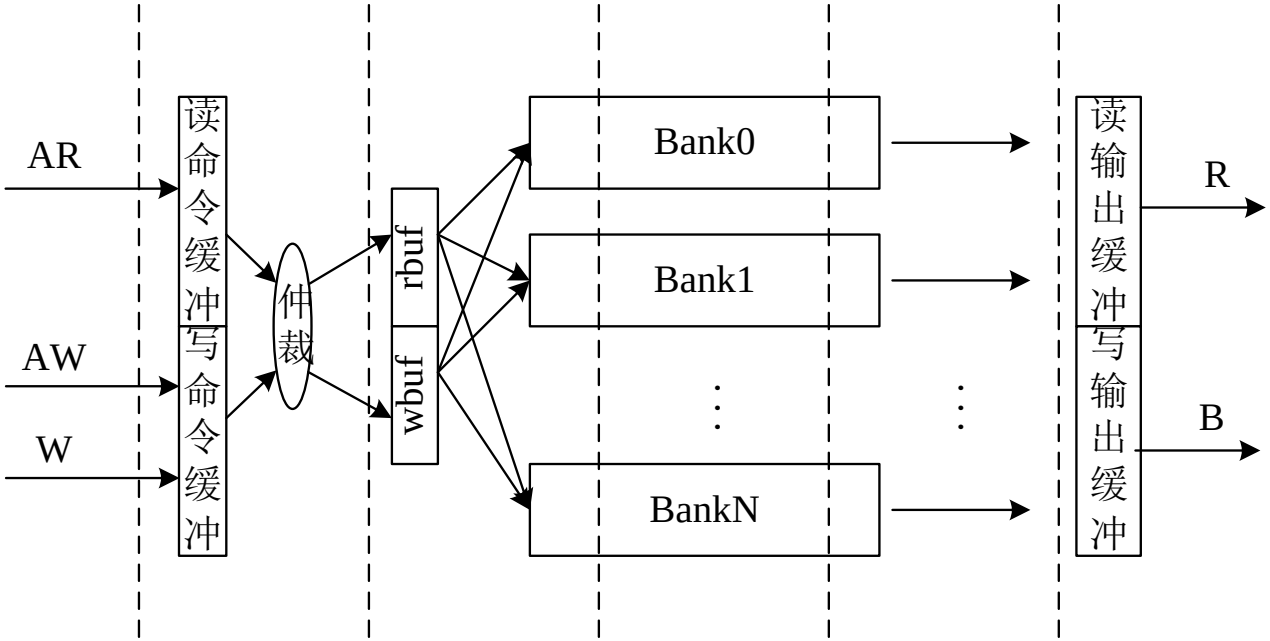
- 设计一个AXI接口的RAM
  - 容量64KByte
  - 数据宽度128bit
  - 不考虑prot/cache/lock
- 要求
  - 充分发挥AXI总线的优势
  - 达到200%带宽利用率（读100%+写100%）

```
module axi_ram (
    input               aclk,
    input               aresetn,

    input [3:0]         s_awlen,
    input [7:0]         s_awid,
    input [31:0]        s_awaddr,
    input [2:0]         s_awsize,
    input [1:0]         s_awburst,
    input               s_awvalid,
    output              s_awready,
    input [7:0]         s_wid,
    input [127:0]       s_wdata,
    input [15:0]        s_wstrb,
    input               s_wlast,
    input               s_wvalid,
    output              s_wready,
    output [7:0]        s_bid,
    output [1:0]        s_bresp,
    output              s_bvalid,
    input               s_bready,
    input [3:0]         s_arlen,
    input [7:0]         s_arid,
    input [31:0]        s_araddr,
    input [2:0]         s_arsize,
    input               s_arvalid,
    output              s_arready,
    output [7:0]        s_rid,
    output [127:0]      s_rdata,
    output [1:0]        s_rresp,
    output              s_rlast,
    output              s_rvalid,
    input               s_rready
);
```

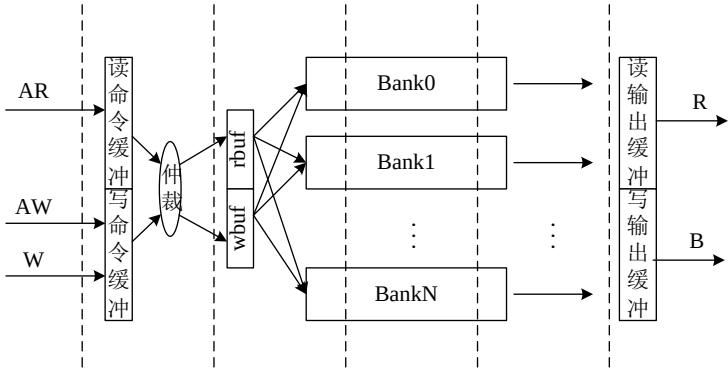
AXI RAM设计

- 思路
  - 充分流水，避免时序瓶颈
  - 多体实现，增加读写并行度



AXI RAM设计

- 读写命令缓冲及拆分
  - 存储AXI总线的访问请求，并将Burst长度大于1的访问拆分为多周期访问，往后级转发。读和写访问的拆分同时进行，如果不发生冲突，则读写可一起前进。
- 读写控制分发
  - 进行地址译码，产生对应RAM块的访问信号。
- **RAM访问**
  - **RAM访问需要1周期。**
- **RAM读出数据缓冲**
  - 传递RAM的输出结果，以及读写的ID信息
- 输出缓冲
  - 产生R*和B*两组输出。输出缓冲没有空项作为流水线停顿的控制信号。
  - 冲突检测只在第一级流水完成，为避免由于读写不同步而导致新的冲突，读写流水之间采用相同的控制信号。即如果写回应被堵住，读数据返回也将停止，反之亦然。



AXI RAM设计

- 读缓冲设计

```
wire                rd_accept; // accept ar request
wire                rd_issue;  // read addr gen
always @(posedge aclk) begin
   if (areset) begin
      rbuf_busy <= 1'b0;
      rbuf_len  <= 4'b0;
   end else begin
      if (rd_accept) begin
         rbuf_busy <= 1'b1;
         rbuf_addr <= s_araddr;
         rbuf_len  <= s_arlen;
         rbuf_size <= s_arsize;
         rbuf_id   <= s_arid;
         rbuf_wrap <= (s_arburst == 2'b10) && (s_araddr[4:0]==5'b10000) & (s_arlen==4'b1);
      end else if (rd_issue) begin
         rbuf_busy <= (rbuf_len != 4'b0);
         rbuf_len  <= rbuf_len - 4'b1;
         rbuf_addr <= rbuf_wrap ? rbuf_addr - 5'h10 : rbuf_addr + 5'h10;
      end
   end // else: !if(areset)
end
```

AXI RAM设计

- 写缓冲设计

```
wire                wr_accept; // accept aw request
wire                w_accept;  // accept w request
wire                wr_issue;  // write addr gen
always @(posedge aclk) begin
   if (areset) begin
      wbuf_busy   <= 1'b0;
      wbuf_len    <= 4'b0;
      wdata_valid <= 1'b0;
   end else begin
      if (wr_accept) begin
         wbuf_busy <= 1'b1;
         wbuf_addr <= s_awaddr;
         wbuf_len  <= s_awlen;
         wbuf_size <= s_awsize;
         wbuf_id   <= s_awid;
         wbuf_wrap <= (s_awburst == 2'b10) && (s_awaddr[4:0]==5'b10000) & (s_awlen==4'b1);
      end else if (wr_issue) begin
         wbuf_busy <= (wbuf_len != 4'b0);
         wbuf_len  <= wbuf_len - 4'b1;
         wbuf_addr <= wbuf_wrap ? wbuf_addr - 5'h10 : wbuf_addr + 5'h10;
      end

      wdata_valid <= w_accept | wdata_valid & ~wr_issue;
      if (w_accept) begin
         wdata_value <= s_wdata; // forget about IDs...
         wdata_strb  <= s_wstrb;
      end
   end // else: !if(areset)
end
```

如果写请求的序与写数据的序不一致会怎样?

AXI RAM设计

- 控制逻辑

```
wire                rpipe_run; // read pipeline run
wire                wpipe_run; // write pipeline run

assign rbuf_last = (rbuf_len == 4'b0);
assign wbuf_last = (wbuf_len == 4'b0);

assign rd_accept =            s_arvalid & (~rbuf_busy | rbuf_last & rd_issue);
assign wr_accept = s_wvalid & s_awvalid & (~wbuf_busy | wbuf_last & wr_issue);

assign s_arready =            (~rbuf_busy | rbuf_last & rd_issue);
assign s_awready = s_wvalid & (~wbuf_busy | wbuf_last & wr_issue);

assign w_accept  = s_wvalid & ( wbuf_busy & ~wbuf_last & wr_issue |
                                wbuf_busy & ~wdata_valid          |
                               (wbuf_busy &  wbuf_last & wr_issue | ~wbuf_busy) & s_awvalid);
assign s_wready  = w_accept;

assign rd_issue = rbuf_busy & (~bank_conflict | cflt_read ) & rpipe_run ;
assign wr_issue = wbuf_busy & (~bank_conflict | cflt_write) & wpipe_run & wdata_valid;

assign rpipe_run = ~obuf_rd_valid[1] & ~obuf_wr_valid[1]; // interlocked pipelines -___-||
assign wpipe_run = ~obuf_rd_valid[1] & ~obuf_wr_valid[1];
```

AXI RAM设计

- **RAM接口**

```
// address mapping
wire [4:0] rbank_id = rbuf_addr[15:12];
wire [4:0] wbank_id = wbuf_addr[15:12];
wire [11:0] roffset = rbuf_addr[11:0];
wire [11:0] woffset = wbuf_addr[11:0];

// deal with conflict
reg  wr_prio;

wire bank_conflict = rbuf_busy & wbuf_busy & (rbank_id == wbank_id);
wire cflt_read     = bank_conflict & ~wr_prio;
wire cflt_write    = bank_conflict &  wr_prio;
always @(posedge aclk) wr_prio <= areset | bank_conflict ^ wr_prio;

wire [31:0] rb_sel = (32'h1 << rbank_id) & {32{rd_issue}};
wire [31:0] wr_sel = (32'h1 << wbank_id) & {32{wr_issue}};
```

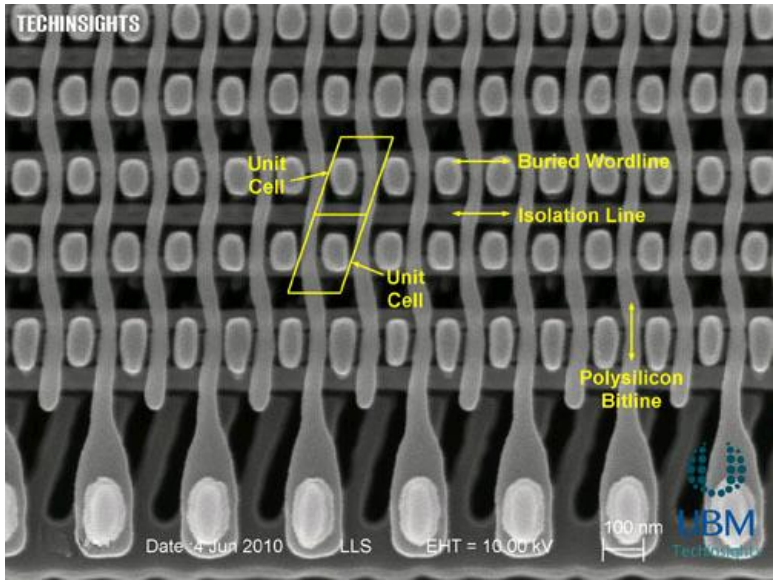
```
wire             cen = ~(run&(r_sel|w_sel));   // active low
wire [15:0]      wen = ~({16{w_sel}} & wstrb);
wire [7:0]       addr = {8{r_sel}} & raddr[11:4] | {8{w_sel}} & waddr[11:4];
wire [127:0]     Q;
reg [127:0]      dat;
reg              read_d;
always @(posedge clk) begin
   if (rst) begin
      read_d <= 1'b0;
      dat    <= 128'b0;
   end else if (run) begin
      read_d <= ~cen & r_sel;
      dat    <= read_d ? Q : 128'b0; // dat
   end
end
```

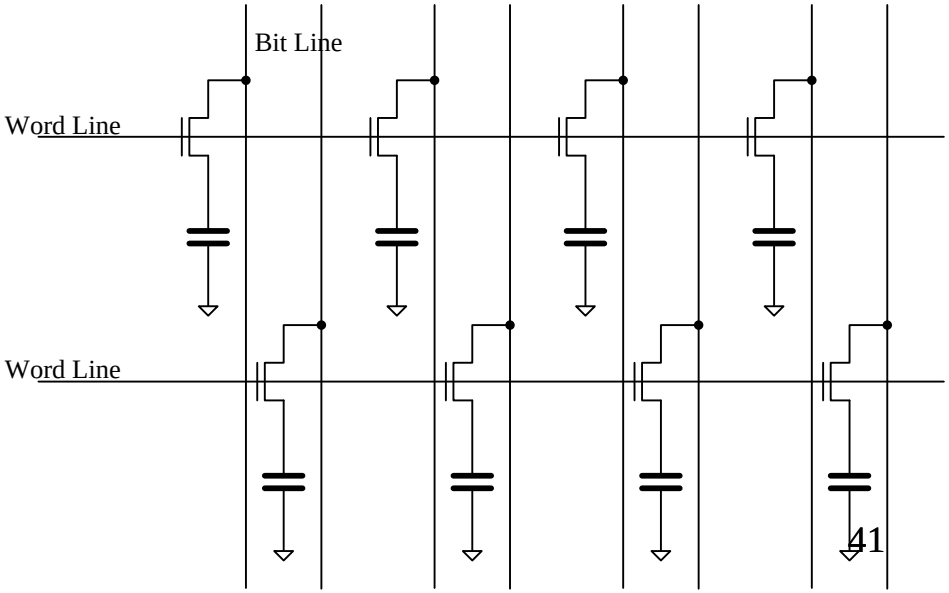
内存总线

DRAM内存结构

DRAM内存结构

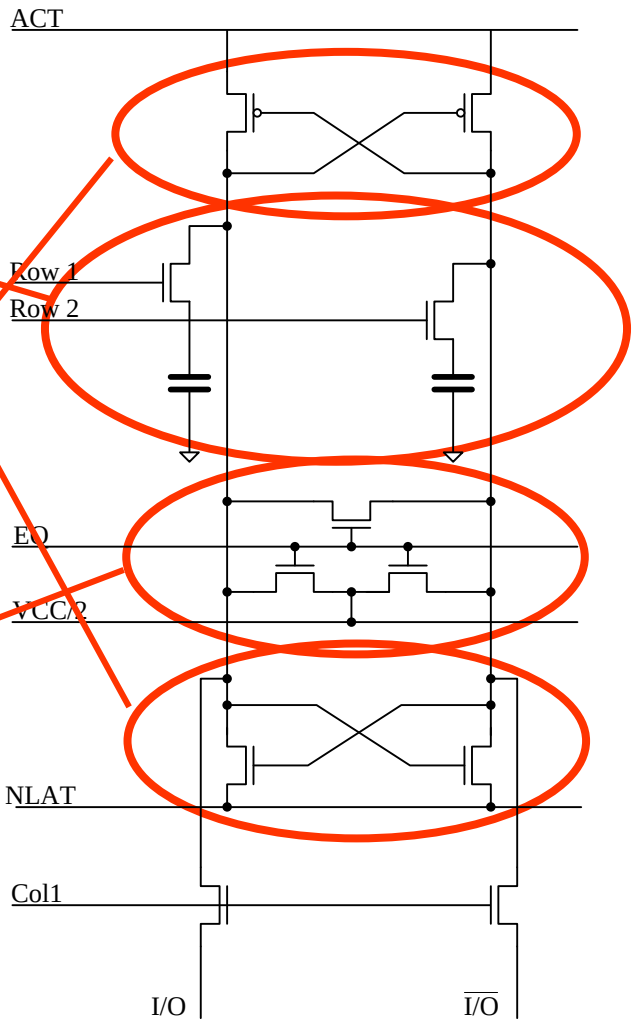
- **DRAM**
  - ***Dynamic* Random Access Memory**
  - **Dr. Dennard发明于1966年，使用1T1C结构存储数据**
    - **当时磁芯存储器是主流，MOS工艺尚不成熟**
  - **简单的结构使DRAM成为计算机的主流存储器**
  - **历经数十年发展，其基本存储原理仍保持不变**





