2 片上存储系统

2.1 ICACHE 模块

L1 Icache 设计为 16KB，具体机制参见小核设计文档。

2.2 Private Memory 模块

Private Memory 指集成在 core 内部的一块 SRAM。目前 GodsonT 支持两种运行模式，即基本模式和专业模式。在基本模式中，private memory 作为 L1 Dcache 使用；而在专业模式中，private memory 作为 SPM（Scratch-Pad Memory）使用。SRAM 在 SPM 和 Dcache 之间的配置切换，通过 REMOTE_MMIO 写控制寄存器，根据控制寄存器的值确定是否为 SPM 或 Dcache。

下面分三部分介绍 private memory 模块的设计方案。首先介绍 private memory 和外界各个模块的接口。第二部分是 private memory 分别作为 L1 Dcache 和 SPM 时的设计方案。第三部分是 private memory 模块和片上其他模块之间交互的消息及含义。

2.2.1 模块接口/总线接口

2.2.1.1 SRAM 模块的环境

SRAM 与数据传输代理（Data Transfer Agent, DTA）和 LSU 都有连接，如下图所示。

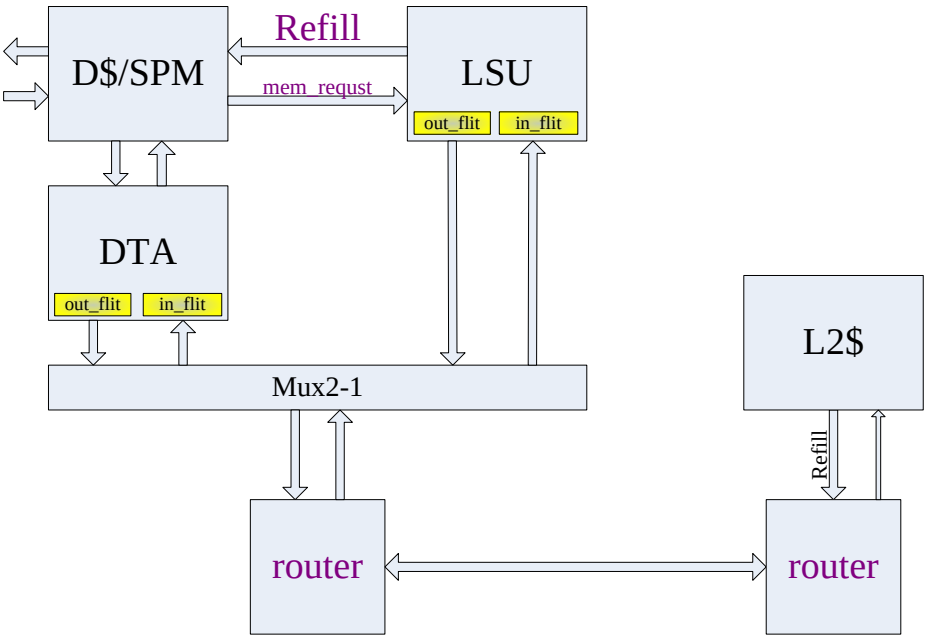


图 2.1　DTA 在 core、router 和 core 内 SRAM 之间的关系示意

在基本模式下，SRAM 配置成 DCache，与 DTA 之间没有数据通信情况，Dcache miss 通过 LSU 转发到 router 并传递到共享的 L2，L2 对 L1 Dcache 的 refill 数据也通过 LSU 返回给 Dcache；

在专业模式下，SRAM 配置成 SPM。SPM 对应于 core 的一段地址空间，当 load/store 的地址属于 SPM 的地址范围时，直接对 SPM 进行操作，没有替换等操作；当 load/store 的地址不属于 SPM 的地址范围时，通过 LSU 访问共享的 L2；

在专业模式下，DTA 可以直接访问 SPM，DTA 发送给 SPM 访问的起始地址（64 位对齐）和长度（SPM 一次最多可以读出 128bit；输出的数据长度为 64 bit 或 128bit→cache line 为 256bit，所以需要加上一个选择前 128bit 或后 128bit 的选择逻辑，或者选择哪一个 64bit 的选择逻辑），SPM 直接更新相应地址的数据，或者将数据返回给 DTA。

从 DTA 的角度可以将对 SPM 的访问分为两种，一是本地 core 对本地 SPM 的访问，二是远程 core 对本地 SPM 的访问。但是在 SPM 的角度，只有一种来自于 DTA 的操作，即 DTA 给 SPM 一个起始地址和操作数据的长度，SPM 将处理的结果返回给 DTA。

2.2.2 模块功能及设计

2.2.2.1 可配置的 Private Memory

GodsonT 中分为两种模式，即基本模式和专业模式。其中，基本模式下，片上存储层次为 L1 ICache/Dcache，L2 Cache；专业模式下，片上存储层次为 L1 Icache，SPM（Scratch-Pad Memory）和 L2 Cache。也就是说，在专业模式下，将片上 L1 Dcache 配置为程序员可控的片上存储空间 SPM，程序员可以直接对其进行存取访问。

设置程序员可控的私有片上存储空间的目的：程序员写程序时，可以将已知的常用数据放置到片上，避免 cache 中由于替换而造成的颠簸现象，从而提高程序的性能。

2.2.2.2 L1 DCACHE 工作原理

L1 Dcache 的新值传播可以有如下几种选择，各项都有优缺点：

（1）L1 设置 bit vector，临界区内 write through，临界区外 write back；

增加的存储很大，一个 cache line 为 32B，需要对每一个 cache line 增加 32bit 的存储。

（2）设置目录，支持目录协议；

实现复杂

（3）不支持一致性协议，由程序员负责；

（4）全部使用 write through——工程实现简单，暂定 write through；

（5）采用广播机制，支持 Token Coherence——适合于中等规模的多核结构（16~32 core）

思想：每个 core 只记录使用每个 cache line 的 core 的数目 N，而不记录位置。每次一个 core 对一个 cache line 进行 store 操作时，要等待获得 token，方法是向其他所有的 core 发送广播信息，等待所有使用该 cache line 的 core 反馈信息，收到 N 个反馈后就获得 token，可以执行 store 操作。

目前 GodsonT V3 的工程实现中，使用 write through 机制。（注意，现在的实现已经不是这样了，而是类似方案 1，使用 bit mask 实现临界区内 WT，临界区外 WB）

