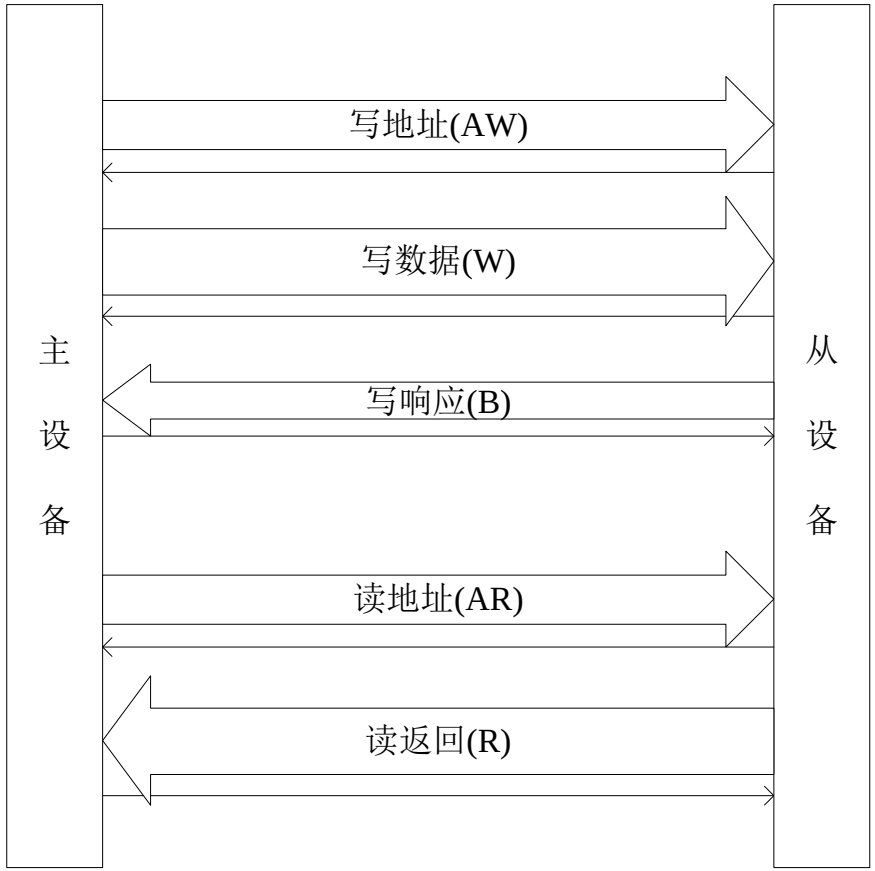
GS132总线接口设计的经历

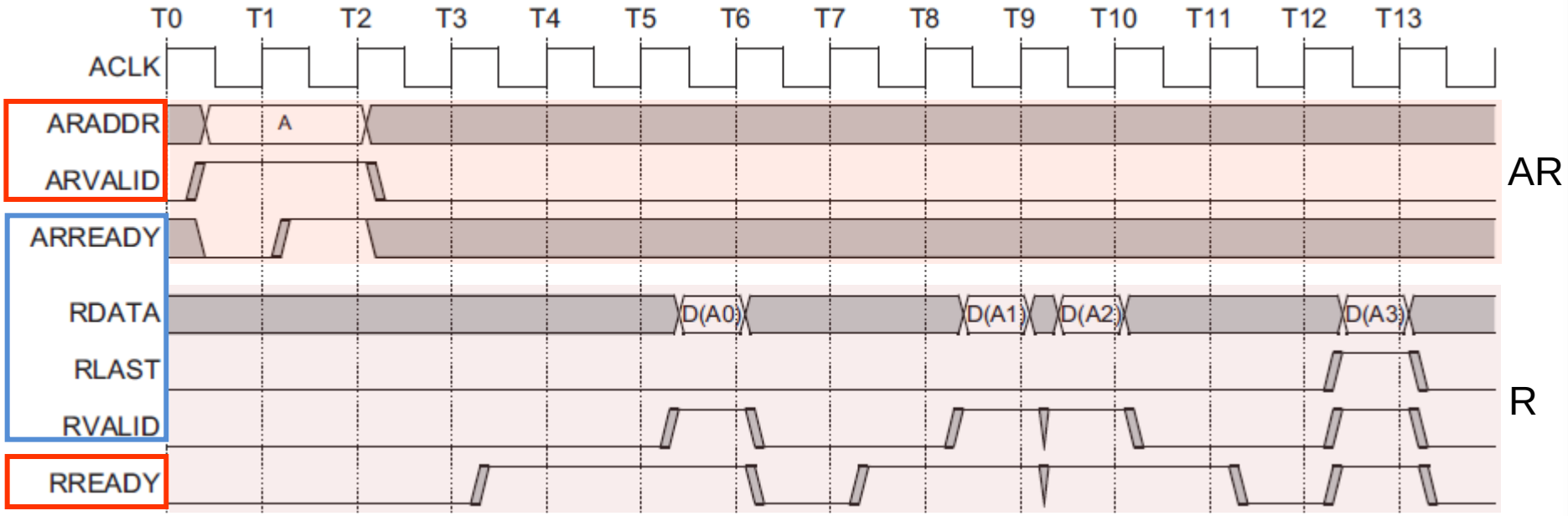
- **GS132由总线接口部分引入的设计复杂性占据了整个处理器核设计验证近2/3工作量**
- 认识一：总线响应延迟的不确定性是设计复杂性的根源。
- 认识二：尽管AXI比AHB-Lite更容易写出一个高效率的Master接口，但其具备的地址通道和数据通道分离的特性进一步引入了设计复杂性。
- 认识三：离开SoC芯片全局视角，片面追求核的总线访问效率和资源开销，也许事倍功半。

