目录

一、实验目的和内容

1.1实验目的

通过设计一个一级数据 cache 控制器增强对 cache 结构和机制的掌握。

1.2 cache 参数和机制

1. 一级数据 cache 的参数说明说下。

A Simplified Model of TMS320C621x/C671x DSP	
Internal memory structure	Two Level
L1D size	4Kbytes
L1D organization	2-way set associative
L1D line size	32 bytes
L1D replacement strategy	Least Recently Used (LRU)
L1D read miss action	1 line allocates in L1D
L1D read hit action	Data read from L1D
L1D write miss action	No allocation in L1D, data sent to L2
L1D write hit action	Data updated in L1D, line marked dirty
L2 line size	128 bytes
L2 → L1D read path width	128 bit
L1D → L2 write path width	32 bit

图 1 Cache 参数

注：以下对该一级数据 cache 简称 L1D。

2. 本次实验假定 write buffer 无限大；
3. 不考虑 TLB；
4. 同一时间只允许一次存或取操作。

1.3 实验步骤

1. 状态机设计：
   (1) 画出该控制器的状态图，进行状态编码，定义状态转换条件；
   (2) 根据状态图和输出函数，画出状态转移图；
   (3) 画出流程图；
2. 基于以上设计，用计数器或者多个选择器画出逻辑电路。
3. Verilog HDL 语言实现该电路。

二、Cache 原理

该部分描述了 L1D 的一些处理机制，同时对控制器信号进行说明。

2.1 Cache 访问

2.1.1 地址分配

由 1.2.1 中 L1D 的参数：4K-byte cache size、 two-way set associative 、32-byte line size、64 sets.可以得知 L1D 的地址分配如图 1：

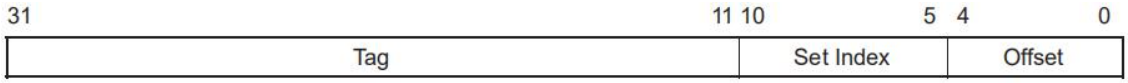

图 2 L1D Address Allocation

2.1.2 cache 访问

图 3 是一个 Read 访问时的简单示意图，需要执行 Read tags 和 Read data，本次实验将两个操作放在分离状态中，而不是并行执行，可以改进时钟周期。

对于写访问，没有下图的 read data 部分，只有 tag 比较，输出 hit/miss 信号。

注：对于其中数据输出的方式，图中采用了带有译码选择信号的 2 选 1 多路选择器，也可以采用其他方式选择数据，对本实验控制器设计影响不大。

图中未画出 Dirty bit、Reference bit、write enable 标志。

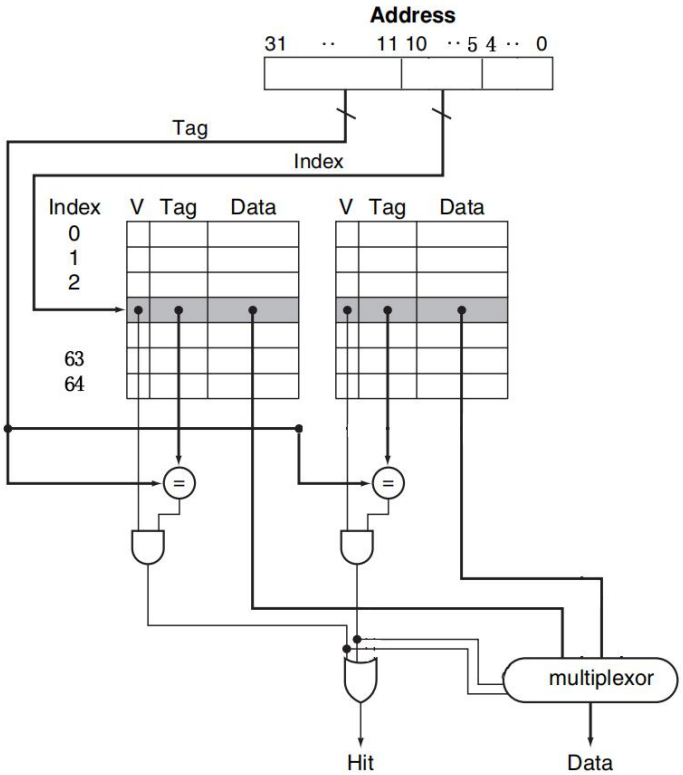

图 3 L1D 读访问

2.2 读/写操作处理

2.2.1 L1D 读写处理过程

根据 L1D 的各个读写操作机制（图 1）画出其处理过程的简单流程图如图 4。（这不是实际写 Verilog 的流程图。在实际情况，write buffer 大小有限，read miss 时还需要考虑由于 write buffer 为空导致的 stall，下文会详述。在 Verilog 实现中假设 write buffer 大小是无限大的）：