L1 Dcache 和 L2 cache 之间是 non-Exclusive 的关系（不区分是否 inclusive 和 exclusive），和目前 GodsonT V2 中一样。

（1）Dcache 的结构

L1 Dcache 的容量设计为 32KB，cache line 为 32 byte。Dcache 设计为 4 路组相联，256 bit 的 cache line，总容量为 16KB。每一路 8KB，共有 256 个 cache line。

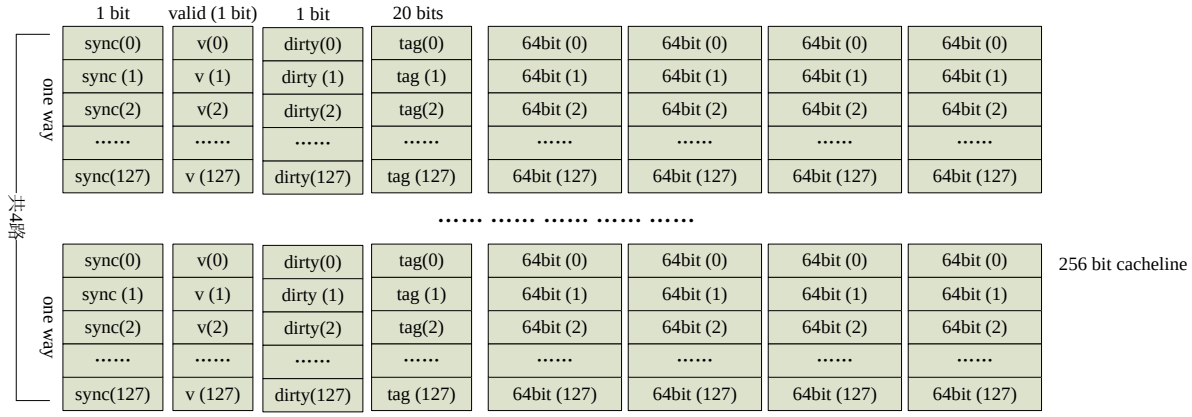


图 2.2 Dcache 内部结构

L1 Dcache 采用 write through 策略，即所有的 store 都直接写到 L2 上：如果在 L1 Dcache 中发生 store hit，则将 store 同时写到 L1 Dcache 和 L2 中；如果在 L1 Dcache 中发生 store miss，则直接将 store write through 到 L2，而不在 L1 Dcache 中分配此 cache line，即 L1 Dcache 不是写分配方式（store allocating）。

（2）Cacheline 的结构：

表 2-1 L1 Dcache cache line 的结构

名称	**宽度(bit)**	**描述**
Valid	1	Cacheline 所处状态(invalid , valid)
Tag	19	该 cacheline 的 tag （物理地址高 19 位 ）
Sync	1	标志是否需要同步的信号。1 表示在临界区内已经被回填，表示此处为最新的数据；
dirty（不采用 write back；不需要，但保留此位）	1	标志该 cache line 是否被修改过。1 表示修改过，替换时需要写回；0 表示未修改过，可以直接替换
DATA[Nblocksize_d]	4×64	一个 cache line 的数据

（3）在 L1 DCache 中维持一致性

L2 为所有 core 共享，因而只需要在 L1 Dcache 间维持一致性。

对于共享数据的访问，要求都加 acquire 和 release，在临界区中对其访问。如果对临界区中的数据进行了修改，直接 write through 到 L2 中。对于临界区中的数据，core 在首次使用时，都不信任 L1 Dcache 的备份，而是直接从 L2 中读取，即 L1 Dcache 对于临界区内首次 load 的地址都采取 invalid 后 refill 的方法实现，保证新值在多个 core 之间的传播。

当 core 执行 release 操作退出临界区时，需要保证临界区内的 store 操作都确实已经写完成，即在下一个 core 进入临界区时保证上一个临界区已经将最新修改的值写入 L2。方法是：core 发送的 release 消息需要经过 L1 Dcache，并由 L1 Dcache 沿着临界区内 store 消息的路径转发至所有的 L2 banks。

（4）L1 Dcache 的流水处理

Dcache 的一个 load 需要两拍才能读出数据，第一拍送入读 RAM 和 tag 的信号，第二拍比较 tag，并输出数据。需要将 Dcache 设计为流水处理，全部流水的 L1 Dcache 的访问延迟为 1 拍，将一次访问 L1 Dcache 的 2 拍流水处理掩盖为 1 拍。如下图。

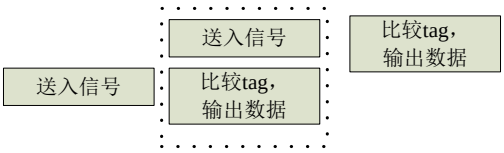


图 2.3　Dcache 流水操作示意

（5）Store Buffer 的设计

需要设计 dcache 的 store buffer，用于缓存对 cache 的 store 请求；load 时，既需要查 Dcache 也需要查 store buffer。如果 dcache hit，直接返回数据；如果 dcache miss，store buffer miss，直接将请求发送到 load miss queue，从 L2 中取数据；如果 dcache miss，store buffer hit，分为两种情况：一是 load 的数据全部在 store buffer 中命中，直接返回数据；二是，load 的数据有一部分为在 store buffer 中 hit，或者 load 的数据为 store buffer 中各项之间的交叉，则直接发送到 load miss queue，并等待 store 操作完成后再从 L2 中取数。

2.2.2.3 L1 DCACHE 的状态转换图

Dcache 的状态机：初始状态为第一个状态 empty，在流水线没有阻塞信号的情况下，从 addr 流水级过来的地址信号 dmemref 有效时，指令是读取 Dcache 的操作，且在 dcache 命中的情况下，直接进入第二个状态 out，将数据返回给 dcacheread 输出总线；如果在 dcache 没有命中的情况下，进入第三个状态 enter。在 enter 状态时给出 dcache miss 标志，并向流水线发出 stall 信号，同时向 router 发送 DCACHE_MISS_TO_L2 的消息。在 router 给出应答信号时进入第四个状态 wait_ref，等待 router 传回消息，此状态由于回填的数据并没有准备好，所以此状态下仍要向流水线发出 stall 信号。当从 router 传回回填的数据时，进入第五个状态 refill，回填 dcache，同时将数据输送给 dcacheread 输出总线。下一拍回到初始状态 empty。（无 refill 状态，修改为 refilled；在 wait_ref(ill)状态等待所有的回填数据包接收完毕，收齐再写 dcache，进入 refilled 状态；refilled 状态写 Dcache，并将数据拼好发送到下一级流水线，下一拍进入 empty 状态，或者进入 enter 状态。）