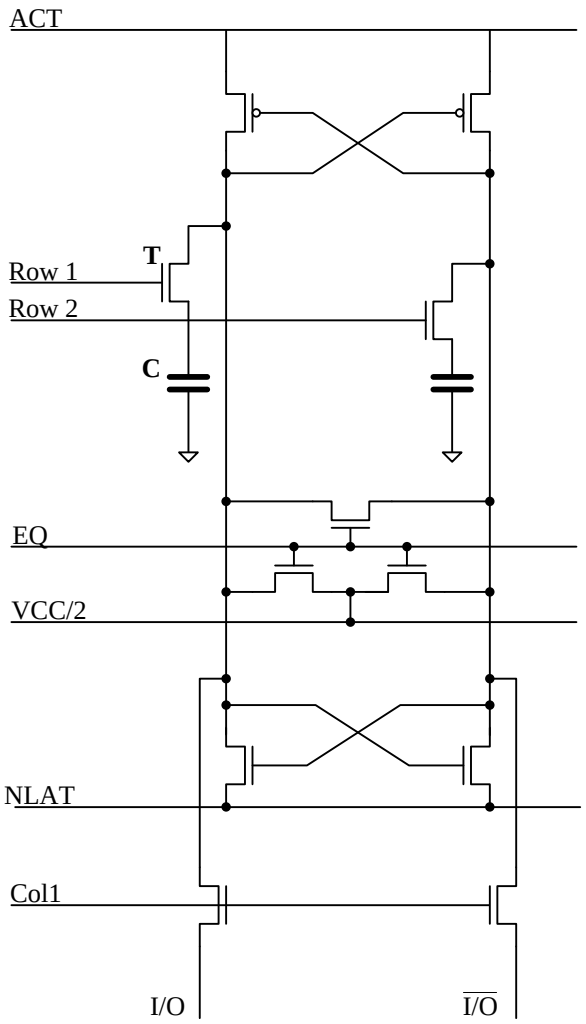
DRAM内存结构

- 存储单元
  - **1T1C: 1个NMOS，1个电容**
  - 电容存储的电位决定存储单元的逻辑值
- 感应放大器
  - 一个存储单元的电容比位线小得多，在读出时只能引起位线电压的微小变化，因而需要进行放大
- 均衡电路
  - 用于读出准备，将字线拉到中间电平



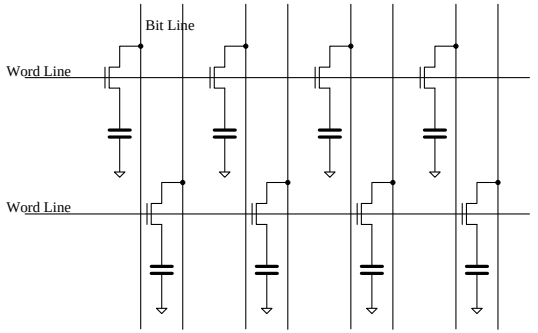
DRAM内存结构

- 读出过程
  - 先把位线预充到**Vref=VCC/2**
  - 字线打开**T**管，**C**引起位线微小的电位差
  - 感应放大器比较、放大，得到**C**所存的值
  - 感应放大器互锁结构保持读出的值不变
  - 读出的值通过位线回送给**C**（原值在接上位线时被破坏）
- 写入过程
  - 写入电路强驱字线，重置感应放大器的值、位线的值以及**C**的值

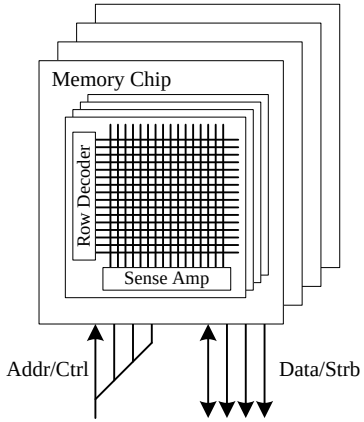


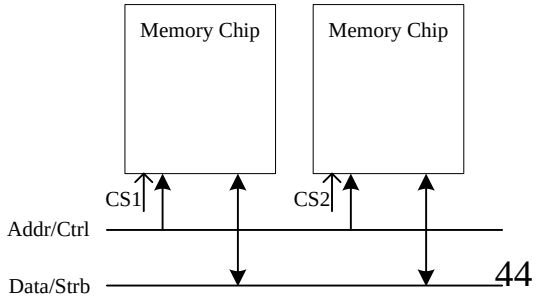
内存组织

- **Row、Column**
  - 对应二维存储阵列
  - 用行地址、列地址寻址

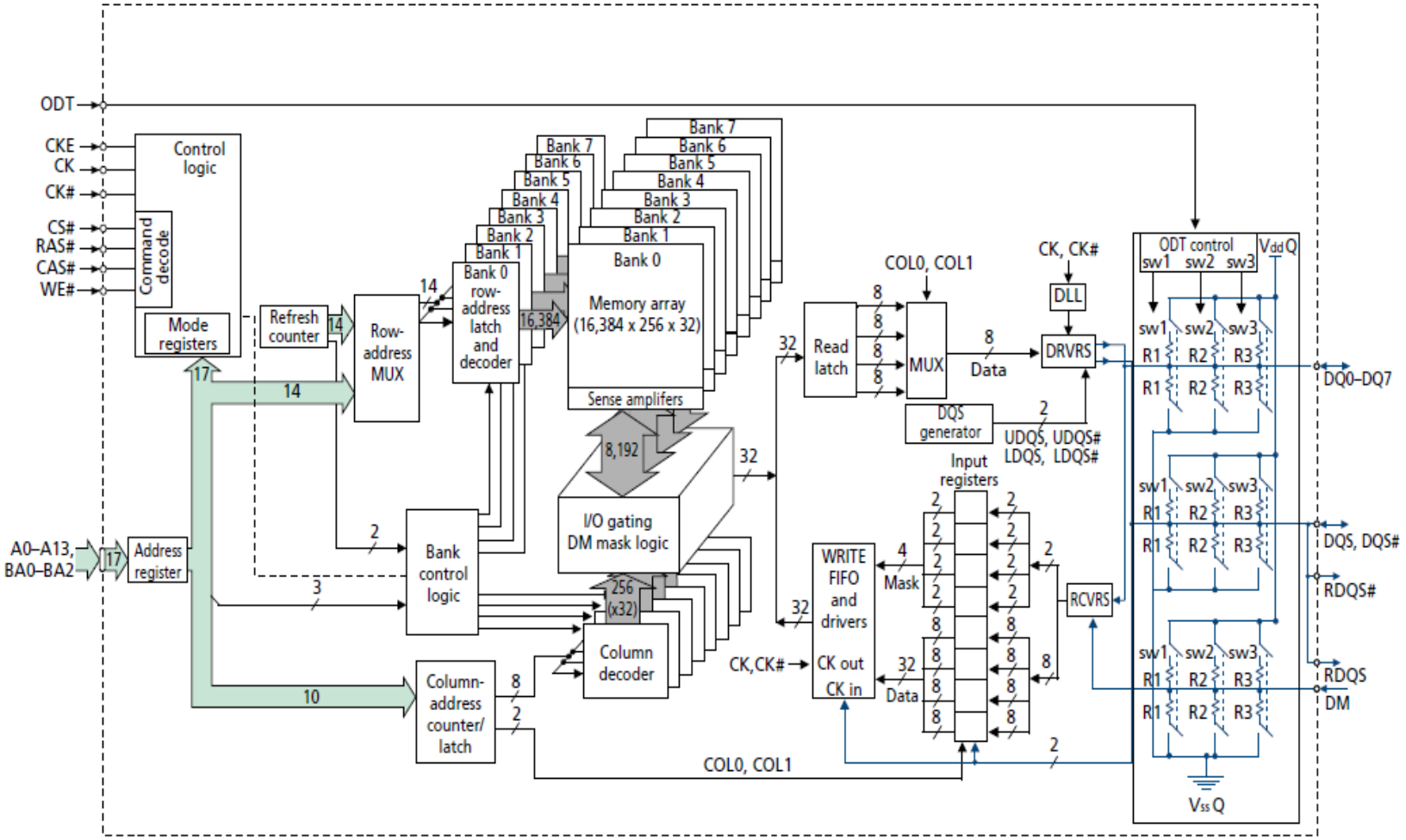


- **Bank**
  - 存储阵列及其感应放大器
  - 一个存储芯片内有多个**Bank**
  - 多个芯片并联扩展数据位宽
- **Rank**
  - 多个存储芯片并联，扩展存储容量
  - 通过片选信号区分



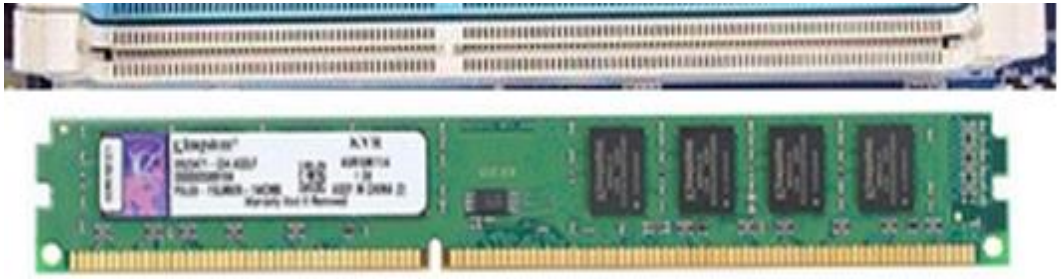


内存内部结构


ODT
CKE
CK
CK#
CS#
RAS#
CAS#
WE#
Control logic
Command decode
Mode registers
Refresh counter
14
Row-address MUX
Bank 0 row-address latch and decoder
16,384
Bank 7
Bank 6
Bank 5
Bank 4
Bank 3
Bank 2
Bank 1
Bank 0
Memory array (16,384 x 256 x 32)
Sense amplifers
8,192
I/O gating DM mask logic
256 (x32)
Column decoder
Bank control logic
Column-address counter/ latch
A0–A13, BA0–BA2
17
Address register
3
10
2
8
32
Read latch
COL0, COL1
MUX
Data
DQS generator
CK, CK#
DLL
DRVRS
UDQS, UDQS#
LDQS, LDQS#
Input registers
WRITE FIFO and drivers
CK out
CK in
Mask
RCVRS
ODT control
sw1 sw2 sw3
VddQ
R1 R2 R3
DQ0–DQ7
DQS, DQS#
RDQS#
RDQS
DM
VssQ

内存总线——机械层

- **JEDEC（Joint Electron Device Engineering Council）**
- **DDR SDRAM**

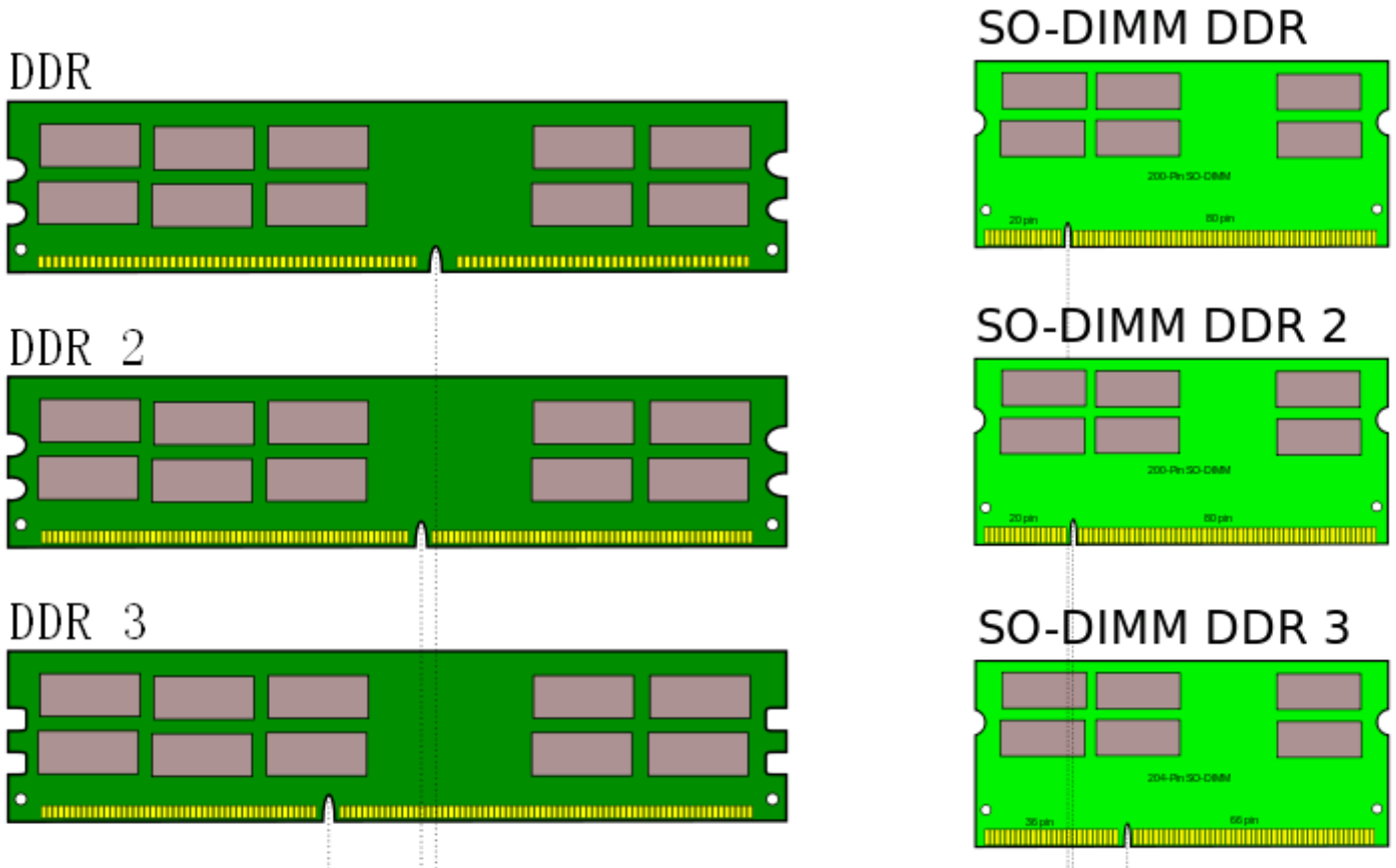


台式机电脑的**DDR3**内存条和内存插槽

台式机电脑的**DDR2**内存条

内存总线——机械层

- **Dual Inline Memory Module(DIMM)**
- **Small Outline DIMM**



内存总线信号分类

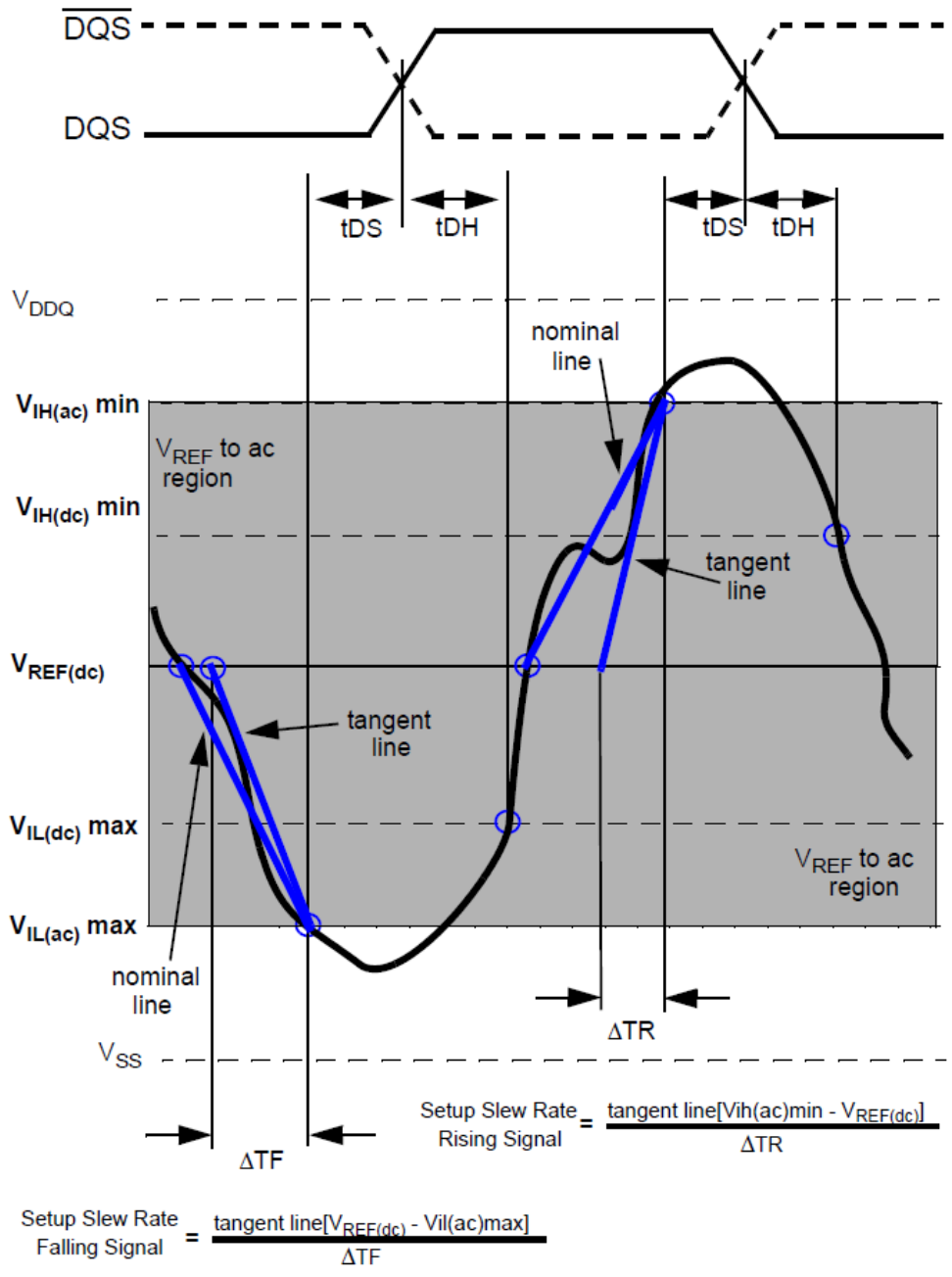
- 时钟信号
  - **CKp/CKn**
  - 用作内存系统工作的参考时钟
  - 用作地址命令信号的采样时钟
- 地址命令信号
  - **BA、ADDR、RAS#、CAS#、WE#、CKE、CS、ODT**
  - 时钟上升沿有效
  - 要求与时钟沿错开以正确采样
- 数据及数据采样信号
  - **DQSp/DQSn、DQ、DM**
  - 以**CKp/CKn**为参考时钟，与其保持一定的相位关系，但并不严格
  - **DQ、DM**的采样以**DQSp/DQSn**的双沿有效