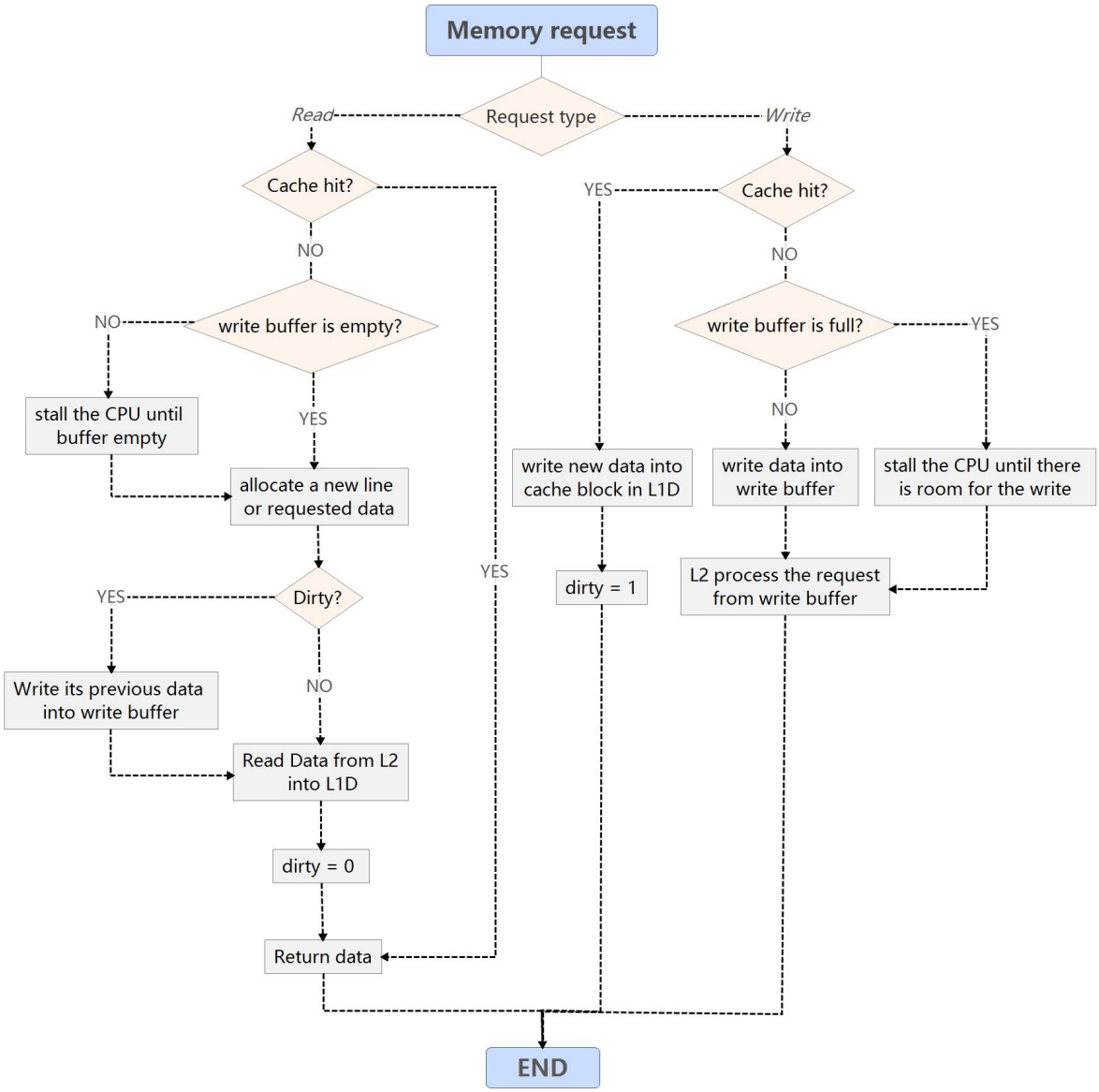

图 4　L1D 读写处理机制流程图

2.2.2 缺失处理

由于存在 write buffer，如果 write miss，数据都直接写入 write buffer，再送给 L2 后续处理。

当 read miss 时候还要考虑 write buffer 是否 empty。写缓冲器中的数据在没有被写入主存之前，是不能被读取的。同样，被替换的 cache 行在写缓冲器中时也

不能进行读操作。如果读的内容正好是 write buffer 里面即将被写回的 dirty data，不等 write buffer 处理完毕就处理 read miss，就会从 L2 取到未更新的旧数据，程序可能出错。所以写缓冲器的深度通常比较小，一般只有几个 cache 行的深度。

还有一些改进 cache miss 的方法是：再在 buffer 前面加一个小的全关联的写缓冲，如下图所示，例如加一个 5 entries 大小的 Write cache。

图 5 改进 cache miss，加 write cache

2.2.3 Snoop 侦测

当 L2 需要 evict a cache line 时，L2 会跟踪 L1D，检测 cache line 是否在 L1D 中缓存过，如果是，L2 会给 L1D 发送 snoop-invalidate 请求，L1D 会将对应 line 的 valid 标记为置 0；此外，如果 line 的 dirty 位为 1，则 line 会被传到 L2，再进行后续的 evict 处理，被写到内存中，或者外部存储器中等等。

如果没有在 L1D 中缓存过，则 L2 的该操作对 L1D 控制器无影响。

该控制器不考虑从 L2 传输过来的 Snoop 信号。

三、系统设计

3.1 状态机设计

state		next state			转换条件
Q_1Q_0	状态名	状态名	Q_1	Q_0	
00	Idel	CompareTag	0	1	ld || st
01	CompareTag	Idel	0	0	hit
		Allocate	1	0	ld & miss &(~dirty)
		WriteBuffer	1	1	(ld & miss & dirty) || (st & miss)
10	Allocate	CompareTag	0	1	l2_ack
		Allocate	1	0	~l2_ack
11	WriteBuffer	Idel	0	0	st & WB_ready