L1 Dcache 的 miss buffer 状态转换图如下所示。

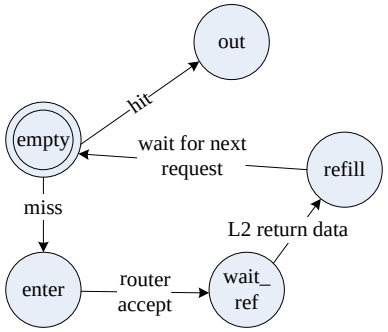


图 2.4　L1 Dcache 状态转换图

2.2.2.4 SPM 工作原理

SPM 与 Cache 的区别：SPM 通常直接对应处理器核的一段地址空间，从而程序可以更快的访问 SPM；而 cache 并不对应处理器核具体的一段地址空间，而是使用 tag 信息识别处理器核所访问的地址是否在 cache lines 中。SPM 中不必执行 cache 的各种操作(如查 tag 等)，因而访问开销较小。

SPM 的操作相当于 cache 操作的一个子集。SPM 只是一块存储空间，SPM 负责的操作只是根据输入总线输入的地址和操作数长度对相应存储单元的值操作，并返回；地址来自于地址解析模块或者 DTA。

片上所有 core 私有的 SPM 空间采取连续编址的方式，任何一个 core 可以直接使用地址访问其他 core 的 SPM 空间。

（core 对 SPM 的 load/store 需要 2 拍（读 1 拍，拼数据 1 拍）；DTA 对 SPM 的操作需要 1 拍）

（1）SPM 的结构

将 SRAM 配置为 SPM 时，将 Dcache 的 tag 配置为 full/empty 位。以两个字节为单位设置 full/empty 位，一个 cache line 对应到 SPM 中有 16 个 full/empty 位。

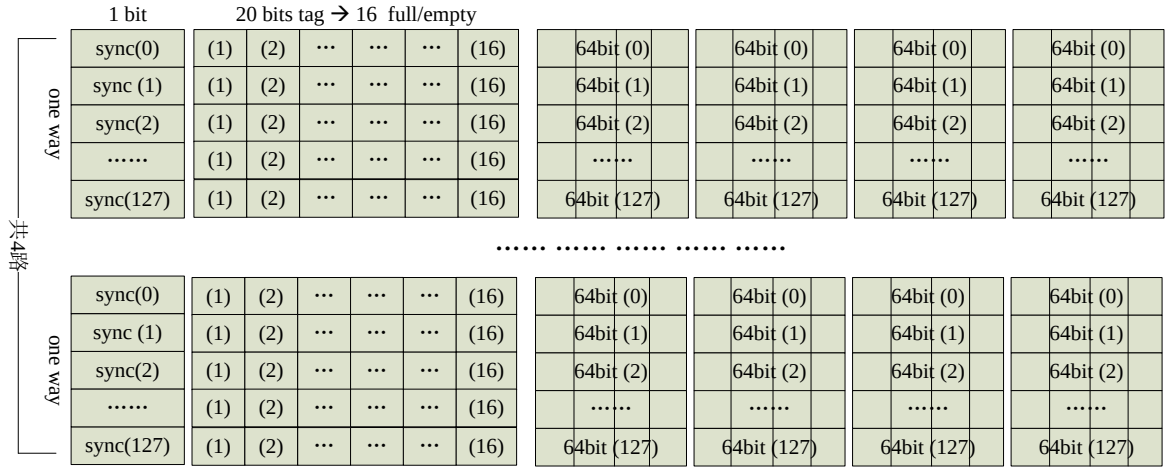


图 2.5 full/empty 位示意

（2）core 对 SPM 的操作

对 SPM 进行读写操作以 byte 为单位，core 生成 mask 位，并发送给 SPM，SPM 根据 mask 位的标识将数据写入 SPM 的相应地址中。

SPM 对应 core 的一块地址空间，Core 直接根据 load/store 操作的地址访问 SPM，如果 load/store 操作的地址不属于 SPM，则通过 LSU 对 L2 访问。SPM 的优势是访问延迟小，对 SPM 进行 load 的延迟是 2 拍。

假设 core 的地址空间表示为[0, B]，并且[0, A]是[0, B]的子集，如果程序员规定 core 的[0, A]地址范围对应于 local SPM 地址空间，则如果 load/store 的地址介于 0 与 A 之间时，直接对 SPM 进行操作；如果 load/store 的地址介于 A 与 B 之间时，需要从 L2 中读取。如下图所示。（访问远程 SPM 的示意图，core 给出私有 SPM 的地址标志，如果是远程 SPM 操作，直接给 store buffer）

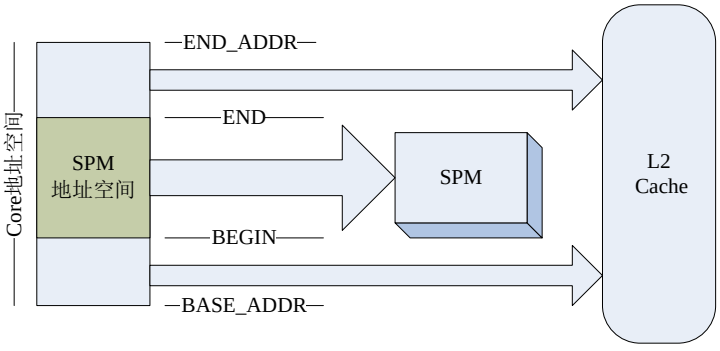


图 2.6 SPM 地址空间及访问

（3）DTA 对 SPM 的操作

对 SPM 进行读写操作以 byte 为单位，SPM 返回给 DTA 64bit 或 128bit 的数据。

DTA 将 DMA 操作的起始地址和操作的长度通过 dmemref 总线传输给 SPM，而 SPM 并不区分是否是本地 core 对它的访问还是远程 core 对它的访问，而只是将操作的结果返回给