内存总线——内存条接口信号

引脚名称	描述	引脚名称	描述
A[15:0]	SDRAM地址线	SCL	EEPROM I2C总线时钟
BA[2:0]	SDRAM bank地址	SDA	EEPROM I2C总线数据线
RAS#	SDRAM 行地址选通	SA[2:0]	EEPROM I2C从设备地址
CAS#	SDRAM 列地址选通	VDD	SDRAM core电源
WE#	SDRAM 写使能	VDDQ	SDRAM IO输出电源
S[1:0]#	SDRAM片选信号	VrefDQ	SDRAM IO参考电源
CKE[1:0]	SDRAM时钟使能信号	VrefCA	SDRAM命令地址参考电源
ODT[1:0]	SDRAM 终端匹配电阻控制信号	Vss	电源地信号
DQ[63:0]	DIMM内存数据线	VDDSPD	EEPROM电源
CB[7:0]	DIMM ECC数据线	TEST	测试引脚
DQSp/DQSn[8:0]	SDRAM数据时钟	RESET#	复位引脚
DM[8:0]	SDRAM数据掩码	EVENT#	温度传感器引脚（可选）
CKp/CKn[1:0]	SDRAM时钟信号线	VTT	SDRAM IO终端匹配电阻电源

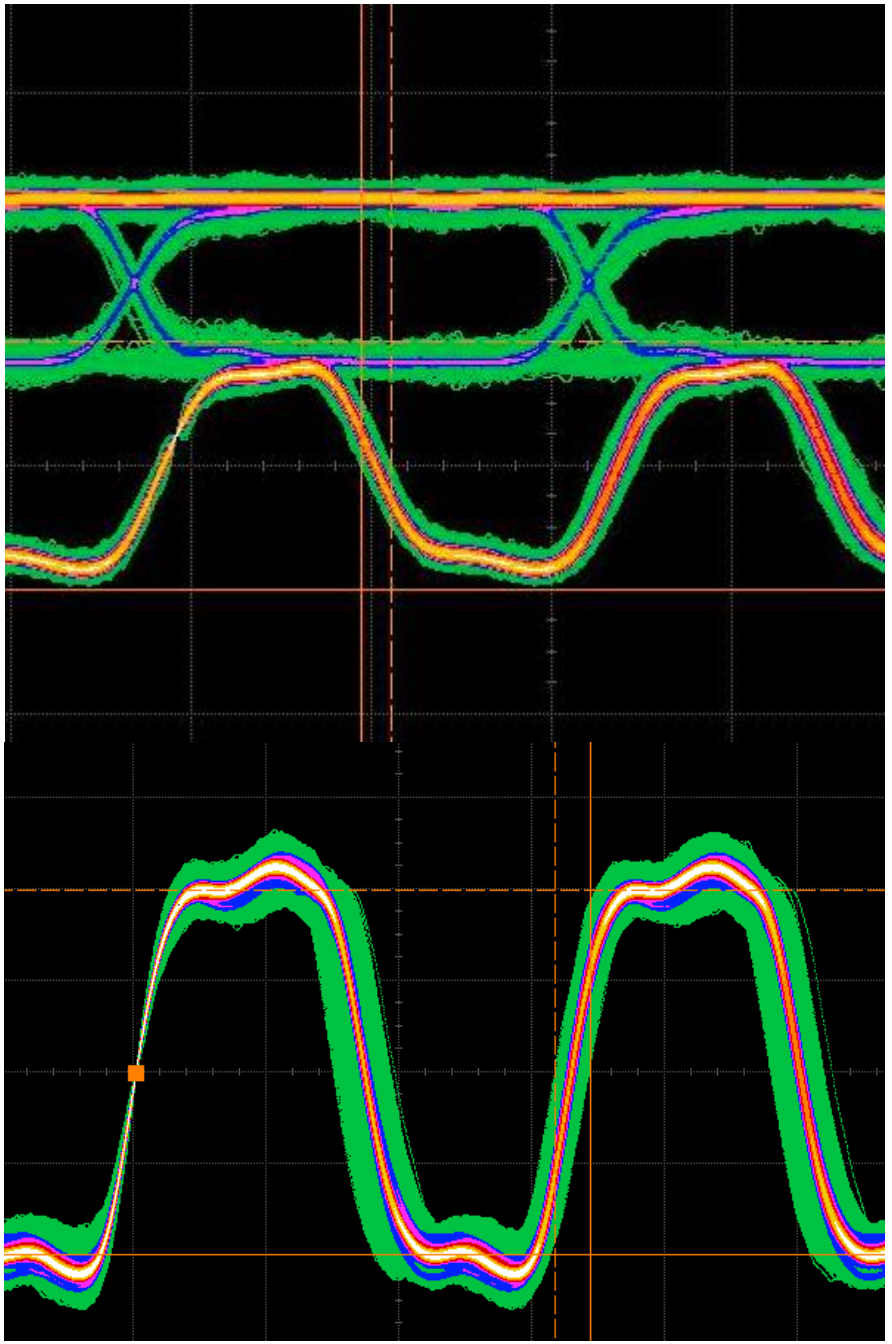
内存总线——电气层

- 内存电压
  - **SDRAM：3.3V**
  - **DDR内存：2.5V**
  - **DDR2内存：1.8V**
  - **DDR3内存：1.5V**
  - **DDR4内存：1.2V**
- 输入输出电压高低电平标准
- 信号斜率
- 时钟抖动范围



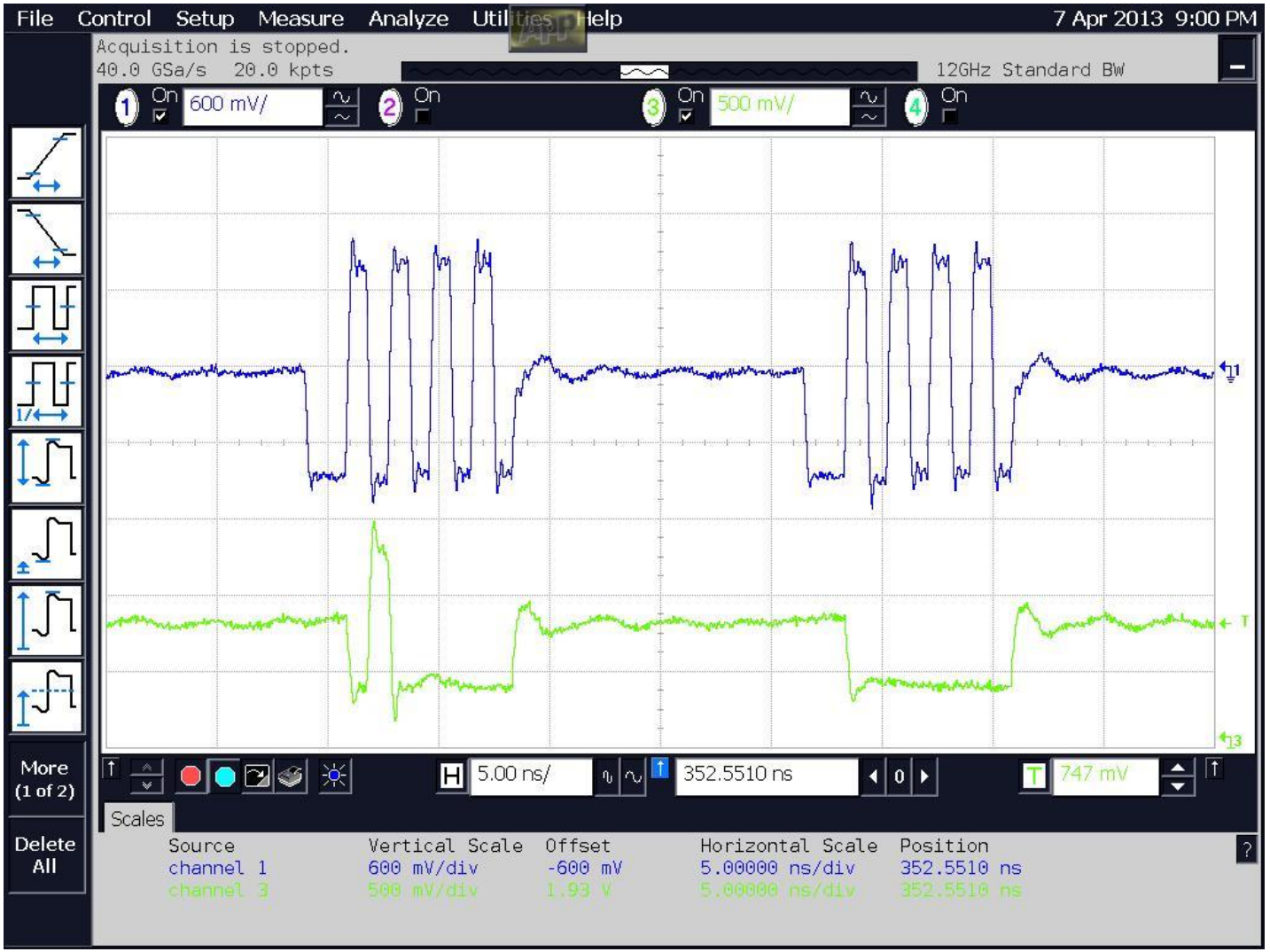
$$\text{Setup Slew Rate Rising Signal} = \frac{\text{tangent line}[V_{ih(ac)min} - V_{REF(dc)}]}{\Delta TR}$$

$$\text{Setup Slew Rate Falling Signal} = \frac{\text{tangent line}[V_{REF(dc)} - V_{il(ac)max}]}{\Delta TF}$$