基础概念——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



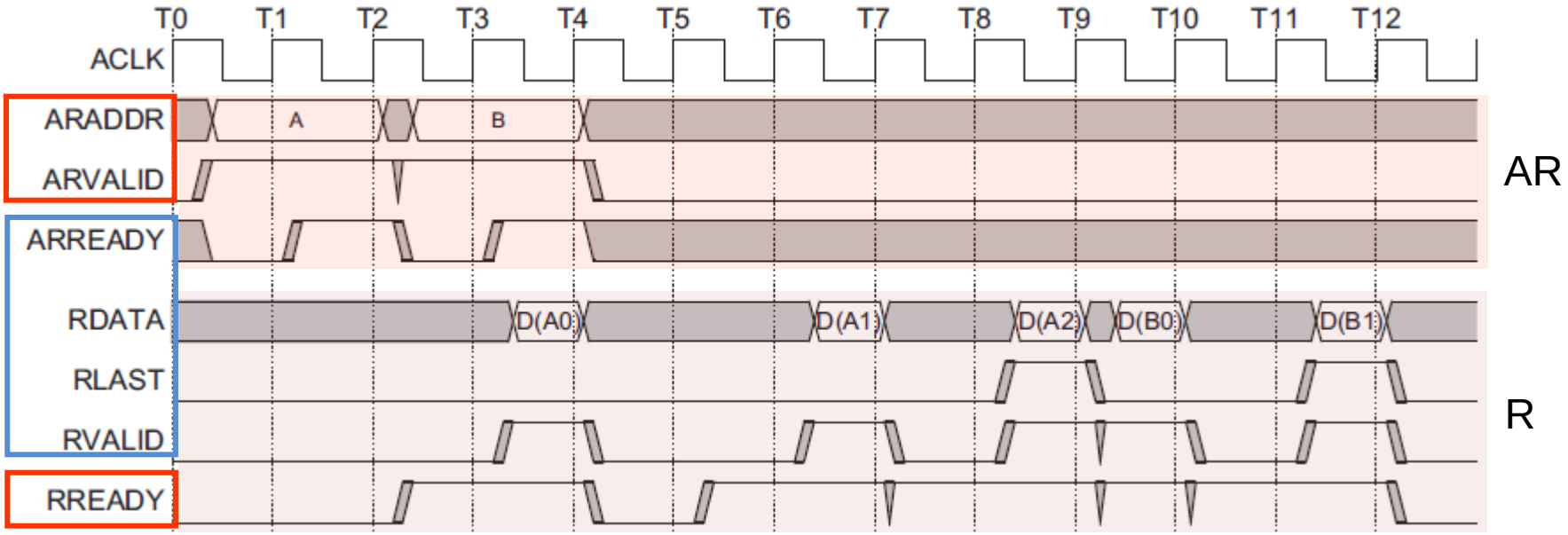
基础概念——AXI读事务示例



master out

slave out

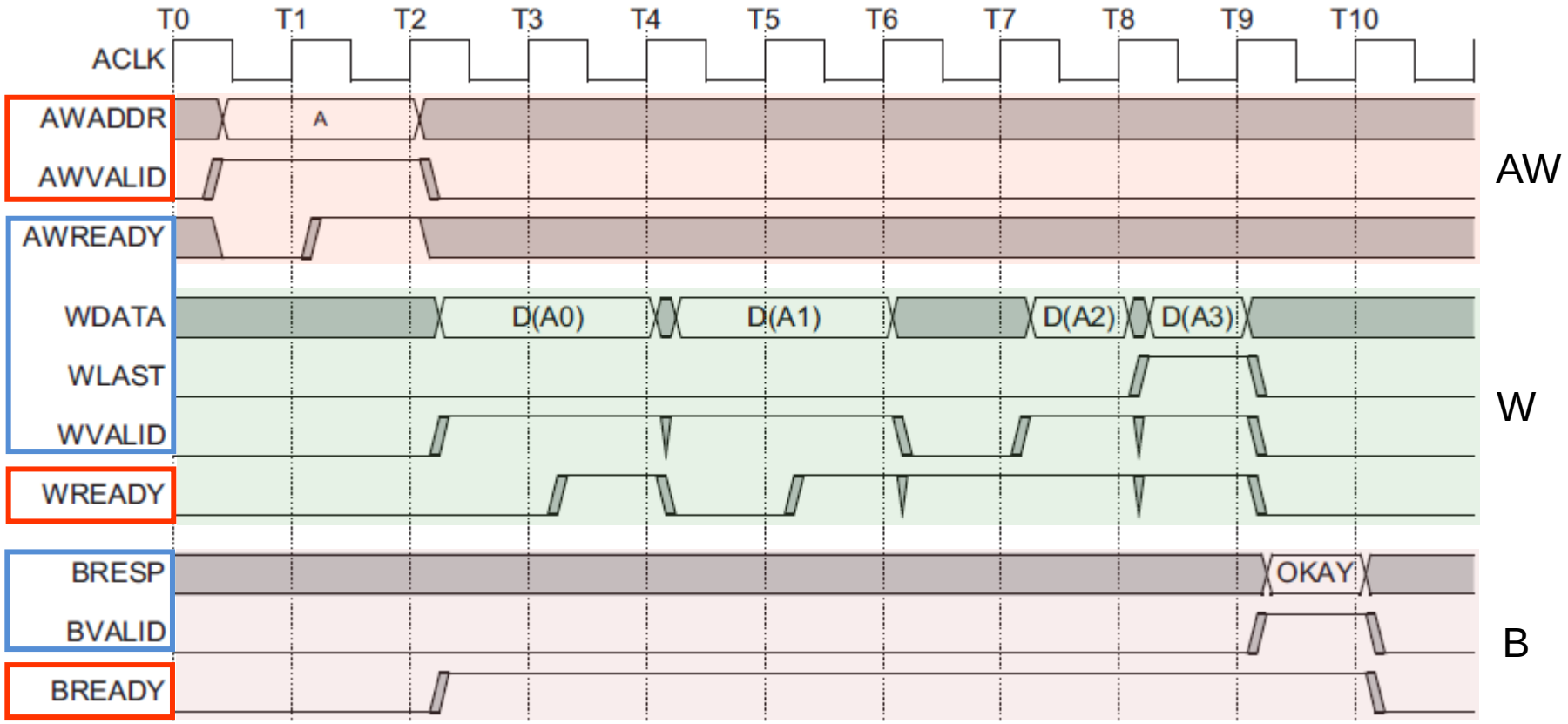
基础概念——AXI重叠读事务示例



master out

slave out

基础概念——AXI写事务示例



master out

slave out

- **读通道握手依赖关系**

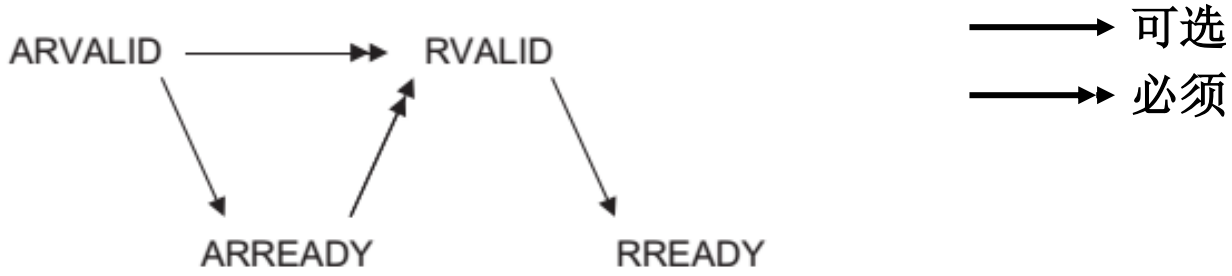


Figure 3-4 Read transaction handshake dependencies

