Cache设计说明

1. 设计考虑点

1、数据与指令访问并行处理：

此Cache使用在流水线CPU设计中进行数据以及指令的存取以及修改。

流水线大体分取指、译码、执行、访存、写回五个阶段，对于指令的访问在第一个阶段，即取指阶段，对于数据的访问在第四个阶段，即访存阶段。流水线的一大特征是不同的阶段是并行进行的，取指的同时也会进行数据的访问。

因此需要对于一个周期中同时到达的数据访问请求以及指令访问请求进行处理，并在一定时间内将数据以及指令请求处理完毕，并返回response信号，以指示CPU数据或者指令访问请求处理完毕，CPU可以继续执行。

2、I$miss且D$在处理请求的情况

此时I$下一步，应该读D$以及RAM，以查找所需的指令。但是D$是在占用状态，D$正在处理来自Core的请求，因此应当先处理完D$的的任务，Response之后，再处理I$miss的情况，之前I$需要等待D$直到D$处于空闲状态。

3、访问处理速率

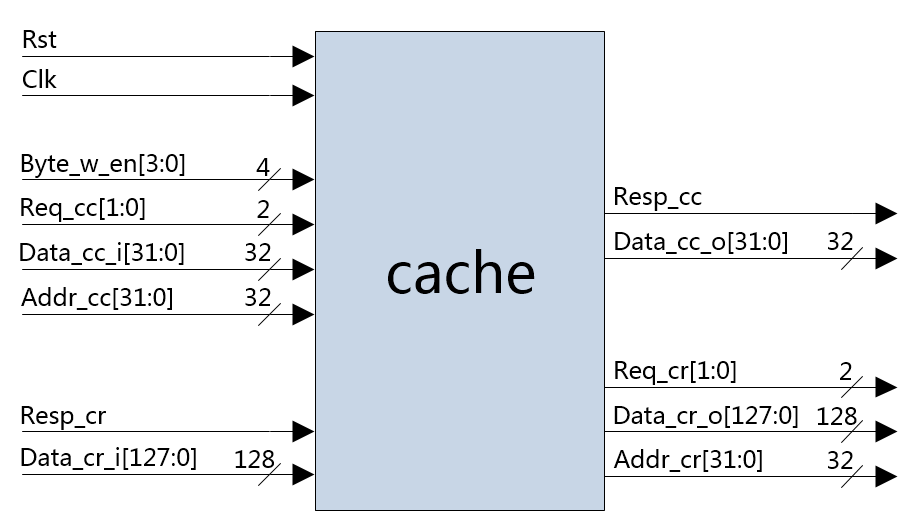
由于我们设计的是一级Cache，因此其命中率较高，大概有80%左右的命中率，剩余20%情况均摊到miss或者miss且dirty的情况。而我们使用的是core等待cache响应的机制，因此要提高cpu的速率，应考虑尽量缩短core对于cache进行访问的时间。

首要先提高命中时的时间花销，此处我们设计使用了状态机转换，因此命中固定的花销为3个时钟，要想进一步提高，则需要用一些方法，比如clk高电平做一些事，低电平做另一些事，应当可以提高一定的速率。而目前进使用状态机机制应当无法进一步提高。