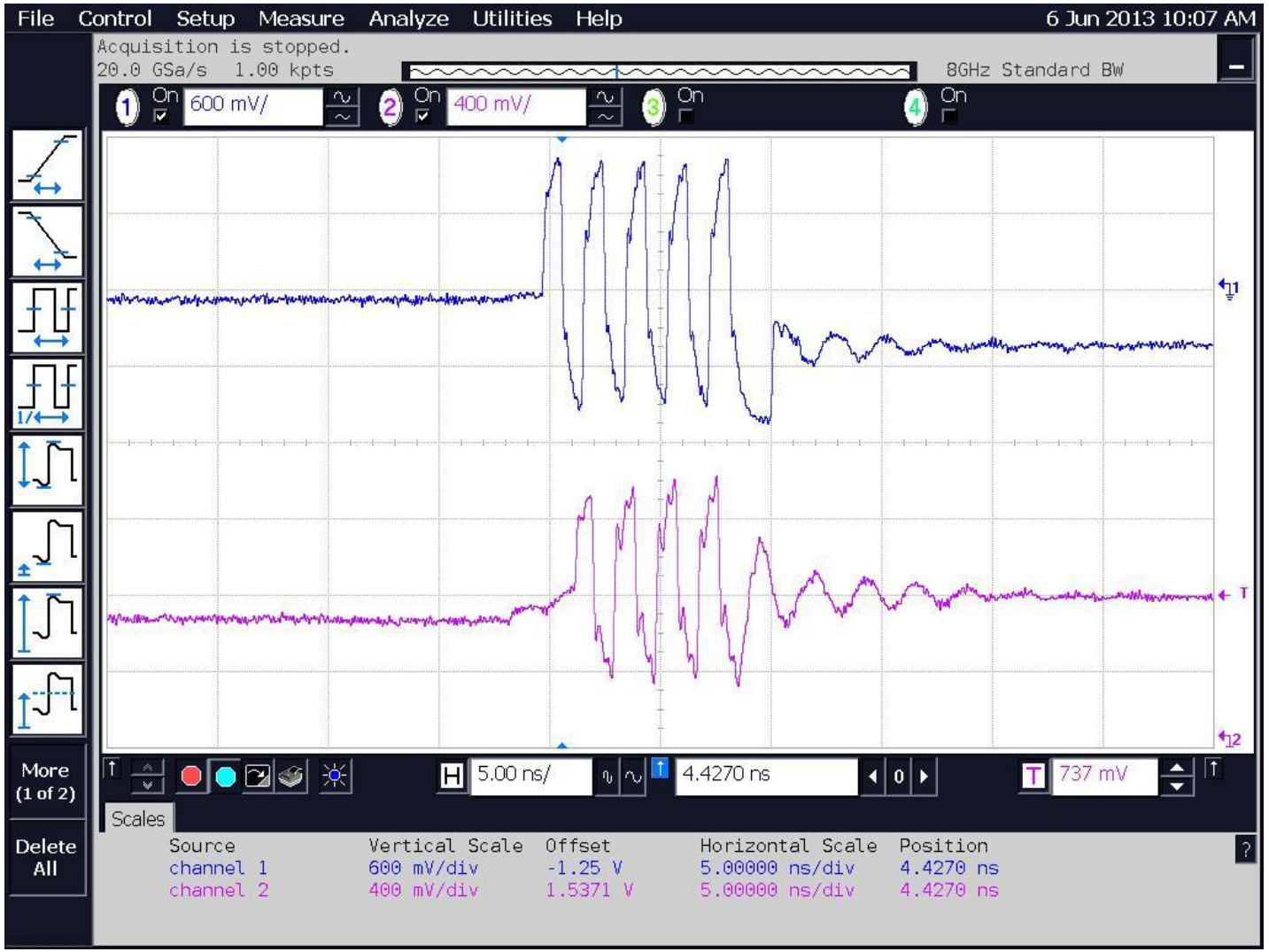
内存总线——电气层

- 读操作DQS与DQ



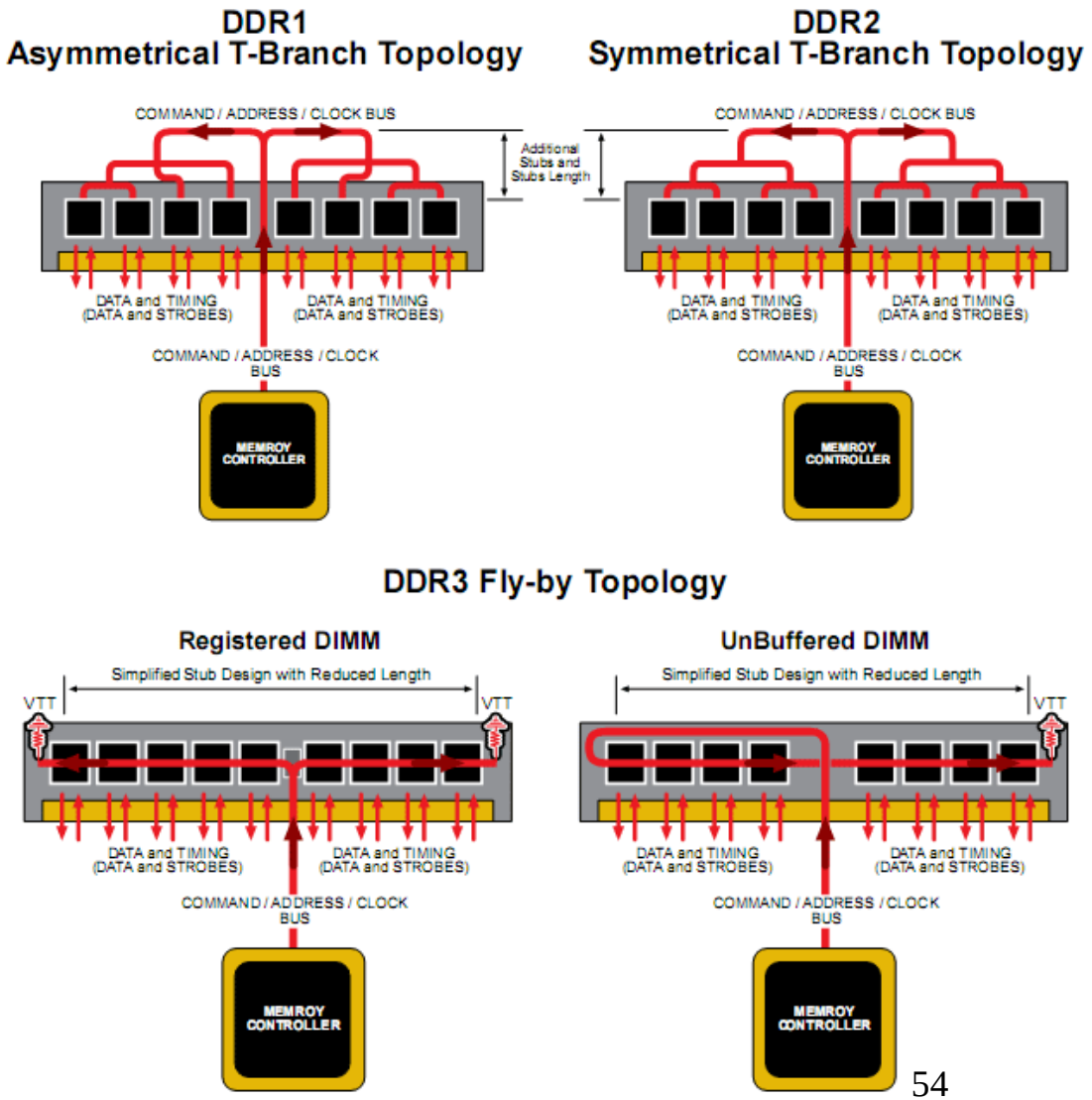
内存总线——电气层

- 写操作DQS与DQ



内存总线拓扑

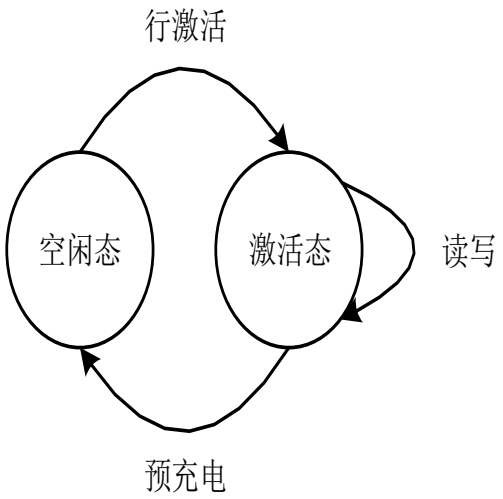
- 地址控制线驱动众多负载
  - 需保证信号完整性
- 实现难度随着速率提高而不断增大
- 连接方式和支持机制不断发展



▲ Simplified Comparison between DDR1, DDR2 and DDR3 Topologies
Illustration: Ryan J. Leng

内存总线——协议层

- 上电时序
- 状态转换
- 时序timing：
  - tAA，tRCD，tRP等
- 低功耗控制：
  - powerdown，自刷新

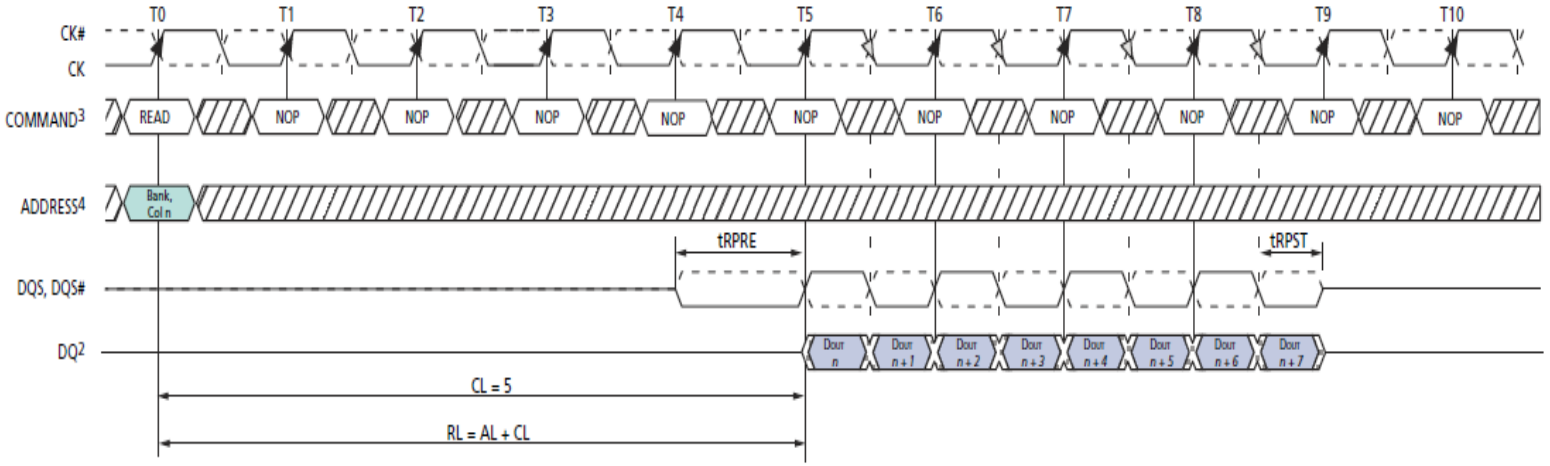


内存命令

	CKE	RAS	CAS	WE	A10
模式配置	1	0	0	0	
刷新	1	0	0	1	
进入自刷新	0	0	0	0	
退出自刷新	1	1	1	1	
单Bank预充	1	0	1	0	0
预充所有	1	0	1	0	1
激活	1	0	1	1	
写	1	1	0	0	0
写并预充	1	1	0	0	1
读	1	1	0	1	0
读并预充	1	1	0	1	1
进入关断	0	1	1	1	
退出关断	1	1	1	1	

内存总线——协议层

- 读操作

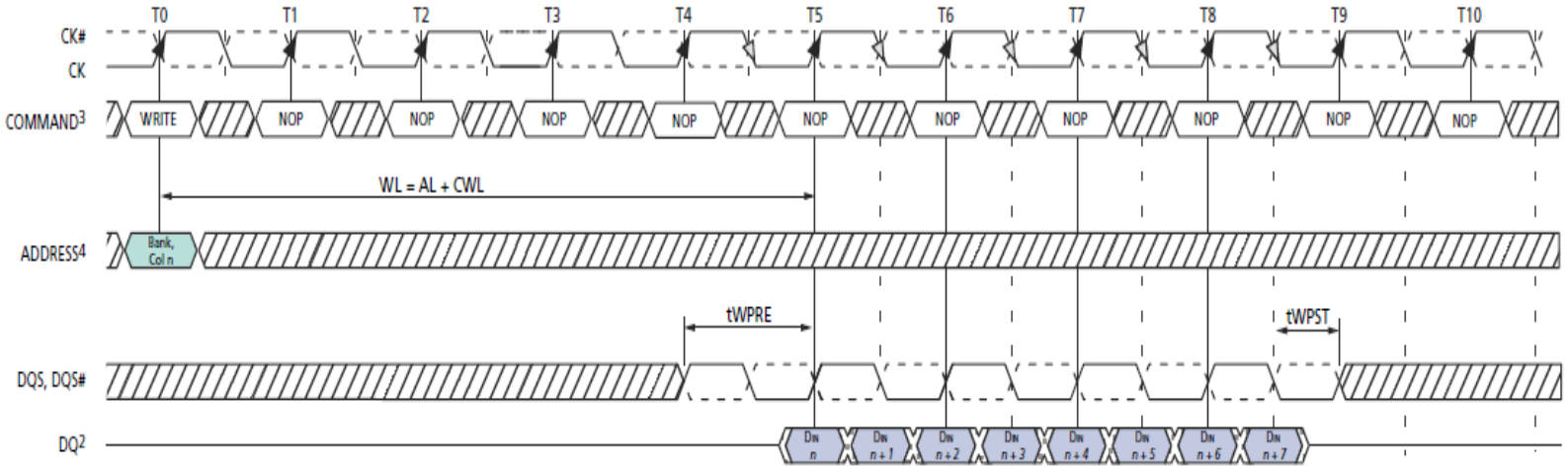


DDR3 SDRAM读操作时序

在当前行已被激活的情况下，读命令在T0时刻发出，经过5个周期，在T5时刻内存芯片开始通过DQ信号返回读数据。内存芯片在连续4个周期内返回数据，由于DDR内存使用双沿采样，因此一共返回8位数据。内存在返回数据的同时也驱动DQS信号，DQS信号和DQ信号是完全同步的，内存控制器使用该信号去采样DQ从而捕获DQ的值。DQS信号有一个preamble，供内存控制器过滤出有效时钟。

内存总线——协议层

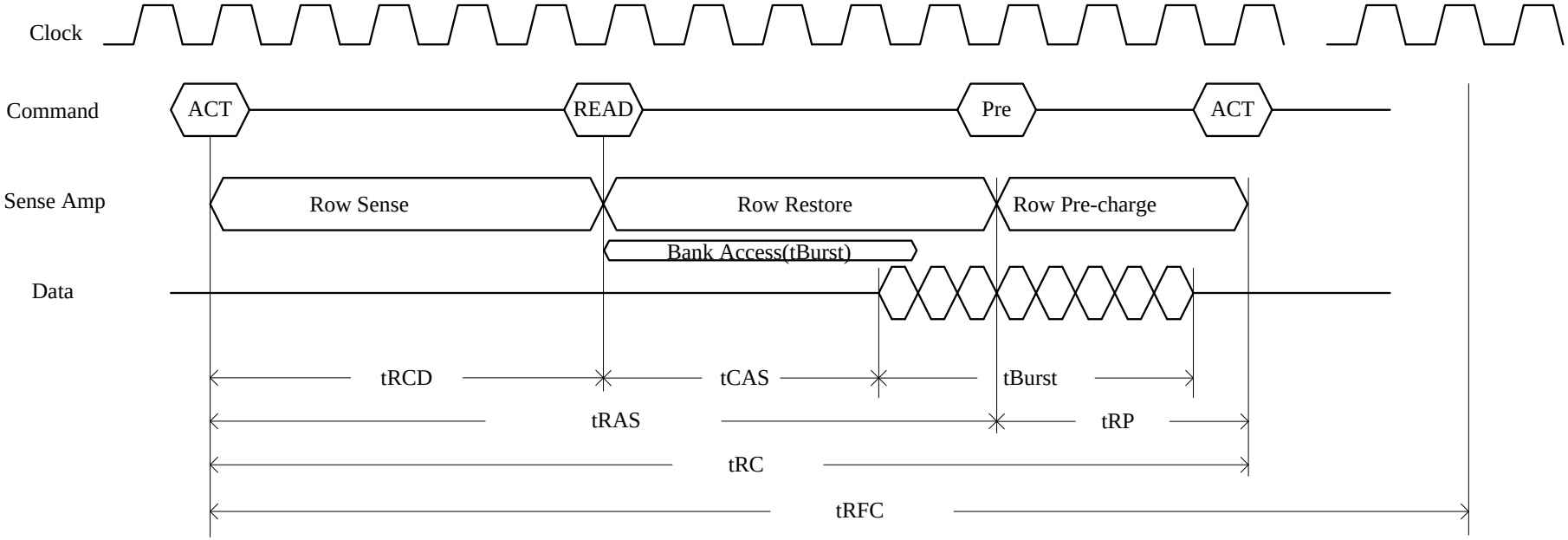
- 写操作



DDR3 SDRAM写操作时序

在当前行已被激活的情况下，写命令在T0时刻发出，经过5个时钟周期，内存控制器开始输出写数据，同时输出DQS。数据以burst方式传输，在BL=8的时候持续4个周期。DQ和DQS的时序是中心对齐的，DQS的上升下降沿对应数据的中心。写DQS同样也有一个preamble。

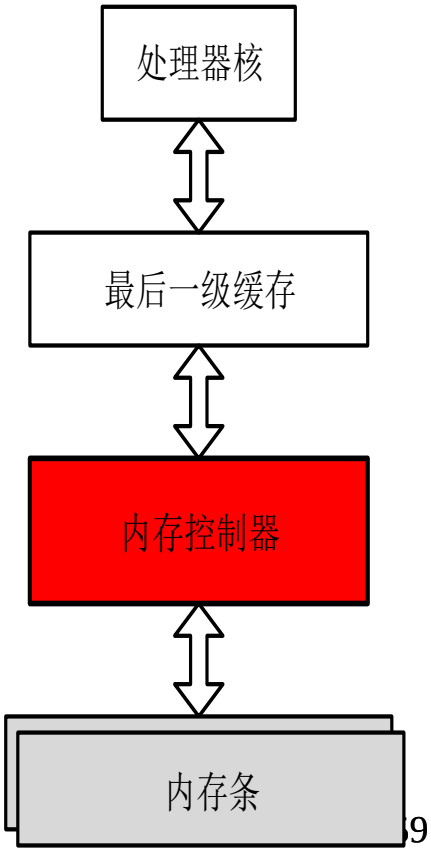
DDR关键时序



tRCD	Row Column Delay	激活行到SA稳定
tCAS(tCL)	Column Access Strobe(Latency)	数据从SA读出到IO
tRP	Row Precharge	预充延迟
tRC	Row Cycle	
tREFI	Refresh Interval	最大刷新间隔
tREF	Refresh	最小刷新间隔
tCWD(tCWL)	Column Write Delay	写命令到数据延迟
tWR	Write Recovery	写数据到存入bit cell延迟
AL	Additive Latency	附加延迟(Posted CAS)
tCCD	CAS to CAS Delay	读写之间最小延迟（内部burst长度）
tRRD	Row to Row Delay	激活之间最小延迟

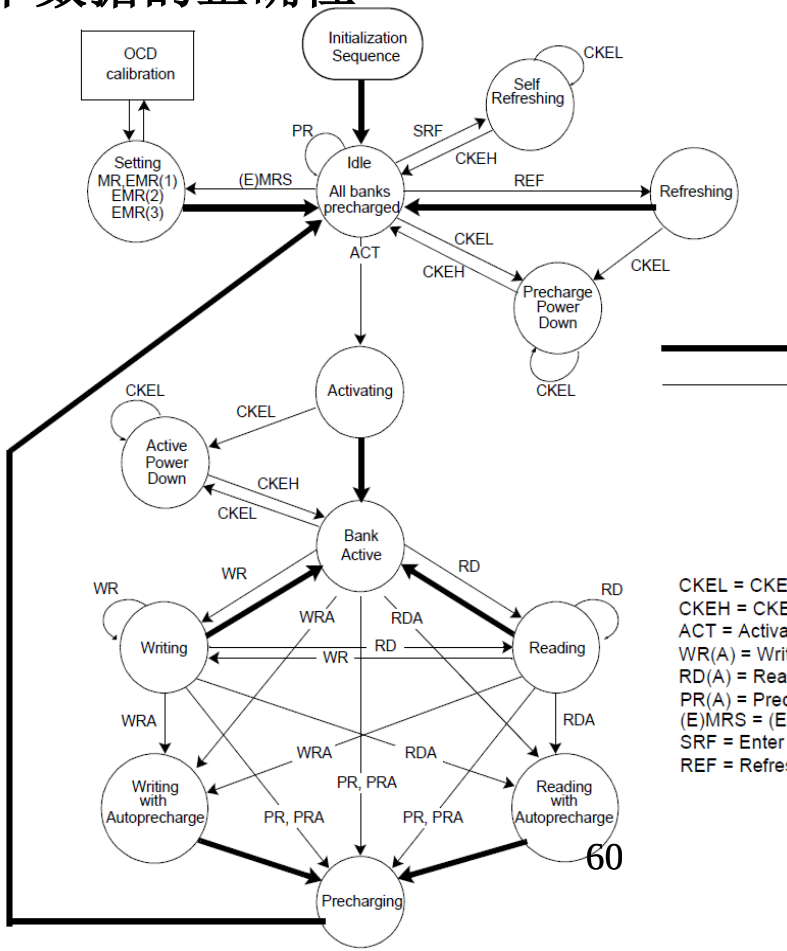
内存总线——内存控制器

- 地址译码
  - 物理地址→SDRAM地址（片选、bank、行、列）
- 命令调度
  - FCFR（First Come First Ready）
  - Bank轮转
- 时序控制
  - 初始化、刷新控制
  - 负责SDRAM的命令发送间隔控制
- 物理接口PHY
  - 单沿双沿转换（并串转换）
  - 信号相位调整（DLL）
  - 数据同步（多个SDRAM）