DTA。再由 DTA 负责结果的传输。

（3）SPM 中维持数据一致性

在专业模式中，数据一致性完全靠程序员维持。

如对于如下的代码：

小核 C 对远程 SPM（小核 A 的 SPM）的变量 S 进行 store 操作：

Acquire()

Store(remote_A_SPM);　　①

Release()

同时，小核 B 对远程 SPM（小核 A 的 SPM）的变量 S 进行 load 操作：

Acquire()

Load(remote_B_SPM);　　②

Release()

由于执行①对 S 的 release 操作后，②中就可以对 S 进行 load 操作，但是此时 C 对 S 的 store 还没有传到，因此②中对 S 取的值即为旧值。

对于这种情况，可以有两种方法解决：一是，需要程序员将上面两段程序中的 load 和 store 指令写为 sync_load 和 sync_store，进行同步操作，保证新值传到；二是，在①的 store 操作后，增加一个等待 A 返回收到新值的 ack 信号，然后才能执行 release()操作，也能保证新值传到。

（在 GodsonT 的工程实现中，采用 sync_load 和 sync_store 操作。）

（4）SPM 接受远程 core 的访问请求

远程的 core，通过 router 将对远程 SPM 的请求发送给相应的 DTA，DTA 对其译码，并将操作的地址和长度传给 SPM，SPM 对其和本地的操作等同处理。SPM 只负责读取相应的数据，并将消息中的目的地址原样返回给 DTA，由 DTA 负责转发。

（5）SPM 的 full/empty 位

SPM 的 full/empty 位的作用是同步。若需要同步，core 对 SPM 进行 store 时，会将其 full/empty 位置为 1。而对 SPM 进行 load 时，需要判断 full/empty 位是否为 1，为 1 时可以返回 load 的值；为 0 时，流水线 stall，等到 store 写回。

SPM 的 full/empty 位只有 0 和 1 两种状态，full/empty 位的状态转换如下：

其中 sync_load 和 sync_store 即为带 full/empty 位的 load/store。

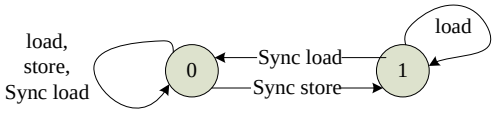


图 2.7　SPM full/empty 位的状态转换图

2.3 Global Memory 模块

Global Memory 目前仅作为 L2 cache，没有可编程的接口，但程序员可利用小核上的 DTA 产生读写 L2 cache 的请求。根据 DTA 发送给 L2 返回几个 128bit 数据包的信号，L2 可以返回给 DTA 128bit、256bit 和 512bit 的数据（回填 L1 Dcache 数据 256 bit；返回给 DTA 也是 128 bit；SPM 返回给 DTA 的数据改为 128bit）。

2.3.1 模块接口/总线接口

2.3.1.1 L2 模块的环境

L2 分为 16 个 bank，分布在芯片的四周，每个 bank 都连在 router 上，通过专用的 crossbar 与 memory controller 相连。如下图所示，图中为四个 L2 bank 与其他模块的连接图。

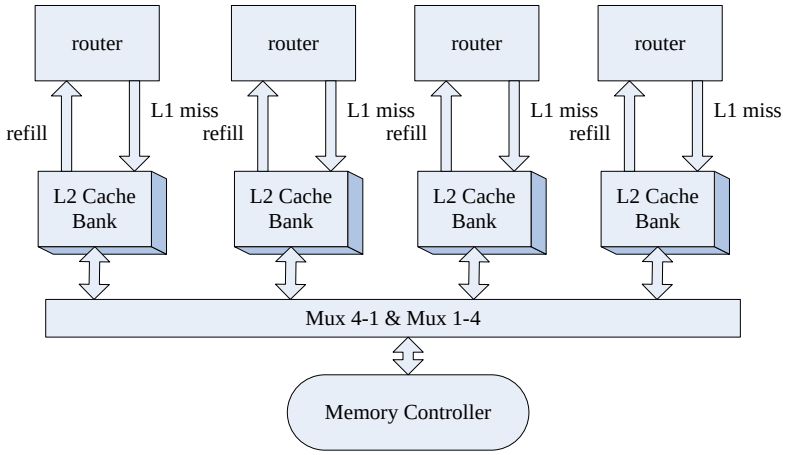


图 2.8　L2 bank 与其他模块的连接示意

L1 Dcache/Icache miss 通过 router 转发到相应的 L2 bank，如果 L2 hit，则 L2 将 refill data 封装成发往 router 的消息 L2_REFELL_TO_DCACHE/ICACHE，转发到 L1；如果发生 L2 miss，则通过专用的 crossbar 将 miss 请求直接发送到 memory controller，从 memory 中取数。

2.3.2 模块功能及设计

目前 GodsonT 的工程设计中，Global Memory 不设计为通过 DTA 和片下的 memory 传递数据，而是通过专用的 crossbar 将 memory controller 和固定的几个 L2 banks 建立硬连接。

2.3.2.1 L2 CACHE 工作原理

L1 Dcache/Icache 对 L2 的 miss 请求直接通过 router 发送到 L2；L2 对 L1 Dcache/Icache 的 refill 也直接通过 router 发送到 core。

（1）操作过程

若 L1 miss，并且 L2 miss。从 memory 读入 cacheline A，A 进入 L2 和 L1。L1 Dcache 中被 A 替换出的 B 进入 L2，L2 中被 A 替换出的 C 进入 memory(write back 模式)。B 进入 L2 时，可能会将原有的 D 覆盖，所以应先将 D 写回 memory，再将 B 写入 L2。

若 L1 miss，L2 hit。从 L2 中读取相应的 cache line A 进入 L1。L1 中被 A 替换出来的 B 进入 L2。B 进入 L2 时有可能将 L2 中原有的 C 覆盖，所以应先将 C 写回 memory，再将 B 写入 L2。

（2）L2 cache 的结构

L2 的总容量为 2MB，共分为 16 个 banks，每个 bank 的容量是 128KB，每个 bank 设计为 8 路组相联，每一路为 16KB，一个 cache line 为 512 bit（64 byte），则每一路有 256 个 cache line。