其他情况由于只占到20%左右，由于只是对cache的初步实现，此处可以几乎忽略掉。

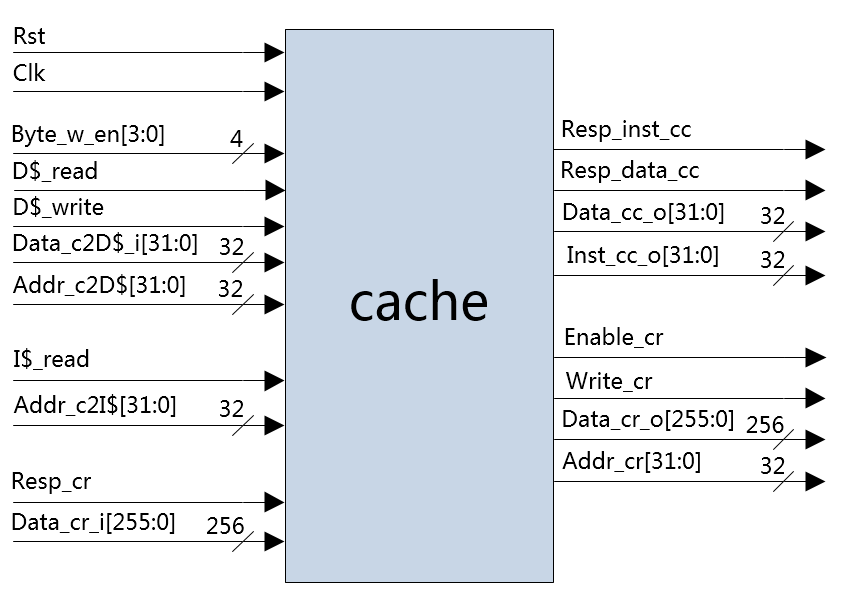
1. 顶层Cache设计
2. 设计问题点

第一次设计没有考虑到指令和数据的并行访问请求，同时由于Cache大小会决定功耗，对于电路进行了一次精简，产生了一个问题，即一次只能处理一个请求，数据以及指令请求到达只能每次处理一个。需要再在外侧封装一层控制器，两个请求命中时间，若是指令请求为n1，数据情况为n2，那么原本需要max(n1, n2)的变成了n1+n2，导致CPU的等待时间会较长，且无法发挥将Cache分为I$和D$的特点。产生线路如下：



考虑指令数据请求并行处理，添加一部分使Data与Inst请求并行的线路，同时D$中相对Write添加Read信号，而不是精简到只有Write信号是为了使其本身有使能的作用（1时有效，0无效），若单独添加使能信号也可以，但是为了与Inst请求对称，且不多添加线路便如此设计，最终得到的框图如下。

2、实际电路框图：



3、框图端口说明：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 另一端接口部件 | 作 用 |
| 1 | Rst | 1 | 输入 |  | 复位信号 |
| 2 | Clk | 1 | 输入 |  | 时钟 |
| 3 | Byte\_w\_en | 4 | 输入 | Core | 由于MIPS指令中有对单字节读写功能，添加对数据字节写使能信号 |
| 4 | D$\_read | 1 | 输入 | Core | 为1则Data Cache读，为0无效 |
| 5 | D$\_write | 1 | 输入 | Core | 为1则Data Cache写，为0无效 |
| 6 | Data\_c2D$\_i | 32 | 输入 | Core | 由Core写到Data Cache的数据(Data\_core\_to\_data\_cache) |
| 7 | Addr\_c2D$ | 32 | 输入 | Core | Core读或写Data Cache数据的地址 |
| 8 | I$\_read | 1 | 输入 | Core | 为1则Instruction Cache读，为0无效 |
| 9 | Addr\_c2I$ | 32 | 输入 | Core | Core读Instruction Cache的指令的地址 |
| 10 | Resp\_cr | 1 | 输入 | RAM | RAM读写请求Response，表示请求处理完毕 |
| 11 | Data\_cr\_i | 256 | 输入 | RAM | 从RAM读到Cache的数据 |
| 12 | Resp\_inst\_cc | 1 | 输出 | Core | Core对Cache的指令读请求处理完毕信号，为1则完毕 |
| 13 | Resp\_data\_cc | 1 | 输出 | Core | Core对Cache的数据读写请求处理完毕信号，为1则完毕 |
| 14 | Data\_cc\_o | 32 | 输出 | Core | Core读Cache数据请求该地址读出的数据 |
| 15 | Inst\_cc\_o | 32 | 输出 | Core | Core读Cache指令请求该地址读出的指令 |
| 16 | Enable\_cr | 1 | 输出 | RAM | RAM的使能信号 |
| 17 | Write\_cr | 1 | 输出 | RAM | 1则Cache向RAM写，0则Cache从RAM读 |
| 18 | Data\_cr\_o | 256 | 输出 | RAM | Cache向RAM写回的数据 |
| 19 | Addr\_cr | 32 | 输出 | RAM | Cache向RAM写回数据的地址 |