- **写通道握手依赖关系**

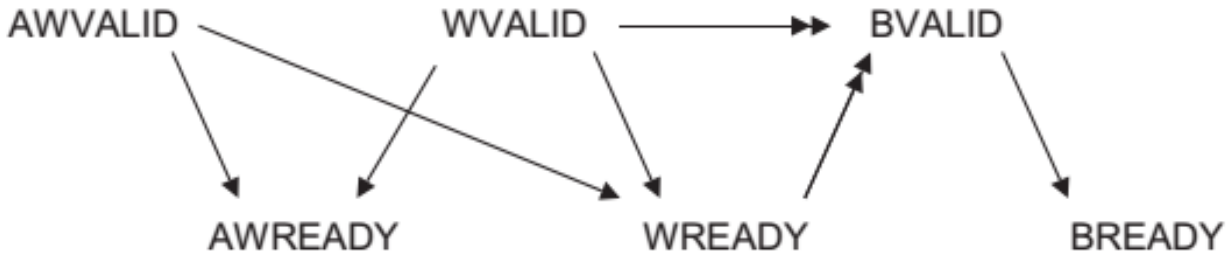


Figure 3-5 Write transaction handshake dependencies

基础概念——AXI其他关键信号

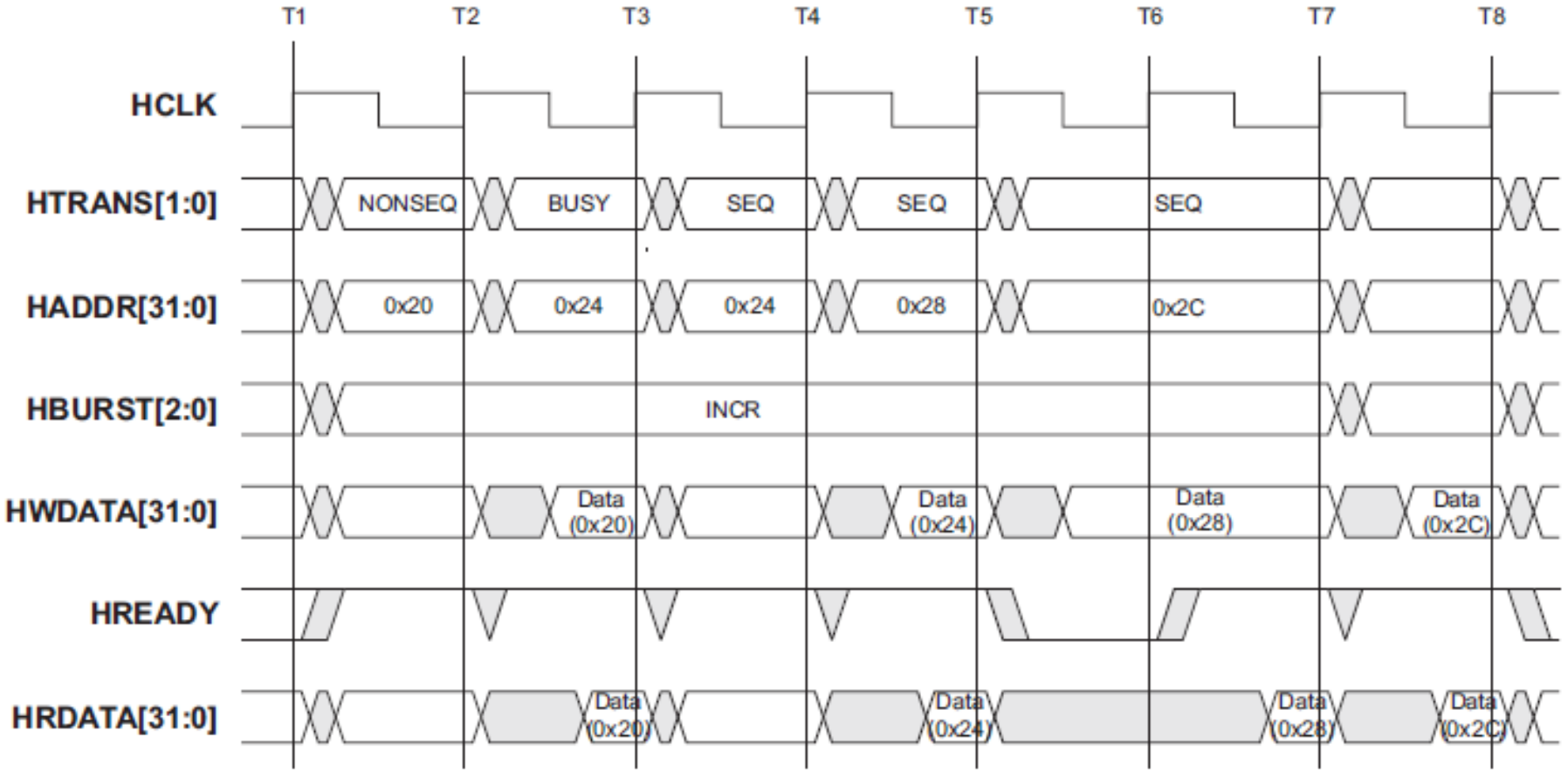
- **ID: 支持读写乱序**
  - 每个事务都有ID标签(ARID/RID/AWID/WID/BID)
  - 同ID的事务必须按序
  - 不同ID的事务可以乱序
- **BURST/LEN/SIZE: 突发传输控制**
  - 突发传输类型: FIXED, INCR, WRAP
  - 突发传输长度: 1~16
  - 突发传输单位: 1,2,4~BW
- **WSTRB: 写掩码**
  - 为高的位对应的字节写有效