每个 bank 的 1 路 cache 如下图的结构。

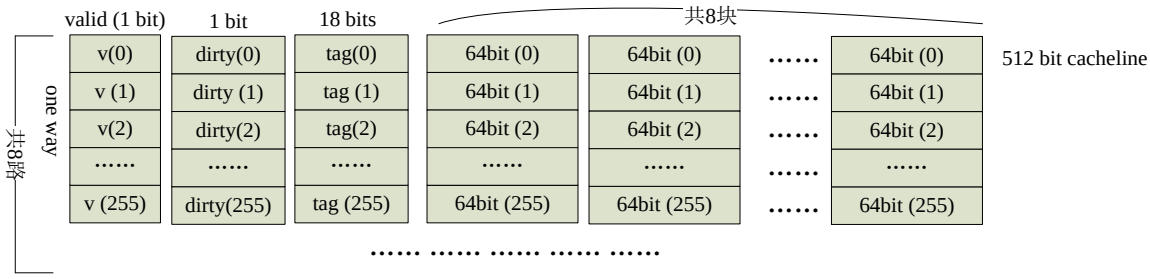


图 2.9　一路 L2 cache 的结构示意

（3）L2 cache 的 cache line

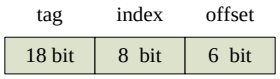
L2 cache line 为 512 bit，一次回填 256 bit（or 64 bit），分 2 次发 512 bit 的数据。

图 2.2　L2 cache line 结构

表 2-2　L2 cache line 各部分的含义

名称	宽度(bit)	描述
Valid	1	Cacheline 所处状态(invalid , valid)
Dirty	1	标志该 cache line 是否被修改过。1 表示修改过，替换时需要写回；0 表示未修改过，可以直接替换
Tag	18	该 cacheline 的 tag （物理地址高 18 位 ）
DATA[Nblocksize_d]	8×64	一个 cache line 的数据

（5）地址总线划分



（4）L2 cache 对 DTA 操作的支持

目前，GodsonT 的实现中，不在 L2 中设置 DTA 逻辑，而只设置 L2 变通处理 SPM 从 L2 中取大块数据的操作，即 L2 cache 接收 DTA 的标志位，表示返回几个 128bit 的数据，L2 可以返回 128bit、256bit 和 512 bit 的数据（对应 private SRAM 设置为 SPM 的专业模式）；若该位置为无效，则 L2 返回 256 bit 的数据（对应 private SRAM 设置为 L1 Dcache 的基本模式）。

L2 中无 DTA 逻辑，每次 DTA 对 L2 的 load 操作都返回 128 bit。

（5）L2 模块增加的原子指令

L2 增加支持两条原子指令的逻辑：FAA 和 test and set。

FAA（fetch and add）的操作数是地址[x]和寄存器 I，其功能是将给定的地址的变量加上寄存器的值，即[x]+I，并返回给结果寄存器；

Test_and_set 的格式为：test_and_set rd, rs, rt; 其中 rd 为目的操作数，rs 和 rt 是源操作数，rt 对应内存地址，其语义为：

```
if [rt] = = 0
   then
   {
      [rt] = rs;
      Rd = 1;
```

}

else

rd = 0;

FAA 需要在 L2 中增加加法器逻辑；test and set 需要在 L2 中增加比较器逻辑。——二者不能互相代替，暂时先都实现。

（6）替换策略

采用 LRU 替换策略。

2.3.2.2 L2 CACHE 对 outstanding miss 的支持

L2 cache 设计为支持 outstanding miss，即将 L2 设计为 non-blocking 的结构。在 L2 controller 中设置 miss queue，用于记录 L2 正在处理的 L1 miss。在发生 L1 miss L2 miss 的情况下，如果将 L2 设计为 blocking 的结构，则需要等 L2 miss 的重填数据从 memory 返回并重填 L1 后，L2 才能继续接收其他来自 L1 的 miss 请求，这种情况对于前后两次 L1 miss 中第一次 L1 miss 也导致 L2 miss，但第二次 L1 miss 能够发生 L2 hit 的情况不利，增大了第二次 L1 miss 的处理延迟；将 L2 设计为 non-blocking 的结构，即使前一次的 L1 miss 导致了 L2 miss，需要从 memory 取数，但只要 L2 的 miss queue 未满，L2 就可以继续接收来自 L1 的 miss 请求，这就减少了后续 L1 miss L2 hit 操作的访存延迟。