		Allocate	1	0	ld & WB_ready

表 1 State diagram

控制器主要分为四个状态。下面对各个状态分别进行设计，各个状态设计到的信

号和定义也在这部分进行说明：

3.1.1 空闲 Idle

控制器初态。当有效的 CPU 请求成立，即 ld || st 时，切换到标志比较状态。

3.1.2 标志比较 CompareTag

• Cache 命中：如下表 Description 中描述；

Signal	Description	电路实现
write_l1	1. enable for L1 cache write 2. 如果 Cache 命中且为写操作，状态转换到空闲态，且 write_l1=1，写 L1D	write_l1= CompareTag & hit & st &clk
load_ready	1. indicate the data is successfully loaded and the data is ready for processor 2. 如果 Cache 命中且为读操作，状态转换到空闲态，且 load_ready=1，信号提供给 CPU 以进行之后的操作	load_ready=CompareTag & hit & ld &clk

• 读缺失且脏块：分为两步，先写 Buffer（next_state=WriteBuffer)，再从 L2 读 cache line(next_next_state=Allocate)
• 读缺失，非脏块：直接从 L2 读取数据（next_state=Allocate)
• 写缺失：直接写 Buffer（next_state=WriteBuffer)

3.1.3 分配 Allocate

Signal	Description	电路实现
read_l2	1. load request to L2 cache 2. 只要 l2_ack=0，即从 L2 读取数据未完成，read_l2 就一直为 1	read_l2= Allocate &(~l2_ack) &clk
l2_ack	1. indicate that the data loaded from L2 cache is arrived 2. L2 为 128byte，然而 L2→L1D 的 read data path 只有 128bit，因此传输一次要 8 个 cycle，这个操作由 L2 控制器控制。未在 L1D 控制器中体现 3. l2_ack =1：取 cache line 完成	来自 L2 控制器