1. Cache内部设计

1、设计问题点

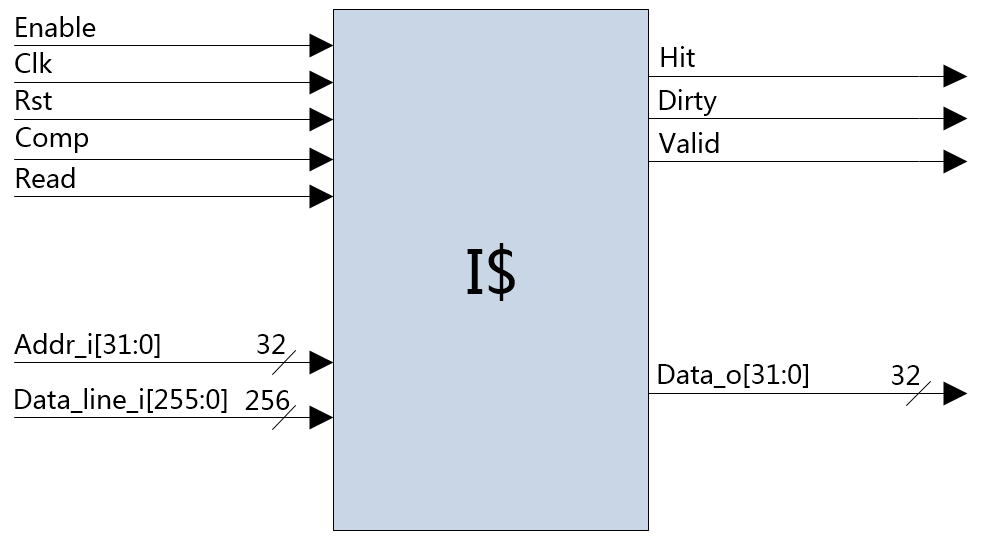
若出现I$ miss情况需要向D$发送读请求，此时应当考虑边界情况，即D$在读或写时，同时出现了I$ miss的状况。则此时交由控制器处理，先对于D$的读写状况（无论是miss抑或存在dirty）进行处理，处理完之后（此处的处理完是指D$的response信号传到了Core，并使用完的下一个周期，否则D$的hit信号与address信号会一直被D$的读写请求占用，无法处理I$读D$的情况）。控制器转为对于i$ miss的情况进行处理即可。

1. I$的设计

⑴、考虑因素：

指令cache来自core的访问只有读请求，而当miss的时候会从数据cache或者RAM读取一行数据，并进行写入。因此输入数据需要一个选择器用以选择是接收来自数据cache的数据还是来自RAM的数据。

⑵、电路框图



⑶、端口说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | Enable | 1 | 输入 | 使能信号，用于指示其余输入信号是否有效 |
| 2 | Clk | 1 | 输入 | 时钟 |
| 3 | Rst | 1 | 输入 | 复位信号 |
| 4 | Comp | 1 | 输入 | 为1则比较每行中的tag与输入地址相应字段，否则不比较 |
| 5 | Read | 1 | 输入 | 为1则读I$，I$输出指令，为0则向I$写入指令 |
| 6 | Addr\_i | 32 | 输入 | Core要读I$的指令的地址 |
| 7 | Data\_line\_i | 256 | 输入 | D$或者RAM读来的指令串，需要写入I$ |
| 8 | Hit | 1 | 输出 | 为1则cache命中，输出正确的指令 |
| 9 | Dirty | 1 | 输出 | 为1则该行是修改过，为0未修改，I$该输出无意义 |
| 10 | Valid | 1 | 输出 | 为1则该行有效，为0无效，目前没有用，用于之后拓展 |
| 11 | Data\_o | 32 | 输出 | 从I$读出的指令 |