- **RESP: 读/写响应状态**
  - 响应类型：OKAY、EXOKAY、SLVERR、DECERR

基础概念——AXI其他关键信号

- **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
- **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

基础概念——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



总线接口设计思考

- **问题1-1：总线响应的不确定性对流水线控制的影响**
  - 取指发出的请求并不一定会被立刻响应，取指请求响应后下一拍不一定会返回指令
  - 访存发出的请求并不一定会被立刻响应，Load请求响应后下一拍不一定会返回数据
- **问题1-2：每个总线事务需要完整执行完毕，对流水线控制的影响**
  - 复位期间不要对外发起总线请求，此时外部总线或从设备可能尚未结束复位，不能响应总线请求，有可能造成各状态机进入错误状态。
  - 如果因为控制的不精细造成同一条指令的取指或同一条访存指令的访存请求重复发出多次，对于普通的SRAM而言没有风险，但是一旦发到总线上，则每个请求都是一个独立的事务，都必须有始有终的处理完，而且每个事务都会产生其执行效果。
  - Uncache属性的load指令，不能够推测执行。即如果一条uncache load指令会被例外取消，那么这条uncache load指令不能发出总线请求。
  - 若完全兼容AXI总线协议，请求一旦置起后，如果没有响应，则不能更换请求。这个要求导致，一个取指请求置起后，如果此时发生例外清空流水线从例外入口处取指，需要仔细控制好总线接口上的请求。