3.1.4 写缓冲 WriteBuffer

Signal	Description	电路实现
write_l2	1. enable for write back to write buffer 2. 只要 WB_ready=0，即写操作未完成，write_l2 就一直为 1	write_l2= WriteBuffer &(~WB_ready)& clk
WB_ready	1. WB_ready=1：写 buffer 完成 2. 写一次要 8 个 cycles，用计数器 counter 计数：8 个时钟周期后，WB_ready=1	控制器内部信号 counter_n #(.n(3)) counter8(.... .co(WB_ready));

当写缓冲完成，即 WB_ready==1 时，如果是 st 指令，则 next_state=Idle 回到空闲态；如果是 ld 指令，next_state=Allocate ，L2 读分配。

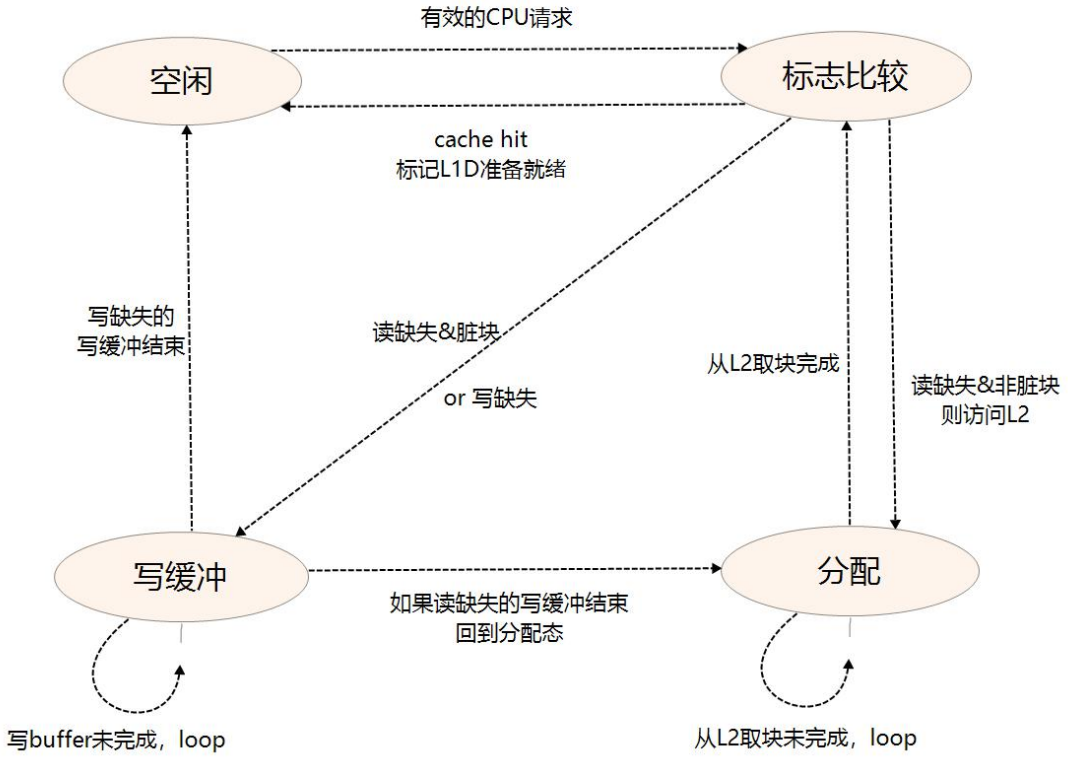

图 6 控制器流程图

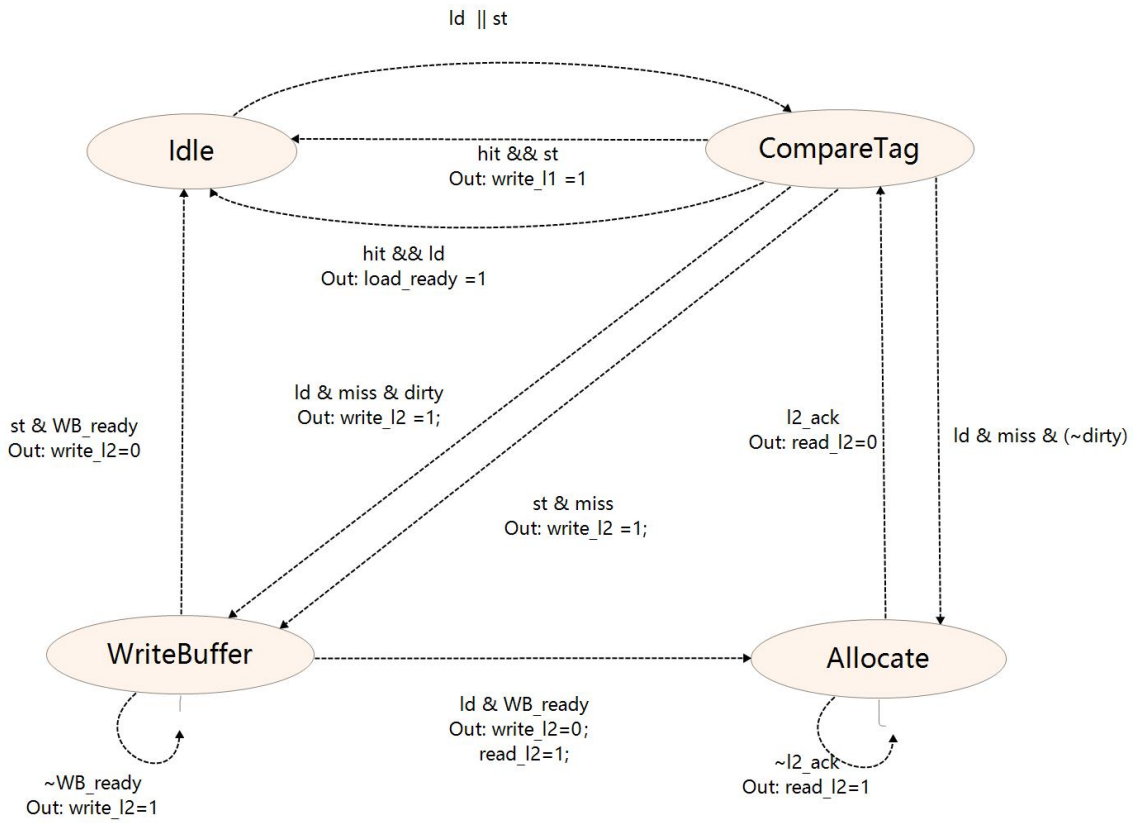

图 7 状态转换图

其他信号说明：

L1D 为 2-way set association ，因此从 L1D 索引到的数据为两个 cache line ，valid，dirty,tag_loaded 信号都为两个。取名为 way1、way2

Input/Output	Name	Description
input	valid1,valid2	valid bit for way1,way2
	dirty1,dirty2	dirty bit for way1,way2
	[31:0]addr	Cache access address
	[20:0]tag1_loaded	The tag of the indexed cache data address