内存控制器时序控制

- 基于内存操作的状态变换要求
  - 上电复位后的初始化
  - 周期性发出刷新操作以维持内存中数据的正确性
  - 发生读写时根据当前状态操作
    - 激活
    - 读写
    - 预充
  - 保证不同操作间的时序要求
  - 低功耗状态控制
- 实际上更为复杂
  - 真实系统上会有多条内存
  - 每条内存的状态都需要记录维护



内存调度带来的影响-行冲突及命中

CPU 请求	内存地址分解		
	Bank	Row	COL
读 A	0	2	5
读 B	0	3	6
读 C	0	2	7

内存命令分解		
激活	读写	预充
ACT 0,2	READ 0,5	PRE 0
ACT 0,3	READ 0,6	PRE 0
ACT 0,2	READ 0,7	PRE 0

tRCD	5	行打开到读写的间隔
tRASmin	20	最小行打开时间
tRP	10	预充时间
tRL	5	读出时间间隔
tCCD	4	两次列操作时间间隔

周期	0	5	10-13	20	30	35	40-43	50	60	65	70-73	80
命令	ACT 0,2	READ 0,5		PRE 0	ACT 0,3	READ 0,6		PRE 0	ACT 0,2	READ 0,7		PRE 0
数据			DATA A				DATA B				DATA C	
	tRCD	tRL		tRP	tRCD	tRL		tRP	tRCD	tRL		
	tRASmin				tRASmin				tRASmin			

- **B与A行冲突，C与B行冲突，但C与A行命中**
- **不经调度的情况下，需要80拍完成三次读操作**

内存调度带来的影响-行冲突及命中

CPU 请求	内存地址分解		
	Bank	Row	COL
读 A	0	2	5
读 B	0	3	6
读 C	0	2	7

内存命令分解		
激活	读写	预充
ACT 0,2	READ 0,5	PRE 0
ACT 0,3	READ 0,6	PRE 0
ACT 0,2	READ 0,7	PRE 0

tRCD	5	行打开到读写的间隔
tRASmin	20	最小行打开时间
tRP	10	预充时间
tRL	5	读出时间间隔
tCCD	4	两次列操作时间间隔

周期	0	5	9	10-13	14-17	20	30	35	40-43	50
命令	ACT 0,2	READ 0,5	READ 0,7			PRE 0	ACT 0,3	READ 0,6		PRE 0
数据				DATA A	DATA C				DATA B	
	tRCD	tCCD	tRL			tRP	tRCD	tRL		
		tRL								
	tRASmin						tRASmin			

- **在读A时已经激活了Row 2，此时可以直接读C，而不需要预充再激活**

内存调度带来的影响-Bank间并行

CPU 请求	内存地址分解		
	Bank	Row	COL
读 A	0	3	6
读 B	1	4	7
读 C	2	5	8

内存命令分解		
激活	读写	预充
ACT 0, 3	READ 0, 6	PRE 0
ACT 1, 4	READ 1, 7	PRE 1
ACT 2, 5	READ 2, 8	PRE 2

tRCD	5	行打开到读写的间隔
tRASmin	20	最小行打开时间
tRP	10	预充时间
tRL	5	读出时间间隔

周期	0	1	2	5	9	10-12	13	14-17	18-19	20	21	22
命令	ACT 0, 3	ACT 1, 4	ACT 2, 5	READ 0, 6	READ 1, 7	READ 2, 8				PRE 0	PRE 1	PRE 2
数据						DATA A		DATA B	DATA C			
	tRCD			tRL								
		tRCD			tRL							
			tRCD			tRL						
	tRASmin											
		tRASmin										
			tRASmin									

- **A、B、C分别落在不同的bank上，互不影响**
- **只需要保证在数据总线传输时不会相互重叠**

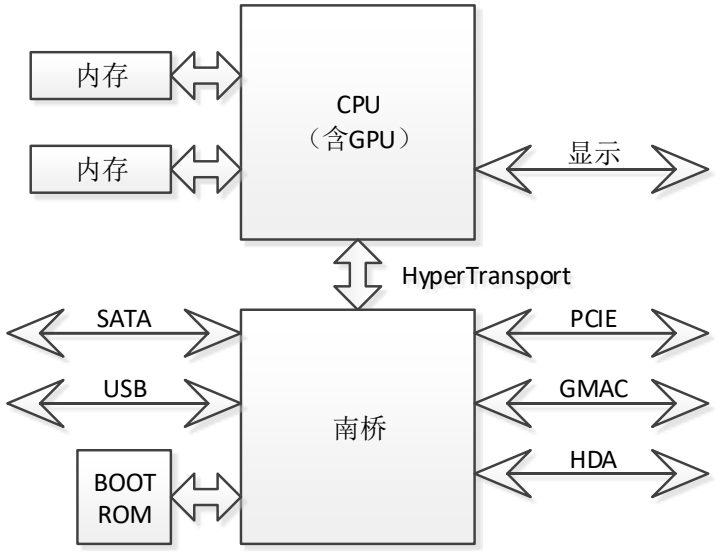
系统总线

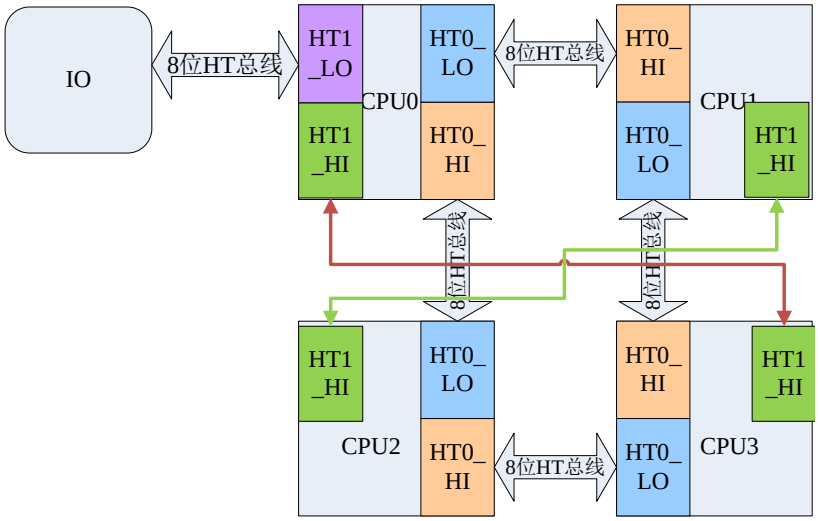
系统总线

- 系统总线通常用于处理器与桥片的连接，同时也作为多处理器间的连接以构成多路系统
- **Intel：QPI、DMI、FSB……**
- **AMD：HT、……**
- 龙芯：**HT**

系统总线所处的位置

- 连接处理器与桥片
  - IO设备与CPU间传输的通道
  - 对IO性能影响大
  - 内存在北桥上时，对系统性能影响更大
- 连接多处理器
  - CC-NUMA多处理器间数据交换的通道
  - 对跨片性能影响大，对NUMA系统的跨片访存产生影响

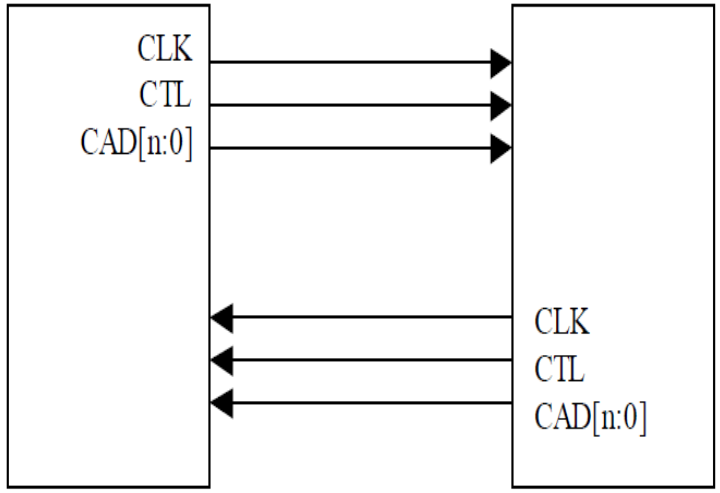




HyperTransport总线

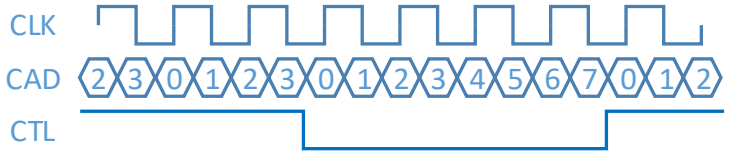
- 串行总线
  - 差分
  - 点对点传输
  - 频率高、信号少
  - 大部分情况下，高速串行总线的频率已经高于芯片内部频率
- 除了串行信号之外，还存在一些“边带”信号
  - 复位、低功耗等

引脚名称	方向	描述
TX_CLKp/TX_CLKn	输出	发送端时钟信号
TX_CTLp/TX_CTLn	输出	发送端控制信号，用于区分命令包与数据包
TX_CADp[n:0]/ TX_CADn[n:0]	输出	发送端命令地址数据复用信号，用于传输各种包
RX_CLKp/RX_CLKn	输入	接收端时钟
RX_CTLp/RX_CTLn	输入	接收端控制信号信号，用于区分命令包与数据包
RX_CADp[n:0]/ RX_CADn[n:0]	输入	接收端命令地址数据复用信号，用于传输各种包



HyperTransport总线传输

- **基于“包”的传输**
  - **命令包**
  - **数据包**
  - **使用CTLp/CTLn信号区分**
  - **此外还有两种不严格的包：**
    - 流控包、校验包
    - 之所以说不严格，因为这两类包本身不参与流控
- **基于“流控”的传输**
  - **片外传输距离长，采用握手机制的传输频率低下**
    - 百兆赫兹已经很困难
  - **采用流控的方式可以完全避免握手**



流控的简要原理（一）

- 流控的硬件支持
  - 总线的接收端需要设计一组或多组接收缓冲，用于临时存储对方发送的包
  - 每个缓冲区需要能容纳一个完整的包
  - 每当缓冲区被释放时，需要通过发送端发出“流控包”通知对方
  - 发送端维护一组或多组计数器，用于记录对方接收端的缓冲包个数
  - 只有发送端的计数器不为零时，才可以发出请求包。且每发一个包，对应的计数器个数减一
  - 收到“流控包”时，对应的计数器个数增加相应数量

流控的简要原理（二）

- 流控的工作流程
  - 初始化时，总线两端发送端的计数器被清零
  - 初始化后，总线两端各自通过发送端将接收端的缓冲区数量“告知”对方，对方将这个数量记录在计数器中
    - 这就是流控包
  - 在需要发包时，首先检查相应的计数器是否为零，仅当不为零时，才可以发包
    - 确保对方可接收
  - 发包之后，将对应的计数器减一
  - 每次缓冲区处理完释放时，通过流控包告知对方

虚通道

- **逻辑上的通道**
  - **AXI总线包括五个不同的物理通道**
    - 每个物理通道对应一个逻辑通道
  - **HT总线分为发送和接收，实际上的物理通道只有两个**
    - 为了避免逻辑上的等待和互锁，采用虚通道
    - HT发送和接收各有三个虚通道
- **在流控机制的基础上，虚通道很好实现**
  - **为不同的虚通道设置各自的缓冲区**
  - **发送时各个虚通道间不能有相互堵塞的情况**

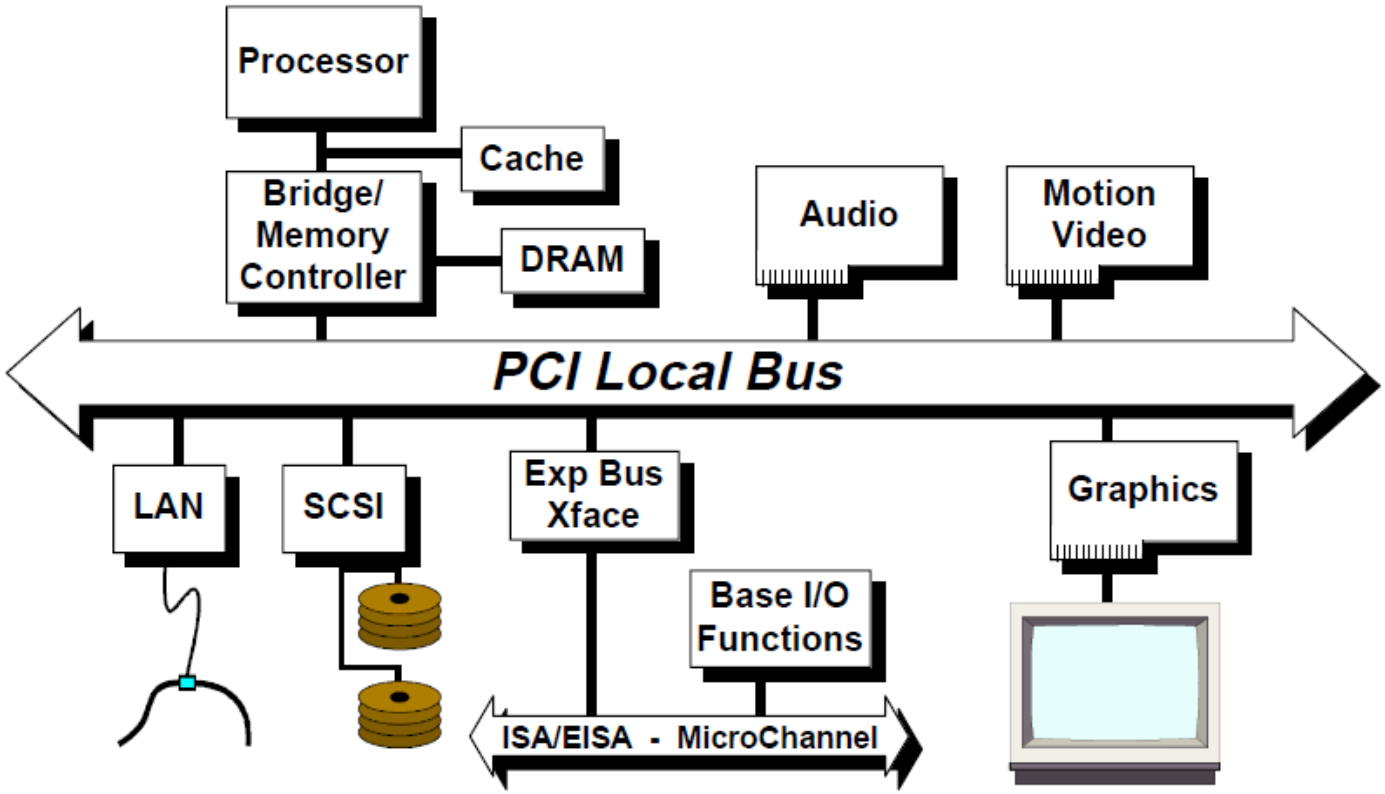
IO总线

IO总线

- **连接处理器与输入输出设备的总线，也称为设备总线**
- **局部总线：ISA、PCI、 AGP、PCIe、……**
- **显示总线：VGA、DVI、HDMI、DP、……**
- **音频总线：AC97、HDA**
- **其他：SATA、USB**