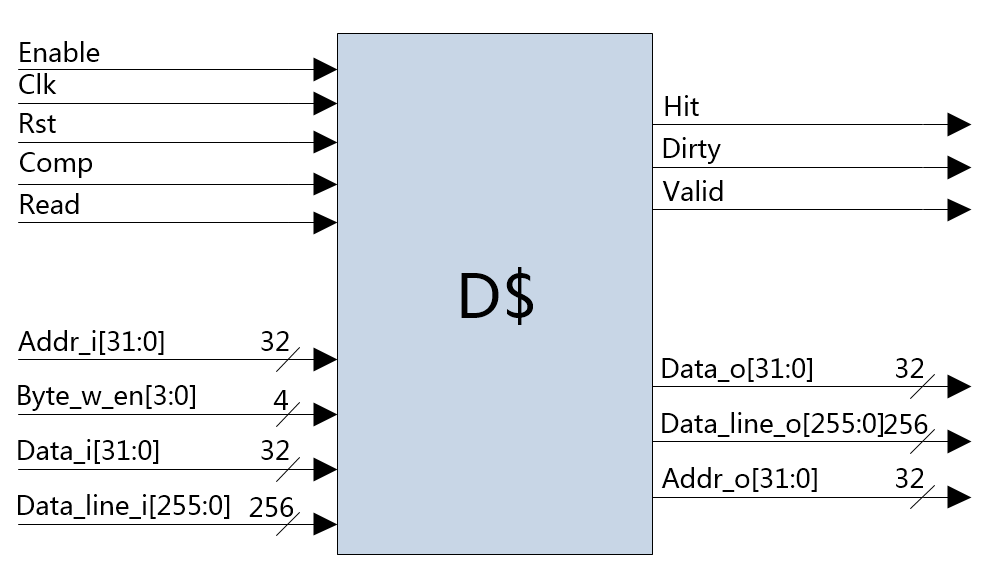
1. D$的设计

⑴、考虑因素

对于D$的访问可能来自I$也可能来自Core，不过此问题无需D$进行管理，交给控制器管理即可。

由于MIPS指令集中存在对于一个字节的修改以及加载的指令，Cache需要对于此功能进行支持，因此需要一个byte\_w\_en，字节的写使能信号进行对字节的操作，而此信号由cpu发出。

⑵、电路框图



⑵、端口说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | Enable | 1 | 输入 | 使能信号，用于指示其余输入信号是否有效 |
| 2 | Clk | 1 | 输入 | 时钟 |
| 3 | Rst | 1 | 输入 | 复位信号 |
| 4 | Comp | 1 | 输入 | 为1则比较每行中的tag与输入地址相应字段，否则不比较 |
| 5 | Read | 1 | 输入 | 为1则读D$，D$输出数据，为0则向D$写入数据 |
| 6 | Addr\_i | 32 | 输入 | Core要访问D$的数据的地址 |
| 7 | Byte\_w\_en | 4 | 输入 | 字节写使能信号，用以支持字节的访问需求 |
| 8 | Data\_in | 32 | 输入 | 需要写入的数据 |
| 9 | Data\_line\_i | 256 | 输入 | RAM读来的数据块，需要写入D$ |
| 10 | Hit | 1 | 输出 | 为1则cache命中，输出正确的数据 |
| 11 | Dirty | 1 | 输出 | 为1则该行是修改过，为0未修改 |
| 12 | Valid | 1 | 输出 | 为1则该行有效，为0无效，目前没有用，用于之后拓展 |
| 13 | Data\_o | 32 | 输出 | 从D$读出的数据，传到Core |
| 14 | Data\_line\_o | 256 | 输出 | 从D$读出的数据行，传到I$或者RAM，用以写入，32位数据与行数据同时输出 |
| 15 | Addr\_o | 32 | 输出 | 要写回RAM的数据行地址 |

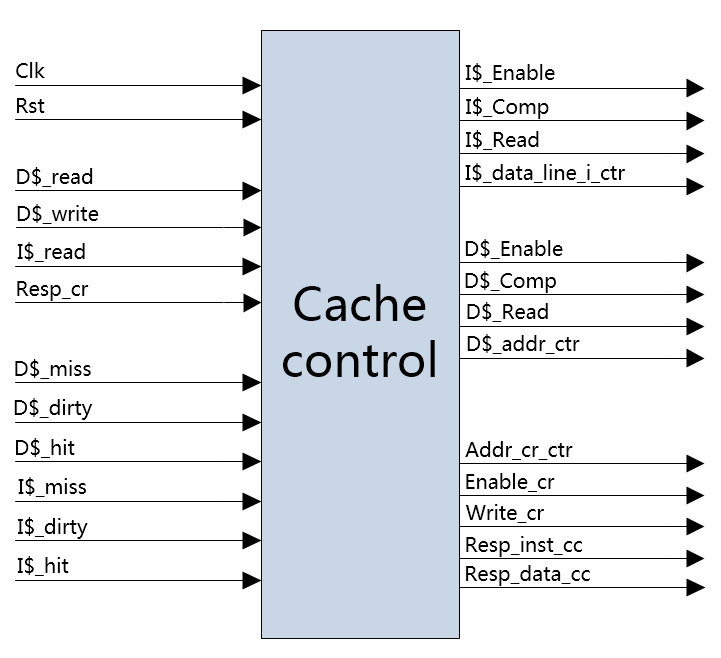
1. Cache控制器的设计

⑴、考虑因素

用以控制I$与D$的同步，且与Core和RAM的交互，实现正确的数据流动。

因此除了普通的对于两个cache的控制信号外，需要管理I$的输入由RAM来的指令或者D$来的指令，D$需要处理的是I$的请求地址还是Core的请求地址，向RAM发送读写请求的地址是I$还是D$的地址，以及一些响应信号。