L2 cache 模块和其他模块的连接图如下所示。

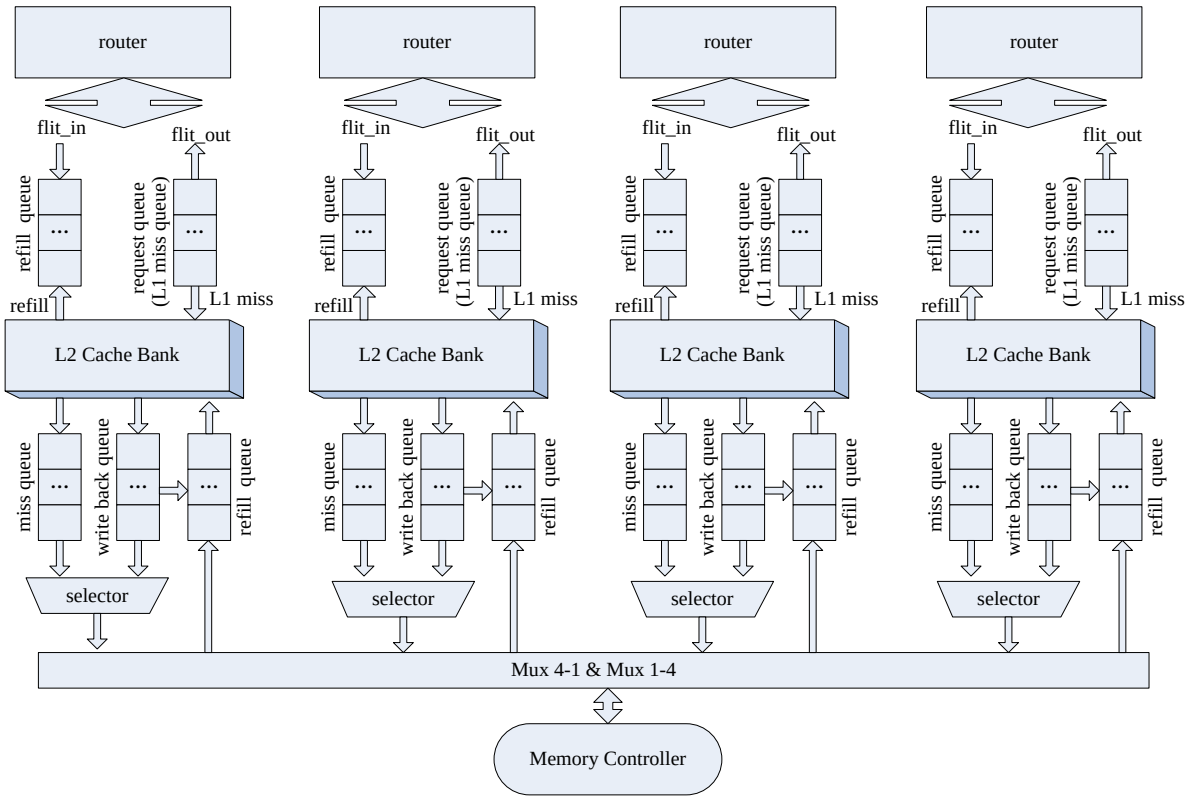


图 2.3 支持 outstanding miss 的 L2 bank 逻辑图

上图中，L2 bank 与 memory controller 之间连接的 miss queue 与 write back queue 之间对 memory controller 的访问仲裁结果是，始终都使得 miss queue 优先级高于 write back queue，并且 miss 可以查找 write back queue，保证能够命中新值。

（1）request queue 的设计

L1 Icache/Dcache miss 和 L1 Dcache store 会进入 L2 的 request queue 中。L2 的 request queue 的初始状态是空状态；

区分进入 request queue 中待处理的请求的类型，

如果是 L1 miss，则进入第二个状态 L1_miss，再此状态中需要查 request queue，看是否有命中的 L1 Dcache store；

如果在 request queue 中 hit，则进入第三个状态 RQ_refill，即将 request queue 中命中的数据直接对 L1 Dcache miss 进行 refill，并在下一拍进入初始状态；

如果在 request queue 中 miss，则进入第四个状态 Lookup_L2，并在此状态中查 L2

如果发生 L2 hit，则进入第五个状态 L2_hit，并在此状态内写 refill queue，在下一拍返回初始空状态；

如果发生 L2 miss，则进入第六个状态 L2_miss，并在此状态等待 memory refill to L2 完成，将 wait_memory_refill 置为无效，并在此信号不变的情况下，一直在 L2_miss 状态循环；如果 wait_memory_refill 变为有效，则写 refill queue，并在下一拍返回初始状态；

如果是 L1 store，进入第七个状态 L1_store 状态，首先查 request queue 是否 hit，

如果 hit，则表明有 WAW 相关，因此第二个 store 不能执行，必须等待第一个写完后才能继续操作，因此一直在 L1_store 状态循环；

如果 miss，则进入第四个状态 lookup_L2，

如果发生 L2 hit，进入第九个状态 Store_L2，此状态内可能会发生 L2_replace 和 L2_write_back 等，并在处理完毕后返回初始状态；

如果发生 L2 miss，则进入状态 L2_miss，并将 wait_memory_refill 置为无效，并在此信号不变的情况下，一直在 L2_miss 状态循环；如果 wait_memory_refill 变为有效，则写 refill queue，并进入 Store_L2 状态。

Request queue 的状态转换图如下图所示。

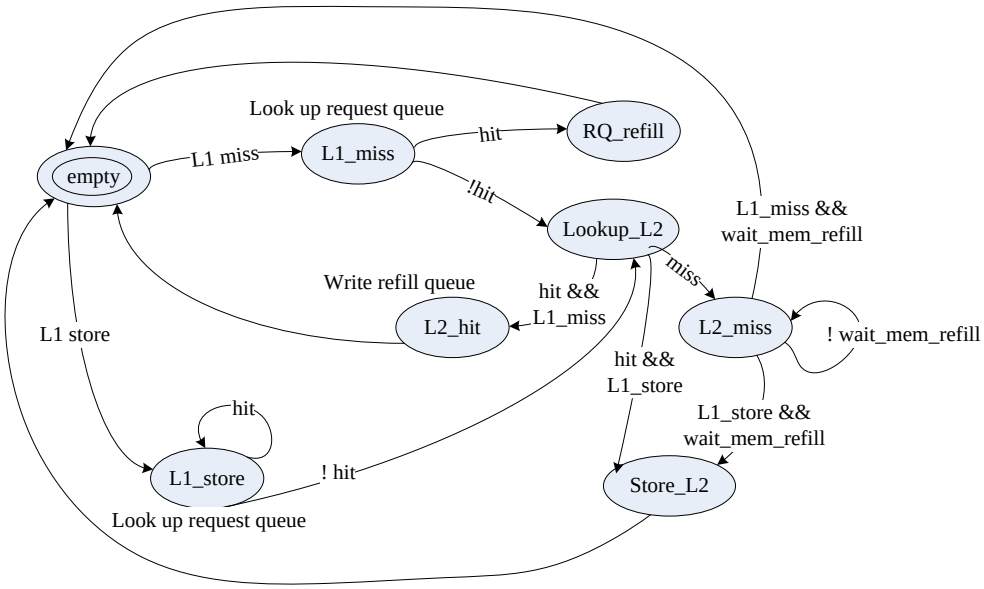
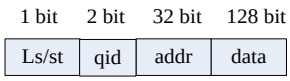


图 2.4　request queue 的状态转换图

Request queue 设计为 4 项，每一项的逻辑结构：



（参考 godsonx0

Request queue 有到 refill queue 的数据通路，当发生一个 load 在 request queue 中 hit 时，直接将数据从 request queue 传输到 refill queue（修改）。

（2）miss queue 的设计

（项数如何确定？——根据流水线的拍数确定）

（项数、逻辑结构、各个 queue 的项数之间的关系、状态位、表、queue 之间交互的信号）

（在 L2 的 miss queue 中需要加入判断两项是否有数据相关的逻辑）

Miss queue 的初始状态为 empty，等待 L2 发出的新的 L2 miss request（包括 load miss 和 store miss）；

如果接收到一个新的 miss 请求，进入第二个状态 enter，在此状态查找 write back queue，并返回是否命中的消息；

如果 write back queue 发生 miss，则进入第三个状态 mem_access，等待 memory refill L2，并置 wait_mem_refill 为无效，如果该信号无效，一直在此状态循环；如果 wait_mem_refill 信号有效，则进入第四个状态 RQ_wait，准备从 write back queue 中将新数据传输到 refill queue，并在此状态等待 refill queue 能否接收写操作的信号；