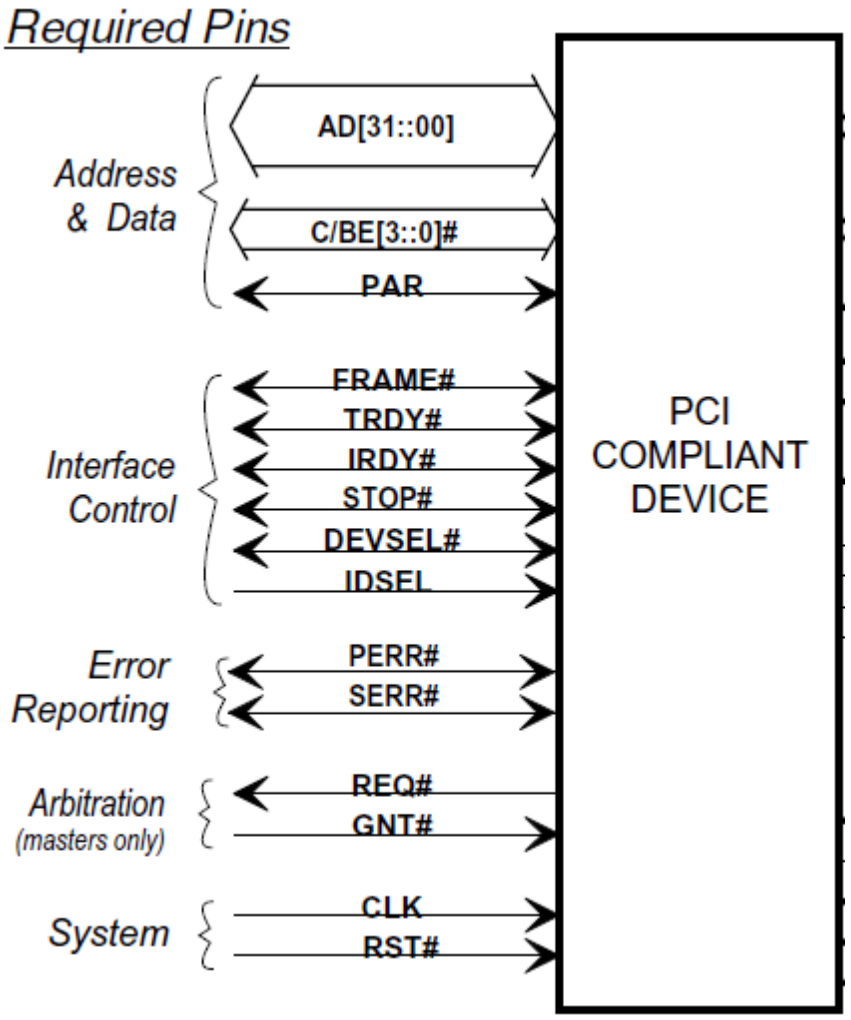
PCI总线

- **PCI = Peripheral Component Interconnect**
- **用于扩展外设的局部总线**



PCI总线信号

- 地址&数据
- 控制
- 仲裁
- 时钟复位
- 中断*(非强制)*



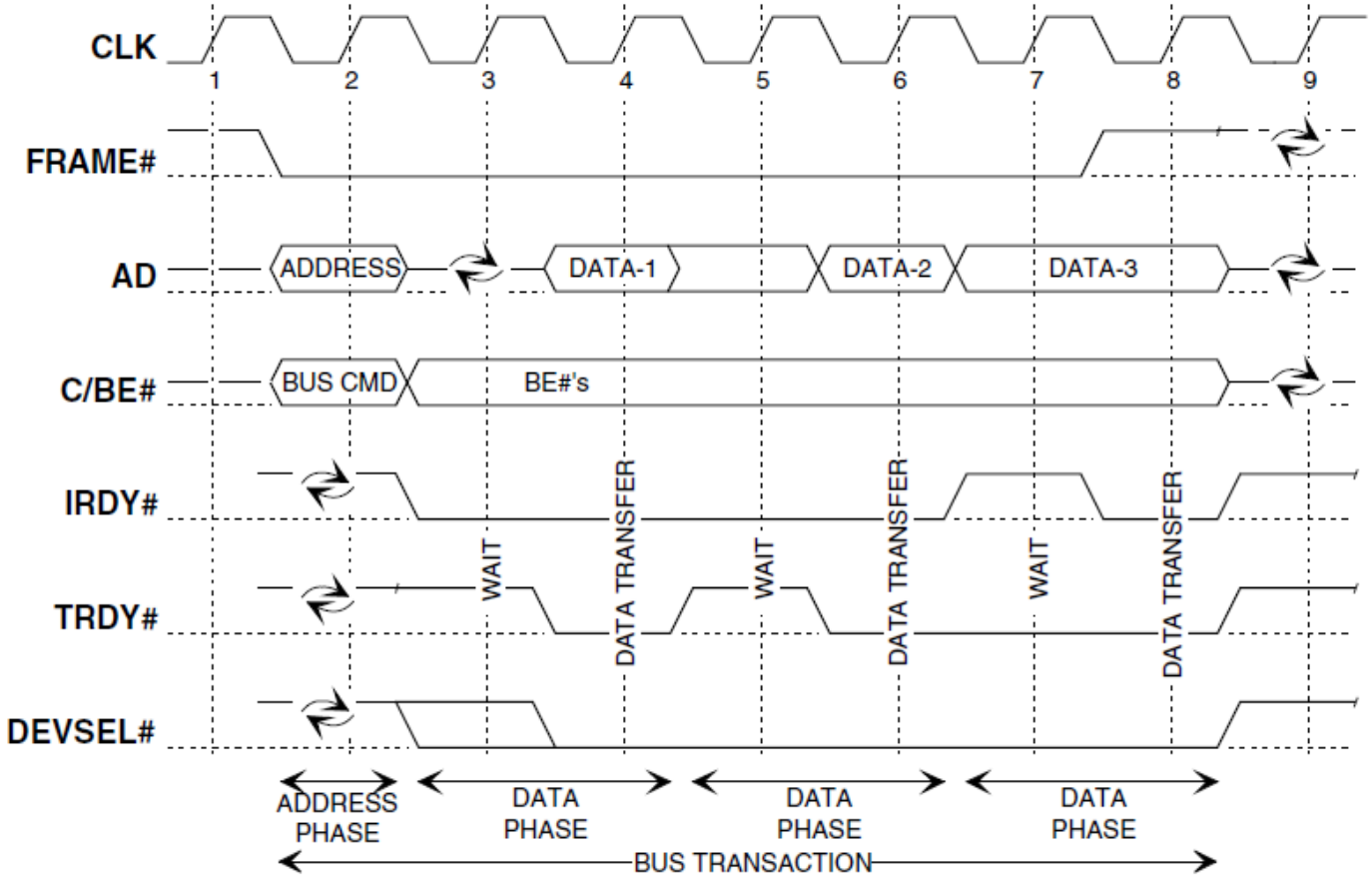
PCI总线特性

- 支持多种类型的访问
  - 基于地址的
    - 内存读/写
    - IO读/写
  - 基于ID的
    - 配置读/写(IDSEL)
- 三态总线
  - 使用中央仲裁器
  - 支持总线停靠
- 支持软件自动配置
- 支持32/64位

C/BE[3::0]#	Command Type
0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Reserved
0101	Reserved
0110	Memory Read
0111	Memory Write
1000	Reserved
1001	Reserved
1010	Configuration Read
1011	Configuration Write
1100	Memory Read Multiple
1101	Dual Address Cycle
1110	Memory Read Line
1111	Memory Write and Invalidate

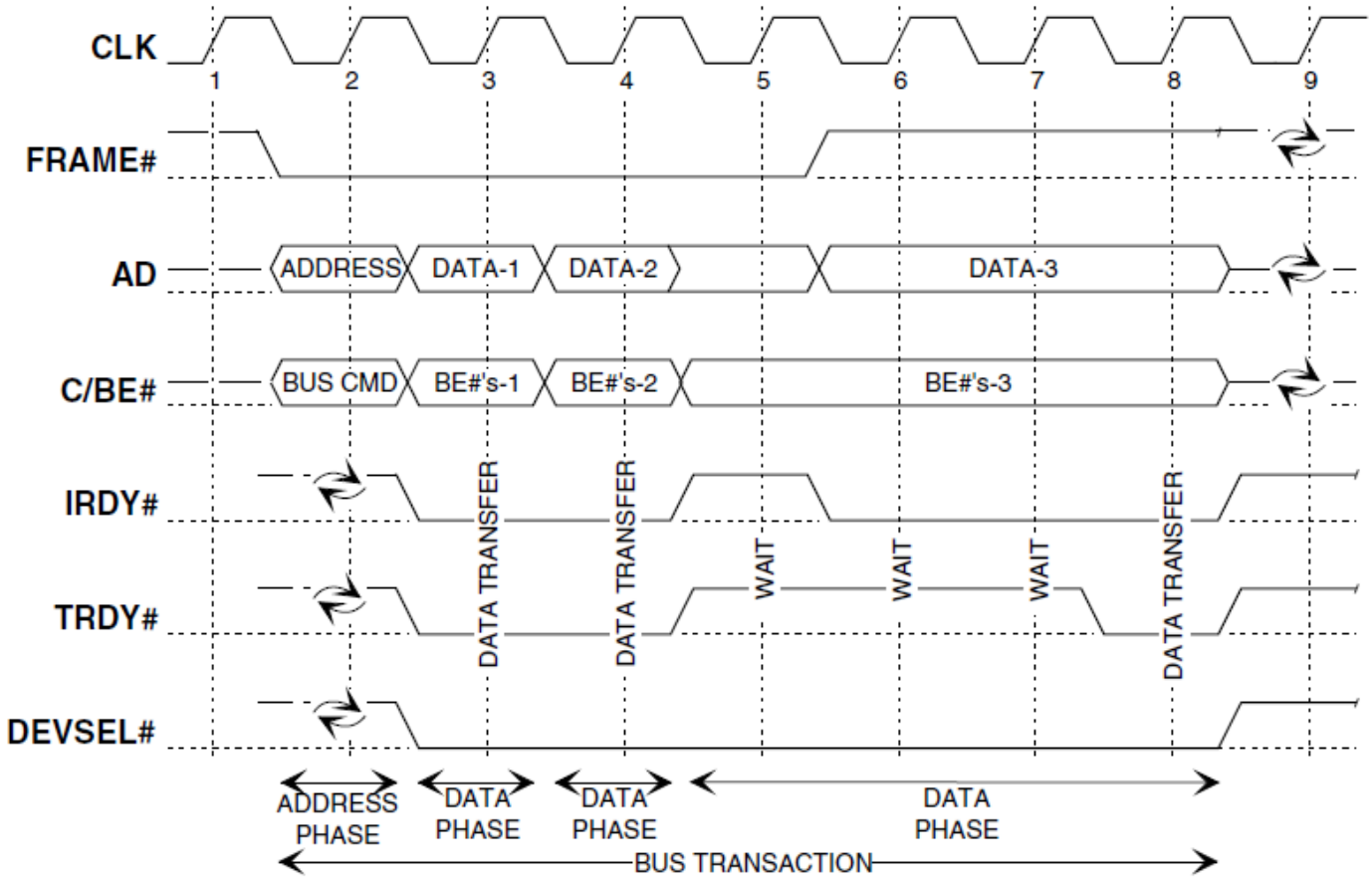
PCI总线事务——读

- 主设备发起，从设备译码、响应(置DEVSEL)
- 数据在IRDY、TRDY同时有效时传输



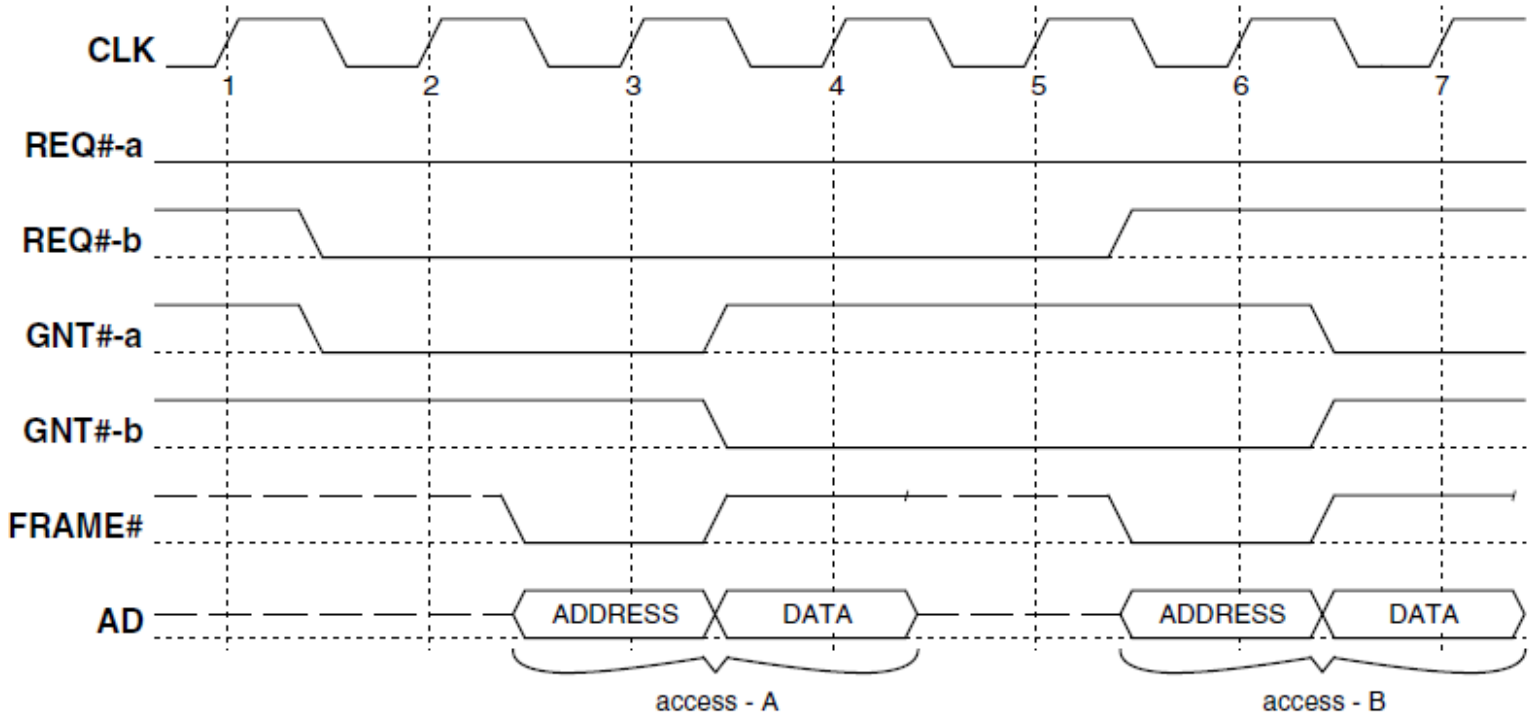
PCI总线事务——写

- **AD上无TA周期**



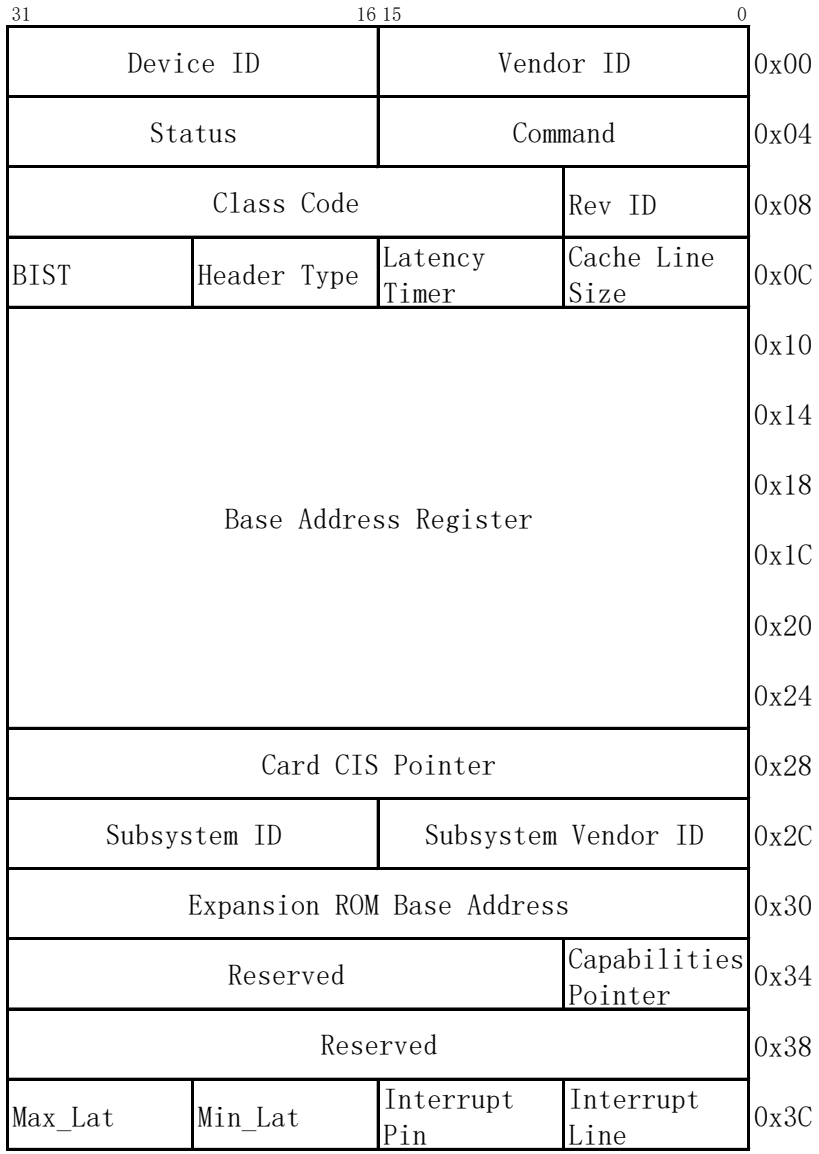
PCI总线仲裁

- 拿到GNT的设备在总线空闲后控制总线
- 总线仲裁可在传输过程中进行



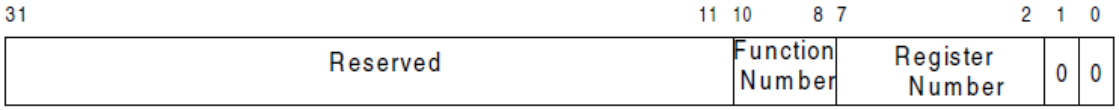
PCI设备配置头

- 设备信息
  - 厂商号、设备号
- 状态与控制
  - **IO/MEM/DMA/中断使能**
  - 中断/能力/错误状态
- 基址
  - 设备向系统声明所需要的空间
  - 系统给设备分配基址
  - **[0]: MEM/IO, [3]:可预取**
  - 比所占空间高的位可写
- 扩展能力