	[20:0]tag2_loaded	
	ld	load data, read access
	st	store data, write access
output	hit	hit for wa1 or way2
	miss	Cache miss

wire	hit1,hit2	Cache hit for way1 ,way2
	dirty	控制器内部信号 由 dirty1 和 dirty2 决定
	valid	控制器内部信号 由 valid1 和 valid2 决定

3.2 电路设计

3.2.1 控制信号输出

（clk 上升沿触发）

用 D 触发器和与门、反相器实现：

load_ready =CompareTag & hit & ld &clk

write_l1= CompareTag & hit & st &clk

read_l2= Allocate &(~l2_ack) &clk

write_l2= WriteBuffer &(~WB_ready)& clk

用比较器和与门、或门实现：

```
assign hit1=(tag1_loaded==addr[31:11]) & valid1;
assign hit2=(tag2_loaded==addr[31:11]) & valid2;
assign dirty= (hit1&dirty1) || (hit2 & dirty2);
assign valid =(hit1&valid1) || (hit2 & valid2);
assign hit = (ld||st) & (hit1||hit2);
assign miss= ~hit;
```

3.2.2 状态转换电路

注：MUX0 和 MUX1 的输入端信号采用的简单门电路：与门非门等等，i.e: ~l2_ack 用 l2_ack 信号通过一个反相器实现，图中未画出，直接用了逻辑式子代替。