如果 write back queue 发生 hit，进入第四个状态 RQ_wait；

如果 refill queue 不可以接收，则在 RQ_wait 状态循环，一直到 write back queue 可以接收为止；

如果 refill queue 可以接收，则进入第五个状态 RQ_write，在此状态将 write back queue 中的新数据写入 refill queue；

如果是 load_miss，则在 RQ_write 状态进入初始状态 empty，等待下一次 L2 miss；

如果是 store_miss，则在 RQ_write 状态进入第六个状态 store，将数据 refill 到 L2，并在下一拍进入初始状态。

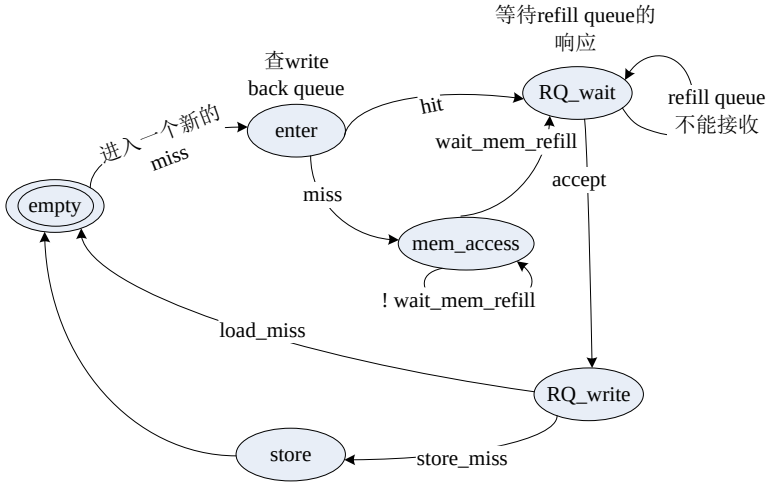
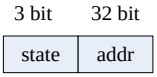


图 2.5　miss queue 的状态转换图

Miss queue 设计为？？项，每一项的逻辑结构为：



Miss queue 的状态有四个：查 write back queue、RQ_wait、mem_access、RQ_write，用两位可以表示四种状态。

在物理实现中，可以将 request queue（即 L1 miss queue）与（L2）miss queue 合并，用状态位区分二者：发生 L1 miss 后，将 miss queue 中对应的状态位置为 L1 miss；若再发生 L2 hit，则直接返回数据；若再发生 L2 miss，则将该项的状态位置为 L2 miss，再访问 memory。

因此，将 miss queue 的状态位设计委 3 位。

（3）write back queue 的设计

L2 将 dirty 位为 1 的 cache line 写回到 memory 的缓存队列。(请求在 write back queue 中 hit，是否将其写回 L2，要写回 L1)

（4）refill queue 的设计

（5）支持 out-standing miss 后 L2 对 cache 一致性的影响

从 request queue 中缓存的几项访问 L2 的操作考虑：

如果 request queue 中对 L2 的访问没有任何的数据相关，可以将 request queue 中各项按任意顺序执行；

如果 request queue 中有一项对 L2 中某 cache line A 进行 store，位于其后的一项对 A 进行 load，则直接从 request queue 中 bypass 结果；

如果 request queue 中前一项对 L2 中某 cache line A 进行 store，后一项对 A 也进行 store，并且二者之间没有对 A 进行 load 的操作，可以将二者合并执行；

如果 request queue 中前一项对 L2 中某 cache line A 进行 load，而后一项对 A 进行 store，

则二者不能进行 outstanding 处理，只能顺序处理。

2.3.2.3 L2 CACHE 的状态转换图

L2 cache 的状态机：L2 cache 是一个被动的模块，初始状态为等待 L1 miss，发生 L1 miss 后，将数据返回给 L1 进行 refill，或者从 memory 中取数。

L2 cache 的状态转换图如下图所示。

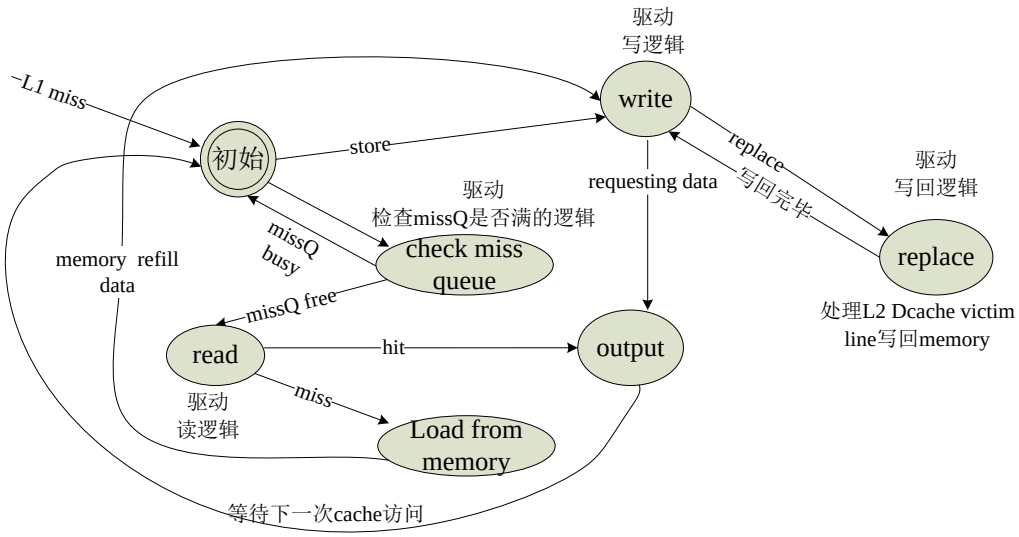


图 2.6　L2 状态转换图

2.4 Selector 模块

Crossbar 是四个 L2 bank 和一个 memory controller 之间互连的专用接口，相当于 L2 和 memory controller 之间的选路逻辑。数据传输总线宽度为 512 bit（因为 L2 的 cache line 是 512 bit）。

（将 L2 miss queue 和 write back queue 修改为和 memory controller 交互的接口）

（1）L2 banks 对 memory controller 的访问：crossbar 对于多个 L2 banks 的同时访问，仲裁成功的一个发送到 memory controller；其他没有仲裁到的 L2 banks 等待？？cycles 继续发送访存请求。