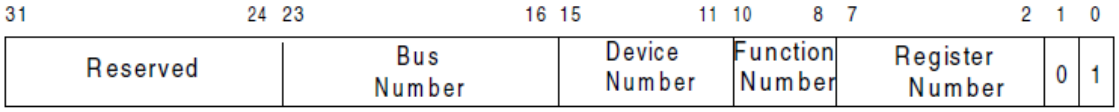


PCI配置访问

- 找到设备并访问其配置空间
  - 配置访问在PCI初始化前可用，使用片选机制
  - 主板上AD的高位与PCI槽的IDSEL相连
  - 扫描AD高位，发配置访问，如果有响应则发现设备



Type 0



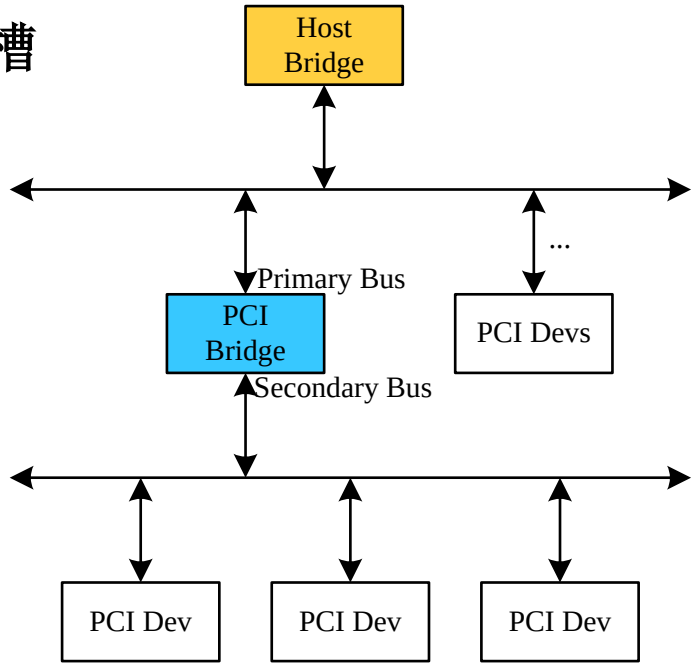
Type 1

PCI桥

- 如果没有桥
  - 扩展性严重受限
    - 一个PCI总线上的设备数量与性能直接相关

      32bit*33MHz = 133MB/s：5个插槽

      32bit*66MHz = 266MB/s：1-2个插槽
- 桥的功能
  - 转发PCI请求
  - 提供配置通路
  - 转换成其它总线（如ISA）
- 特殊的桥
  - **HOST桥：连接处理器域和PCI域**

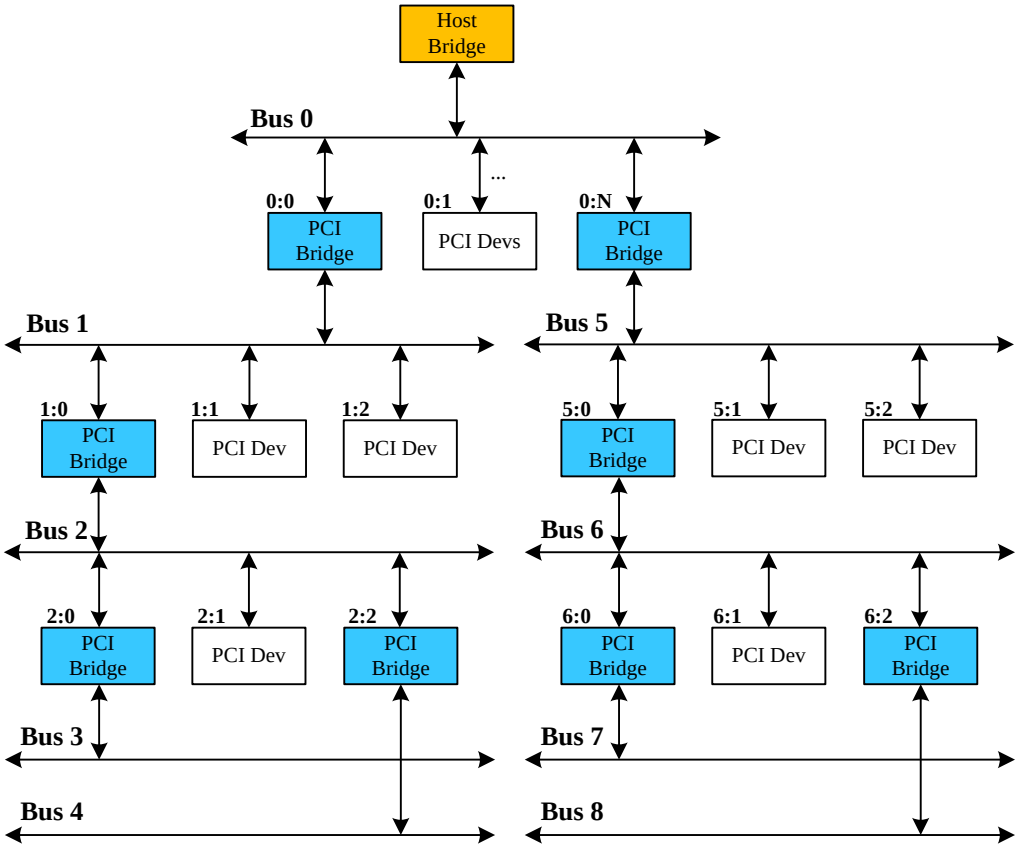


PCI总线树

- 桥引入额外的复杂度
  - **HOST为根的树形结构**
    - 设备为叶结点
    - 桥扩展下级总线
    - 每个总线都有仲裁器
    - 总线内部传输不跨桥
  - 需要有跨桥传输机制
    - **MEM/IO：**

      根据桥下地址空间译码
    - **CFG：**

      根据总线号转换
  - **Posted v.s. Non-Posted**



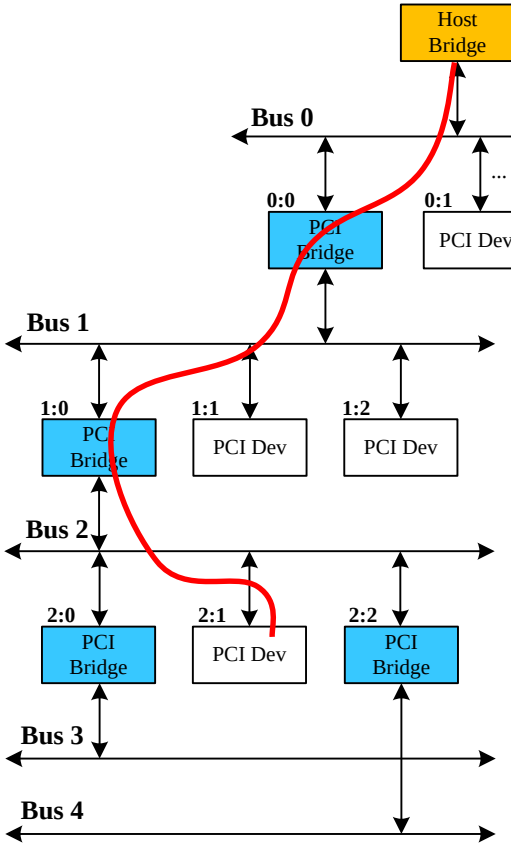
MEM Wr : P

Others : N-P

PCI总线树访问举例

- 处理器读访问设备2:1的MEM空间
  - HOST桥发出读请求，地址在2:1 BAR范围内
  - 0:0匹配读地址，在其下游总线地址中，响应HOST读请求，要求重试，同时转发到Bus1
  - 1:0匹配读地址，在其下游总线地址中，响应0:0读请求，要求重试，同时转发到Bus2
  - 2:1响应读，数据返回到1:0
  - 桥0:0重试，桥1:0给响应
  - HOST桥重试，桥0:0给响应

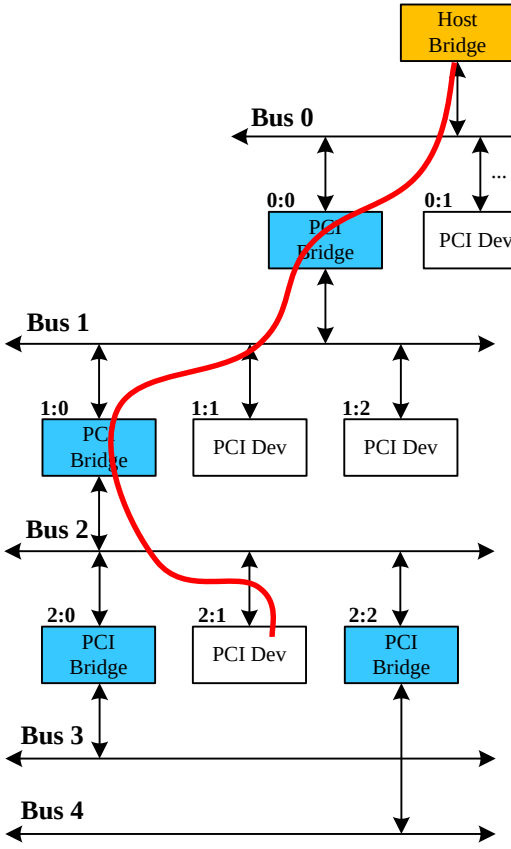
  重试间隔未定义，消耗总线周期



效率有点低...

PCI总线树访问举例

- 处理器读访问设备2:1的配置空间
  - HOST桥发出type1读请求，指定2:1的配置头
  - 0:0匹配总线号，在其下游总线中，响应HOST读请求，要求重试，同时转发到Bus1
  - 1:0匹配总线号，与其下游总线号一致，响应0:0读请求，要求重试，同时转换成type0转发到Bus2
  - 2:1响应读，数据返回到1:0
  - 桥0:0重试，桥1:0给响应
  - HOST桥重试，桥0:0给响应



PCI配置过程

- 从HOST开始用Type0配置访问扫描Bus0
  - 如果发现桥，则进行深度优先遍历
    - 分配总线号，用Type1配置访问往桥下扫描
    - 记录桥后的总线号范围
    - 记录地址空间，覆盖桥后总分配空间
  - 如果是普通设备
    - 读BAR，确定类型
    - 写全1，读，确定BAR申请的空间大小
    - 所申请大小对应的低位读恒为0
    - 高位可写，用于基址设置
    - 分配地址，写BAR
- 加载设备驱动
  - 根据设备信息找到对应的驱动，进行设备相关初始化

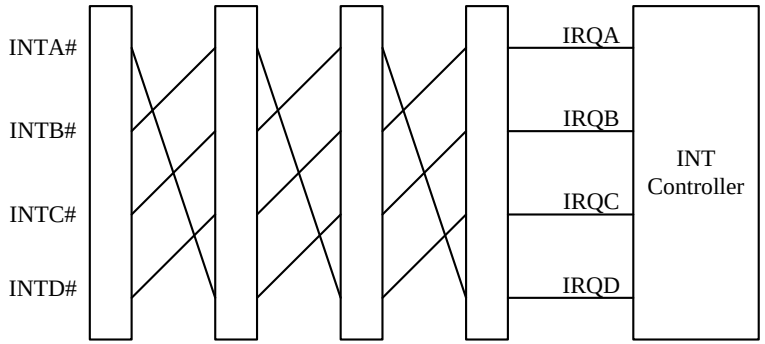
PCIe/HT尽管底层机制完全不同，但软件使用方法一致

PCI树

```
$ lspci
00:00.0 Host bridge: Advanced Micro Devices, Inc. [AMD] RS780 Host Bridge
00:01.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (int gfx)
00:02.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780 PCI to PCI bridge (ext gfx port 0)
00:04.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (PCIE port 0)
00:05.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (PCIE port 1)
00:06.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780 PCI to PCI bridge (PCIE port 2)
00:07.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (PCIE port 3)
00:09.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (PCIE port 4)
00:0a.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] RS780/RS880 PCI to PCI bridge (PCIE port 5)
00:11.0 SATA controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 SATA Controller [IDE mode
00:12.0 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI0 Controller
00:12.1 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0 USB OHCI1 Controller
00:12.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Controller
00:13.0 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI0 Controller
00:13.1 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0 USB OHCI1 Controller
00:13.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Controller
00:14.0 SMBus: Advanced Micro Devices, Inc. [AMD/ATI] SBx00 SMBus Controller (rev 3c)
00:14.1 IDE interface: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 IDE Controller
00:14.2 Audio device: Advanced Micro Devices, Inc. [AMD/ATI] SBx00 Azalia (Intel HDA)
00:14.3 ISA bridge: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 LPC host controller
00:14.4 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] SBx00 PCI to PCI Bridge
00:14.5 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI2 Controller
01:05.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI] Device 9615
02:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI] Seymour [Radeon E6460]
02:00.1 Audio device: Advanced Micro Devices, Inc. [AMD/ATI] Caicos HDMI Audio [Radeon HD 6400 Series]
03:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Con
05:00.0 Network controller: Realtek Semiconductor Co., Ltd. RTL8188EE Wireless Network Adapter (rev 01)
```

PCI中断支持

- 主要使用中断线
  - 插槽上有4根中断线INTa~d
  - 采用低电平有效的方式共享，板级上拉
  - 中断与中断控制器的连接未定义，与实现相关
  - 桥不对中断线进行处理，更加复杂化
  - 需要在BIOS中记录主板上中断连接关系
  - 存在异步的问题
- 后期引入MSI
  - 但用的不多，软件也没改过来



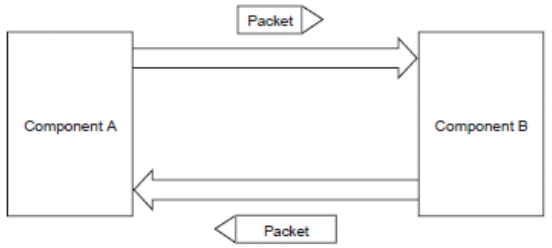
PCI→PCIe

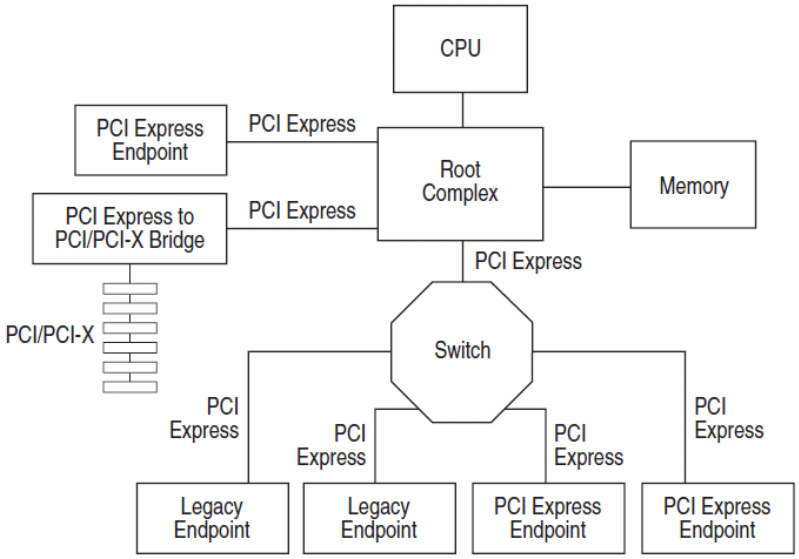
- **PCI总线的问题**
  - **理论带宽受限**
    - **64位66MHz：533MB/s**
    - **进一步提高频率、增加位宽非常困难**