⑵、电路框图



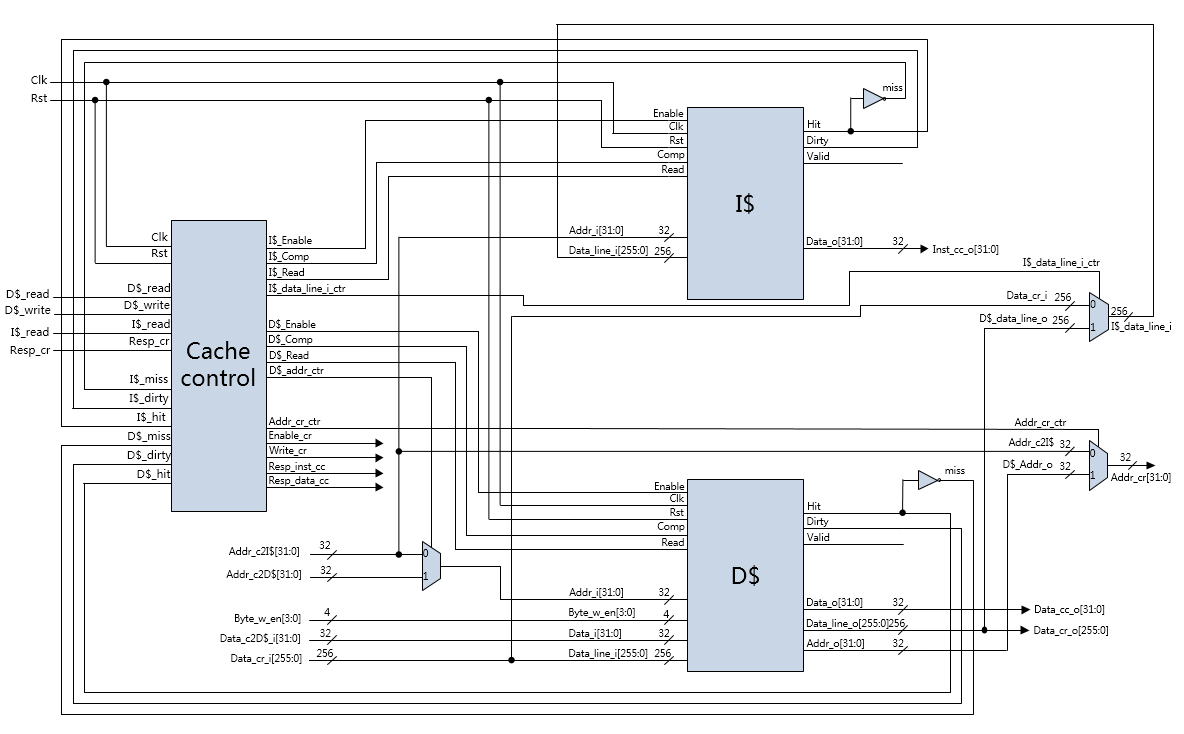
⑶、部分端口说明

由于信号在电路图功能中较为明显，此处仅说明一些较为特殊的信号。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作 用 |
| 1 | Resp\_cr | 1 | 输入 | 来自RAM的响应信号，Response\_cache\_ram |
| 2 | I$\_data\_line\_i\_ctr | 1 | 输出 | 选择写入I$的指令来自D$还是RAM |
| 3 | D$\_addr\_ctr | 1 | 输出 | 选择D$的访问数据地址是I$的还是D$的 |
| 4 | Addr\_cr\_ctr | 1 | 输出 | 选择输出到RAM的地址是来自I$还是D$ |
| 5 | Resp\_inst\_cc | 1 | 输出 | 响应Core的指令请求信号，访问I$ |
| 6 | Resp\_data\_cc | 1 | 输出 | 响应Core的数据请求信号，访问D$ |