如下图所示。

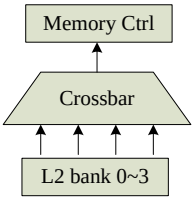


图 2.7　Crossbar 将 L2 banks 的请求转发到 memory controller

（2）memory controller 对 L2 cache refill：crossbar 相当于一个选路逻辑，将 refill data 转发到相应的 L2 bank。

如下图所示。

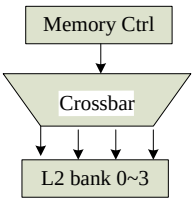


图 2.8　Crossbar 将 memory controller 的输出返回给相应的 L2 bank

2.4.2 模块功能及设计

Crossbar 的功能：在与之相连的四个 L2 banks 和 memory controller 之间快速传递数据。

L2 cache banks 为 interleave 编址，一个地址对应到一个 L2 bank（称该 bank 为该数据块的 home bank）。Router 根据 core 访问的数据地址解析出其 L2 home bank，并转发至相应的 router。若在此 home bank 中 hit，则将数据返回给 router，转发至发出请求的 core；若在其 home bank 中 miss，则将此 L2 miss 通过 crossbar 发送给其端口 0，即 memory controller，从 memory 中取数。

Crossbar 每个 port 都包括 input 端（请求）和 output 端（被请求）；其中 input 端在接到与这个端口相连的部件（L2 banks, memory controller）的消息后，会解析消息的包头地址，然后向相应的出口请求。Output 端会在端口空闲状态查询当拍对该出口的所有请求，仲裁最后胜出的 request_i,使请求端与输入端的连接总线有效，传输消息内容，直到消息传递结束，释放端口，接受下一个请求。

（1）Crossbar 有 5 个端口，分别为四个 L2 bank 的输出/输入缓冲区和 Memory Controller 的输出/输入缓冲区，如下：

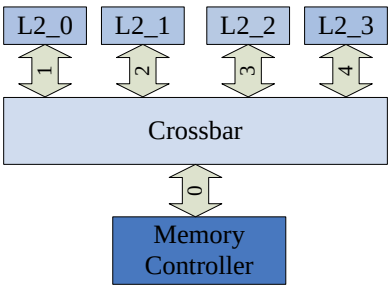


图 2.17　Crossbar 与 memory controller 和 L2 banks 的端口设置

（2）使用 3 位表示不同的端口，端口 ID 如上图所示，端口 ID 表示及对应关系如下。

2	1	0
P_id	P_id	P_id

Port_id 为 0~4，如上图对应关系，即：

① port_0：Memory Controller

② port_1：L2 bank 0

③ port_2：L2 bank 1

④ port_3：L2 bank 2

⑤ port_4：L2 bank 3

（4）Crossbar 内部 buffer 和控制信号

名称	说明	备注
L20_req_list[4:0]	value[i]=1;port_i 有请求；	记录当拍发往 L2 bank 0 端

	value[i]=0;port_i 无请求；	口的请求， 每一位对应一个请求端的 port 号
L21_req_list[4:0]	value[i]=1;port_i 有请求； value[i]=0;port_i 无请求；	记录当拍发往 L2 bank 1 端口的请求， 每一位对应一个请求端的 port 号
L22_req_list[4:0]	value[i]=1;port_i 有请求； value[i]=0;port_i 无请求；	记录当拍发往 L2 bank 2 端口的请求， 每一位对应一个请求端的 port 号
L23_req_list[4:0]	value[i]=1;port_i 有请求； value[i]=0;port_i 无请求；	记录当拍发往 L2 bank 3 端口的请求， 每一位对应一个请求端的 port 号
Memctrl_req_list[4:0]	value[i]=1;port_i 有请求； value[i]=0;port_i 无请求；	记录当拍发往 memory controller 端口的请求， 每一位对应一个请求端的 port 号
L20_outport_DataBuffer	储存仲裁后允许发往 L2 bank 0 的请求的数据	
L20_buffer_state	Value=1:表示 buffer 被占用，该端口不能接受下一个请求 Value=0:表示 buffer 空，可接受请求	
L21_outport_DataBuffer	储存仲裁后允许发往 L2 bank 1 的请求的数据	
L21_buffer_state	Value=1:表示 buffer 被占用，该端口不能接受下一个请求 Value=0:表示 buffer 空，可接受请求	
L22_outport_DataBuffer	储存仲裁后允许发往 L2 bank 2 的请求的数据	
L22_buffer_state	Value=1:表示 buffer 被占用，该端口不能接受下一个请求 Value=0:表示 buffer 空，可接受请求	
L23_outport_DataBuffer	储存仲裁后允许发往 L2 bank 3 的请求的数据	
L23_buffer_state	Value=1:表示 buffer 被占用，该端口不能接受下一个请求 Value=0:表示 buffer 空，可接受请求	
Memctrl_outport_DataBuffer	储存仲裁后允许发往 Memctrl 的请求的数据	
Memctrl_buffer_state	Value=1:表示 buffer 被占用，该端口不能接受下一个请求 Value=0:表示 buffer 空，可接受请求	

表 2-3　crossbar 内部 buffer 和控制信号

（5）Crossbar 内部总线

每一个端口都连有一组与其他端口连接的总线，如端口 0（memory controller）与其他各个端口的总线如下表。其中，每个端口的情况相同，命名方式为 Bus_A_B，A 为请求方端口号，B 为被请求方端口号；每个 clock，在同一个端口只可能有一条数据总线被置有效。

表 2-4　crossbar 内部总线

BUS 名称	说明
Bus_1_0[64:0]	连接 memory controller 与 L2 bank 0 的数据 bus;最低位为有效位，64~1 位为数据位
Bus_2_0[64:0]	连接 memory controller 与 L2 bank 1 数据 bus;最低位为有效位，64~1 位为数据位
Bus_3_0[64:0]	连接 memory controller 与 L2 bank 2 数据 bus;最低位为有效位，64~1 位为数据位
Bus_4_0[64:0]	连接 memory controller 与 L2 bank 3 数据 bus;最低位为有效位，64~1 位为数据位

2.4.3 片上消息

Crossbar 是 L2 banks 和 memory controller 之间的专用连接，不需要通过消息交互，直接通过硬连接通信即可。