      PCI的改进版PCI-X有64位、266MHz的实现
  - **总线重试严重降低效率**
    - **特别是跨桥访问**
  - **无法提供服务质量**
- **PCIe总线的思路**
  - **抛弃并行共享总线，改用串行差分线**
  - **引入网络报文概念，在传输错误时重试**
  - **引入虚通道**

PCIe总线

- 高速串行总线，用于替代PCI总线
  - 速率高PCIe1.0/2.0/3.0：2.5/5/8Gbps
  - 总线信号宽度扩展容易
    - x1，x4，x8，x16
  - 因为串行点对点连接
    - PCI在系统连接时，处理器端只需一个PCI接口即可连接多个设备
    - PCIe针对每一个设备则需要一个接口/控制器
- 软件协议层
  - PCIe/PCI/HT基本兼容
  - 扩展连接方便





PCIe总线信号

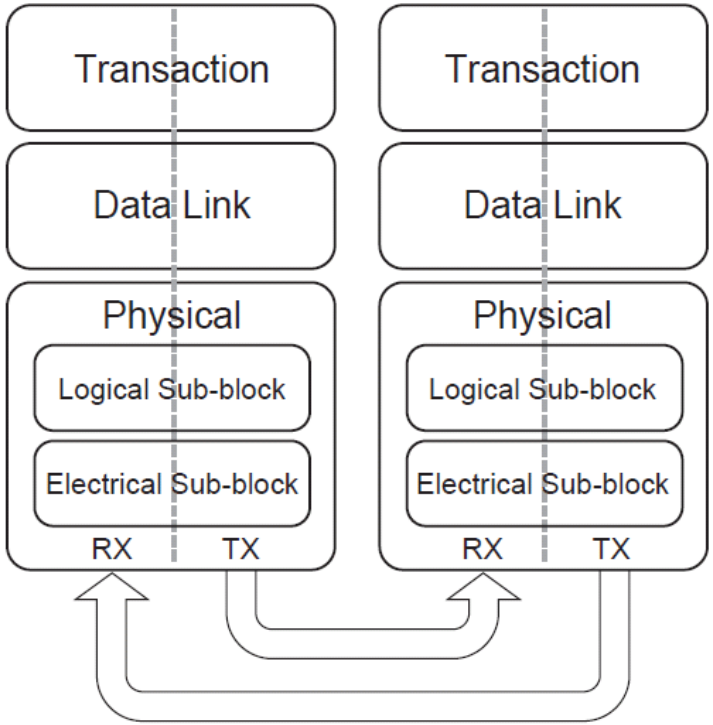
- 分为接收端与发送端
  - 差分传输

引脚名称	方向	描述
TXp/TXn[n:0]	输出	发送信号
RXp/RXn[n:0]	输入	接收信号

- 与HT相比，信号更少
  - 没有CLKp/n，采用编码的方式将时钟信号内嵌在数据传输中
    - PCIe 1.0/2.0: 8b10b编码(8’b0 -> 100111 0100，DC平衡)
    - PCIe 3.0: 128b130b编码
  - 传输信号宽度扩展更方便
    - HT一般以8位为一组扩展为x8或x16，当然也可以采用x2宽度
    - PCIe的扩展以位为单位，每个信号位传输时比较独立。但考虑系统使用方便，一般为x1、x4、x8、x16

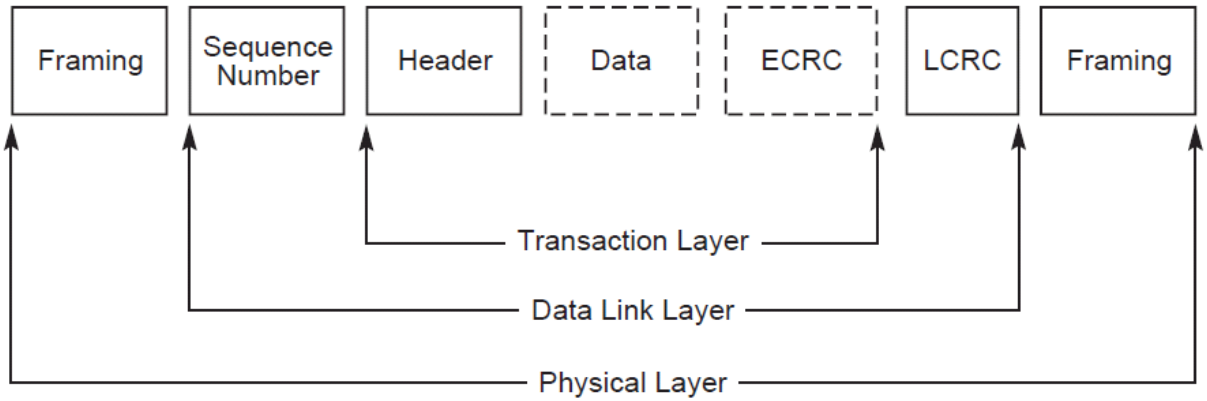
PCIe协议层次

- 事务层
  - “上层应用”与总线通讯的转换
  - 四种类型报文MEM/IO/CFG/Msg
  - **Transaction Layer Packets(TLP)**
  - 流量控制、虚通道
- 数据链路层
  - 链路管理，传递TLP，错误检测与重传
  - **Data Link Layer Packets(DLLP)**
- 物理层
  - 数据编码，并串、串并转换
  - 发送、接收电路



PCIe报文

- 与网络层次结构的做法类似
- 各个层次有对应的头尾，自动封拆



Max_Payload_Size：一个TLP包中数据最大长度

Max_Read_Request_Size：最大读请求长度

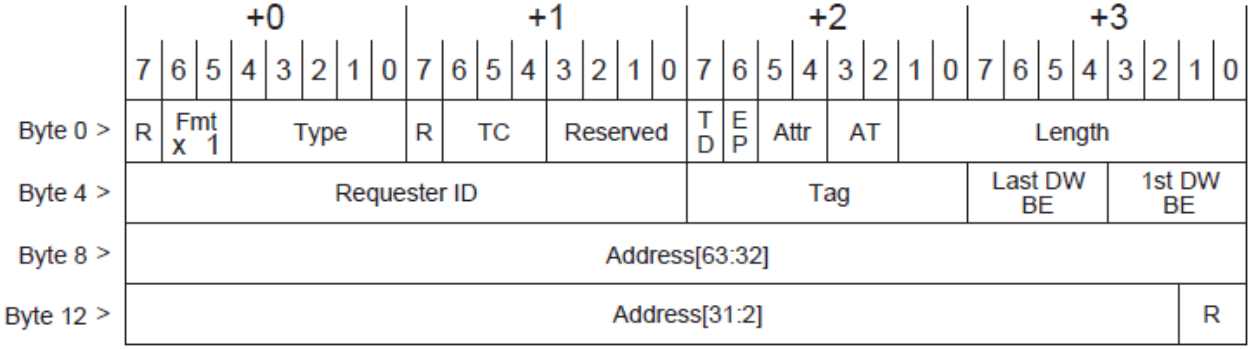
TLP包类型

- MEM读/写
- IO读/写
- CFG读/写
  - Type0/Type1
- 消息
- 响应包
  - 读数据返回
  - IO/CFG写响应

TLP Type	Fmt [1:0][2] (b)	Type [4:0] (b)	Description
MRd	00 01	0 0000	Memory Read Request
MRdLk	00 01	0 0001	Memory Read Request-Locked
MWr	10 11	0 0000	Memory Write Request
IORd	00	0 0010	I/O Read Request
IOWr	10	0 0010	I/O Write Request
CfgRd0	00	0 0100	Configuration Read Type 0
CfgWr0	10	0 0100	Configuration Write Type 0
CfgRd1	00	0 0101	Configuration Read Type 1
CfgWr1	10	0 0101	Configuration Write Type 1
TCfgRd	00	1 1011	Deprecated TLP Type[3]
TCfgWr	10	1 1011	Deprecated TLP Type[3]
Msg	01	1 $0r_2r_1r_0$	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-12).
MsgD	11	1 $0r_2r_1r_0$	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-12).
Cpl	00	0 1010	Completion without Data – Used for I/O and Configuration Write Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.
CplD	10	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions.
CplLk	00	0 1011	Completion for Locked Memory Read without Data – Used only in error case.
CplDLk	10	0 1011	Completion for Locked Memory Read – otherwise like CplD.
			All encodings not shown above are Reserved.

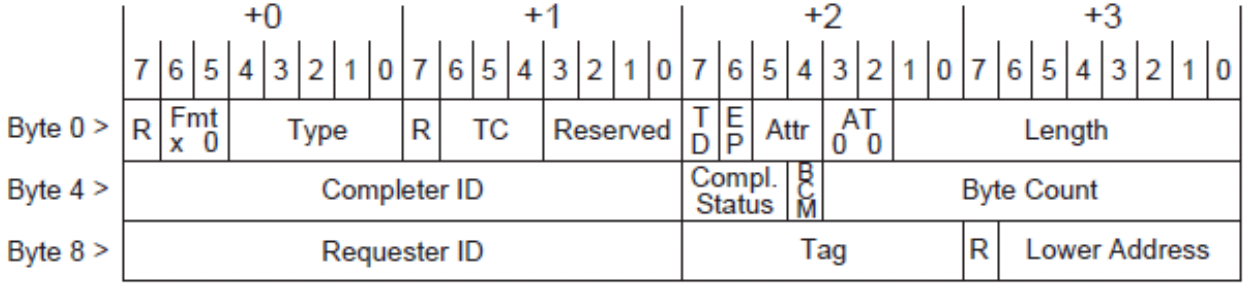
TLP包示例

- **64位地址的MEM访问**
  - **Fmt/Type：TLP类型**
  - **TC(Traffic Class)：映射到不同Virtual Channel实现QoS**
  - **Length：请求传输的长度**
  - **Requester ID：Bus:Dev:Func标识**
  - **Tag：唯一标识号**



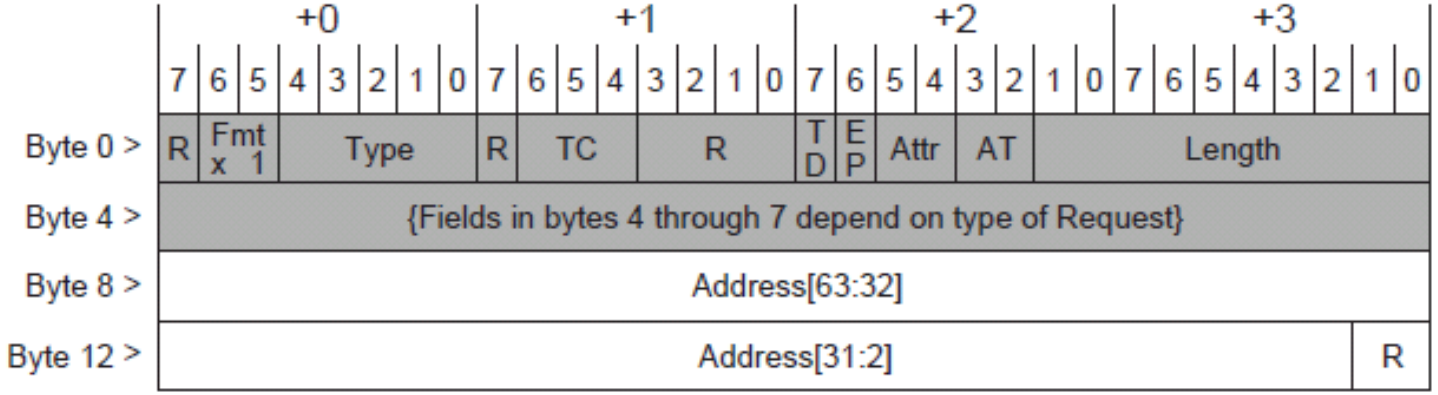
TLP包示例

- 响应(CPL)报文
  - **Completer/Requester ID：Bus:Dev:Func标识**
  - **Compl Status：完成状态(成功/不支持/配置重试/错误)**
  - **Byte Count：剩余字节数**
  - **Tag：唯一标识号，与请求的Tag对应**

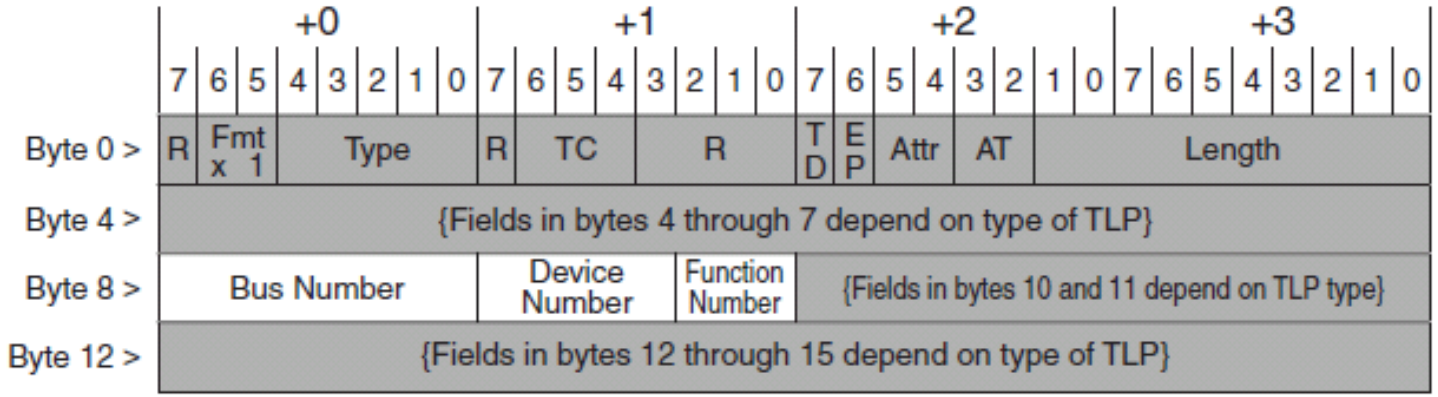


TLP包路由方式

- **基于地址**



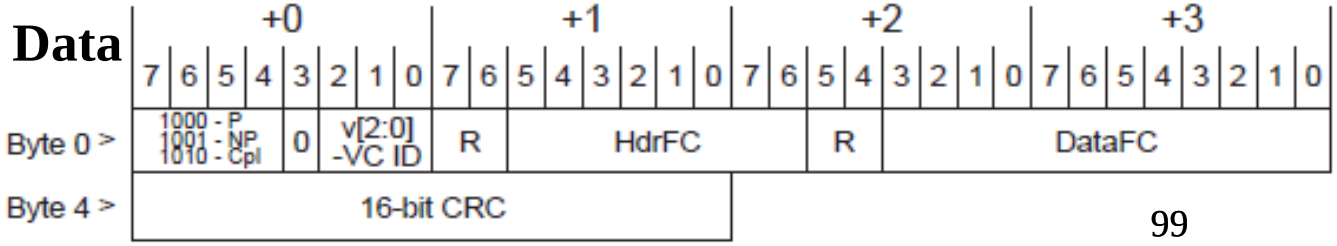
- **基于ID**



Msg类型使用隐式路由

TLP包流控

- **Credit-Based**
  - **在直接相连的链路两端之间维护(如Switch-Switch间)**
  - **链路建立时接收端通知发送端其初始缓冲大小**
  - **发送端记录链路建立以来发送的累计大小，如下一个TLP包发送后不超过接收端缓冲量，则允许发送**
  - **接收端将TLP交给应用层后将释放的缓冲数量加到总允许量上并通知发送端**
- 每个虚通道含六组流控信息
  - **P/NP/Cpl -Header**
  - **P/NP/Cpl - Data**



中断实现

- 传统中断
  - 使用Msg报文模拟
  - 带内传输Assert_INTx/Deassert_INTx消息
- MSI/MSI-X
  - 配置头中包含相应Capability结构
  - 中断消息体现为往指定内存地址写指定范围的中断向量
  - 区别
    - MSI　一个内存地址，连续分配的中断向量号
    - MSI-X多个内存地址，向量号不需要连续

作 业