5、Cache内部连线

根据以上的分析以及两个Cache以及控制器的设计进行电路图的连接，结果如下：



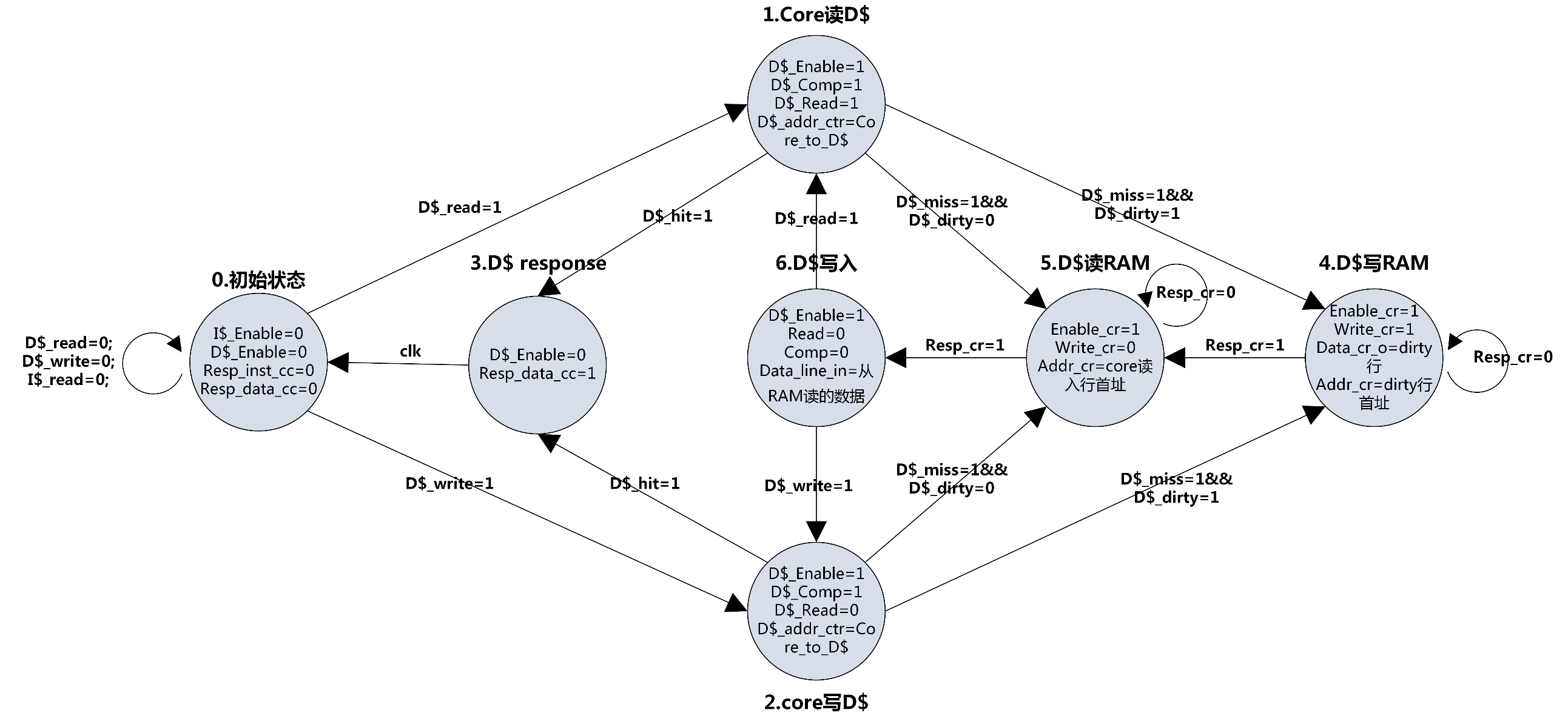
四、状态图转换设计

I$与D$需要进行并行处理，若仅适用一个状态转换图，那么则为串行的结构，无法达到预期的并行处理的设计。因此，此处将原本设计出的串行处理的状态图一分为二，一部分为D$的处理，另一部分为I$的处理，同时由于需要进行I$对于D$的访问实现，需要在I$的处理中对一些特定的条件进行判断（D$处理完毕才能进行来自I$访问的处理）。