总线接口设计思考

- **问题2：单个总线接口与哈佛结构的关系**
  - 流水线的取指阶段和访存阶段可以同时发起总线访问请求，因此需要在二者之间做仲裁，没有被仲裁到的可以视作总线没有响应请求。
  - 遵循“先下后上”的原则，采用访存优先于取指的固定优先级仲裁是合理的。
  - 仲裁的是请求，返回的数据要根据仲裁的情况直接返回给相应的发起者。
  - 对于AXI总线而言，只有AR通道上才有仲裁的必要。
  - 可以利用AXI总线带有ID的特性，将取指和访存发起的读请求标记上不同的ARID，从而在R通道上返回时可以通过RID直接将数据返回至发起者。

总线接口设计思考

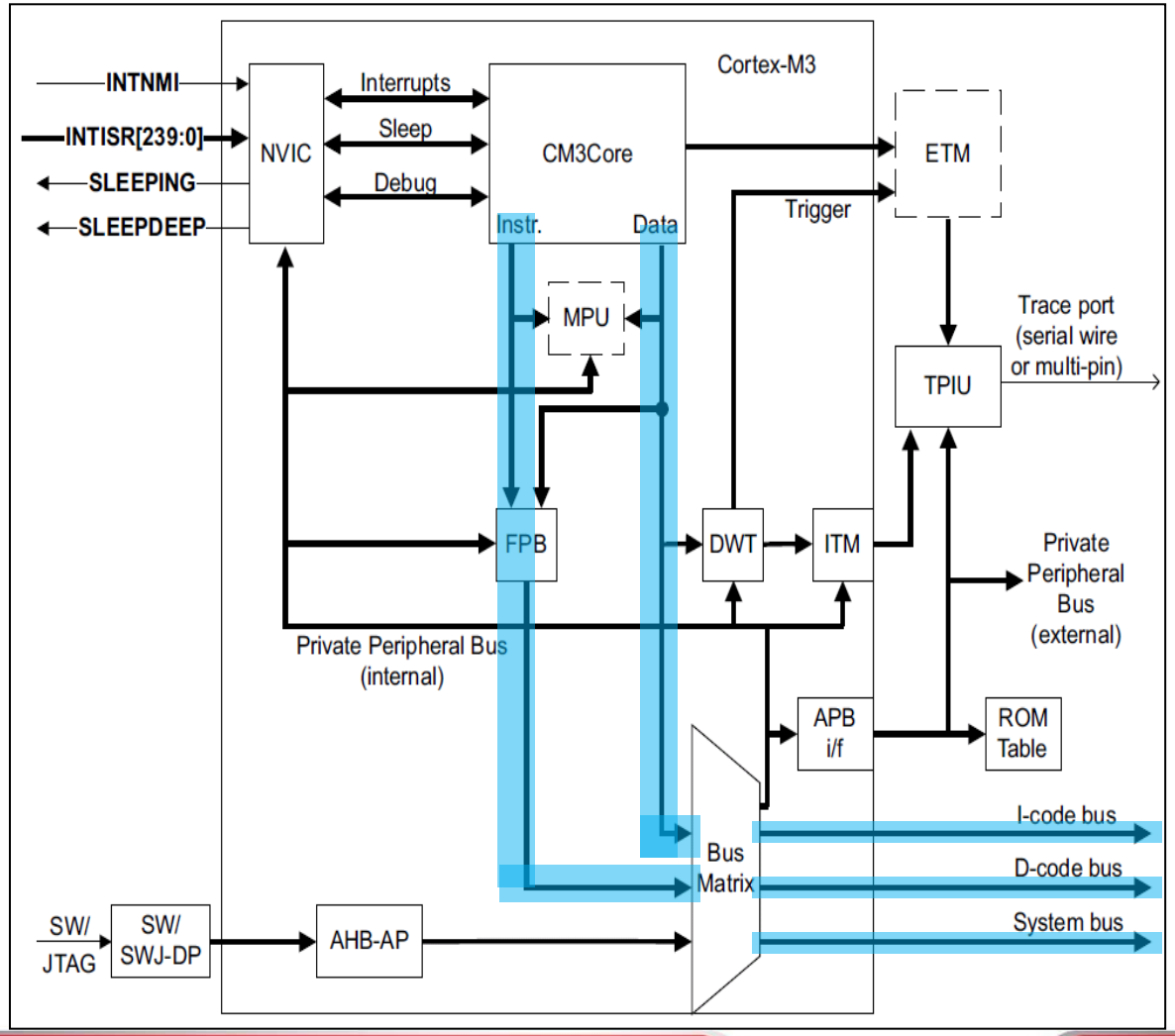
- **问题3：AXI读、写通道分离对流水线控制的影响**
  - AXI读、写通道分离，导致可能出现一种情况：当存在地址相关的总线读写序列是，在Master端写地址和写数据请求先于读请求交互完毕，但是在Slave端读请求先于写地址和写数据请求到达，从而导致读返回的数据并不是写入的新值，出错。
  - 解决的方式是：Master端在发起读请求时，先要确保该请求与“已发出请求但尚未接收到写响应的写请求”不存在地址相关。（至于说因为从设备的相关特性导致的逻辑相关，硬件无法解决，需要软件自己引入栅障操作，同步访存之间的序关系。）
  - 由上面得到一个推论，当SYNC之类的栅障指令执行时，要确保SYNC执行时，该指令之前的所有已发出的写请求都要接收到写响应。
  - 很显然，仅当总线写请求接收到写响应后才允许新的请求发出，是可以解决相关问题的，但是效率很低，可以通过设计缓存记录那些“已发出请求但尚未接收到写响应的写请求”的地址、类型等信息，供后续读请求查询判断用。