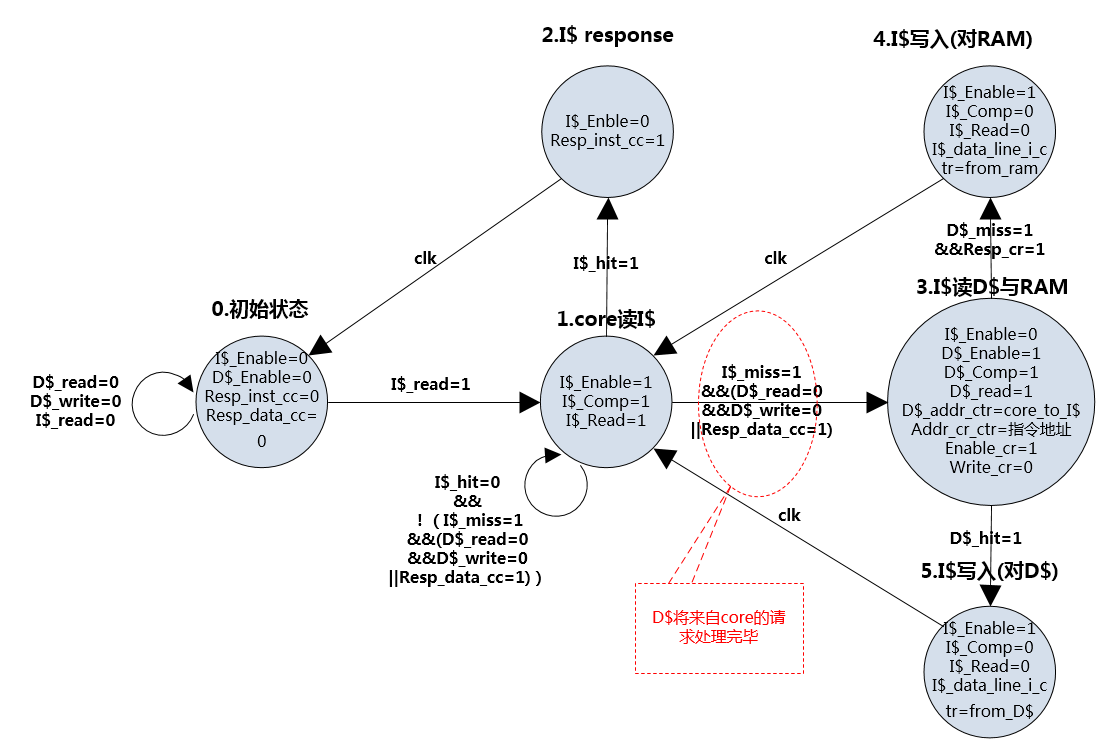
而实际只有一个控制器，因此需要在该控制器中实现两个状态图的并行转换。

实现的转换图如下：

1、D$的状态转换图



2、I$的状态转换图



五、信号值输出表

由于使用了两个状态转换图，因此，输出的控制信号会有重叠，但是不会产生冲突，因此，控制器需要对两个转换图输出的信号值进行一个“与”操作，此处我们约定将无所谓的信号值全赋值为0，有效的信号为1，无效的信号为0。则这样可以得到最终的信号输出。

我们将两个状态转换图的每个状态所产生的信号值在此处一一列举，表格如下：

D$信号表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态\信号值 | I$\_Enable | I$\_Comp | I$\_Read | I$\_data\_line\_i\_ctr | D$\_Enable | D$\_Comp | D$\_Read | D$\_addr\_ctr | Addr\_cr\_ctr | Enable\_cr | Write\_cr | Resp\_inst\_cc | Resp\_data\_cc |
| 0.初始状态 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1.core读D$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2.core写D$ | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3.D$response | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 4.D$写RAM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 5.D$读RAM | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 6.D$写入 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

I$信号表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态\信号值 | I$\_Enable | I$\_Comp | I$\_Read | I$\_data\_line\_i\_ctr | D$\_Enable | D$\_Comp | D$\_Read | D$\_addr\_ctr | Addr\_cr\_ctr | Enable\_cr | Write\_cr | Resp\_inst\_cc | Resp\_data\_cc |
| 0.初始状态 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1.core读I$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2.I$ response | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3.I$读D$与RAM | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 4.I$写入(对RAM) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5.I$写入(对D$) | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

注：D$信号表中从D$写RAM到D$读RAM，需要将D$\_Addr\_o由dirty行首址转换为读入行首址，转换根据D$的信号不同之处来进行相应的操作。