总线接口设计思考

- **问题4：AXI读请求、读返回通道分离对流水线控制的影响**
  - 通常读请求和读返回被对应于两级流水线中以最大化性能，所以当一个总线访问请求被接收后，这条指令进入下一级流水等待总线返回，而此时新指令可以发起新的请求。这种很显然的设计在面对AXI协议时存在一些需要特别要注意的地方：
  - 尽量不要给同一个请求源（如取指）分配不同的ID，这样会引入额外的负担，即前一个请求在等待数据返回时，后一个请求的数据已经返回了。为了解决这个问题，又要消耗更多的资源缓存先返回的新请求的数据。
  - 即使通过给同一个请求源分配相同ID来约束数据返回的顺序，读请求、读返回通道分离还有其它影响，我们以取指阶段为例来说明。假设取指请求和取指返回分布在两个流水级，一条指令能否从取指请求级进入取指返回级（假设当前取指返回级有一条等待返回的指令），既需要这条指令的取指请求响应，也需要等待取指返回的指令取到指令，但是对于AXI总线来说这两个条件并不一定是同时发生的，切记状态转换要涵盖所有的情况。

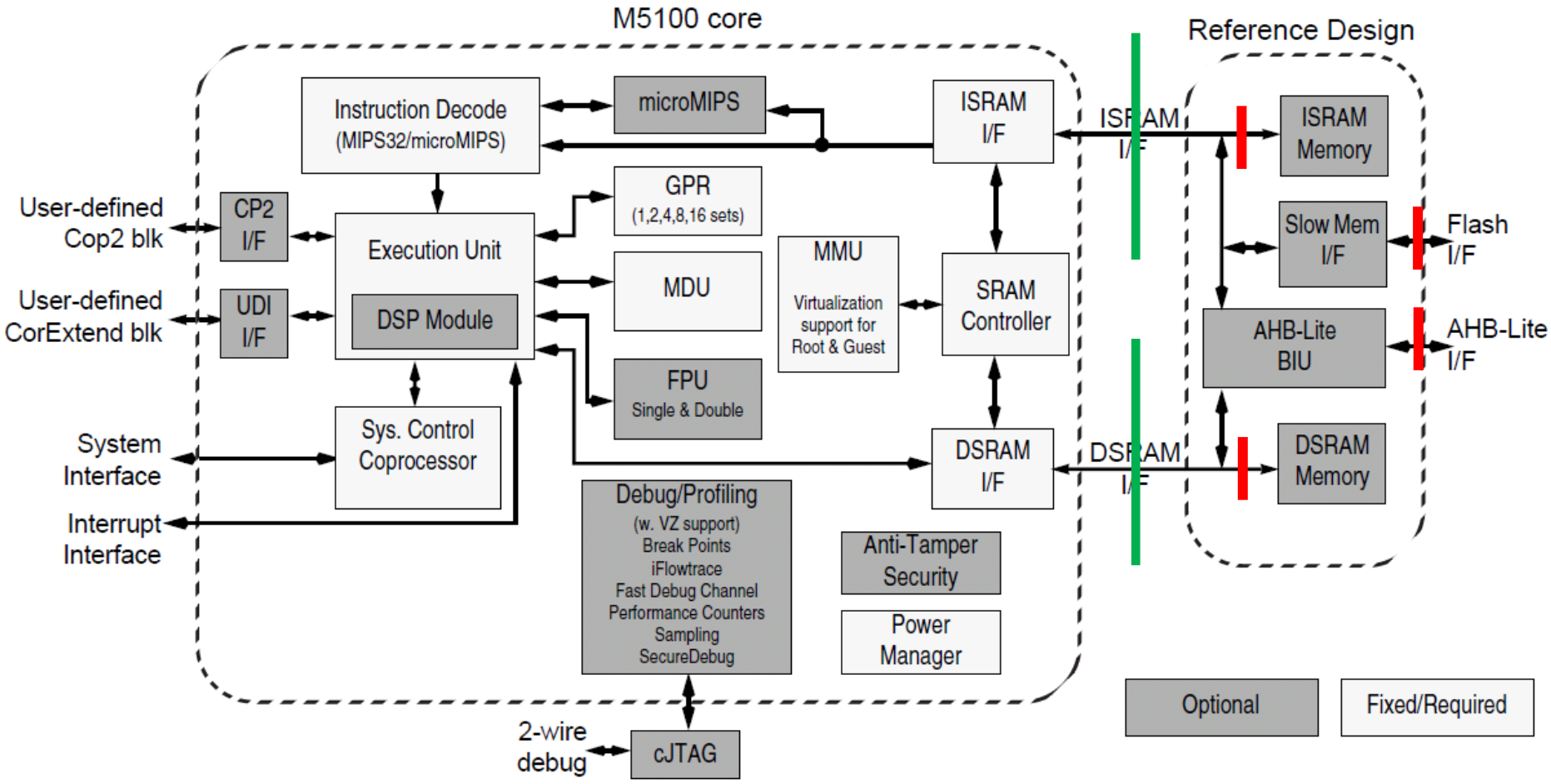
总线接口设计思考

- **问题5：AXI总线接口和类SRAM接口之间的关系**
  - Cortex-M3的顶层结构



总线接口设计思考

- **问题5：AXI总线接口和类SRAM接口之间的关系**
  - MIPS M5100顶层结构



总线接口设计思考

- **问题5：AXI总线接口和类SRAM接口之间的关系**
  - 内部流水线对外有两条接口总线，整个处理器核对外呈现三条接口总线是MCU SoC领域较为普遍和实用的架构方式。其实质上是要在接口层次实现一个2X3的bus matrix。。
  - Bus matrix的具体设计权衡主要在资源开销和性能两个维度之间。
    - 共享式
      - 互联线少
    - 交换式
      - 带宽高
  - 对于一个面向MCU的处理器，我们认为其在多个Slave间交替访问的情况并不普遍，因而共享式是一个比较适合的选择