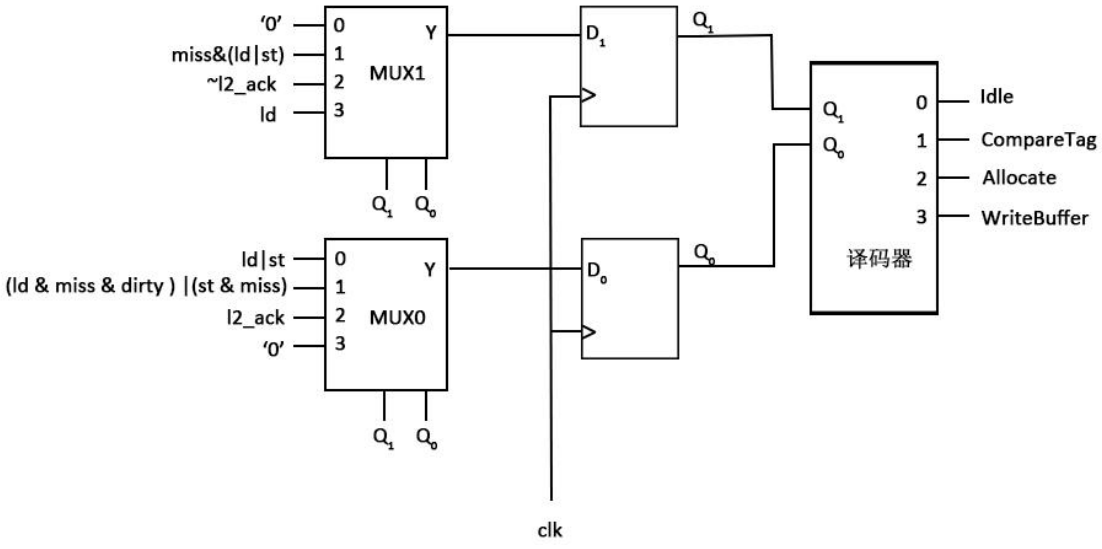

图 8 四个状态的逻辑电路图

3.3 Verilog HDL 编程

3.3.1 代码说明

1. 该控制器只关注控制信号的输入输出和状态的转变，忽略在 controller 和 L1D、L2、CPU 之间的 data 传输。因此 module 中未定义块偏移和字节偏移，以及 LRU bit，对 way1 和 way2 的选择以及 cache line 中 block 偏移处 word 的写/读操作也没有在该 module 中体现出。

2. 设置了 reset 信号，信号高电平有效时，初始化控制器，且状态回到初始的空闲（Idle）状态。

3. 状态机和输出控制信号分开描述的：

```
/*****          State Transition          *****/
....


/*****          Control signal          *****/
....
```

4. 写 buffer 用 8 个 cycles 用计数器实现

```
/*****          WriteBuffer counter          *****/
wire WB_ready;
counter_n #(.n(8) , .counter_bits(3) )    counter8(
...
.co(WB_ready) );
```

3.3.2 Testbench 测试

(1) testbench 说明：

测试程序见 **module** cache_controller_tb;

控制器主要想检测的是状态的转化逻辑和信号的输出是否正确，可以通过改变数据的 valid bit 来实现 cache hit/miss，而不用一直改变 addr，因此为了简洁起见，采用的 addr 和两个 tag_loaded 数据一直不变，如图 9，其中 tag1_loaded 等于 addr[31:11]， tag2_loaded 不等于 addr[31:11]

之后不再列出这三个信号。

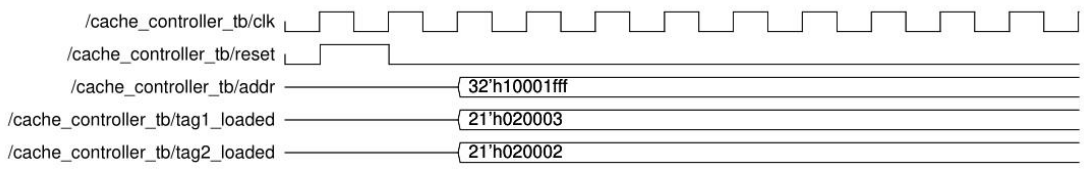

图 9 测试数据 1

(2) read hit 测试

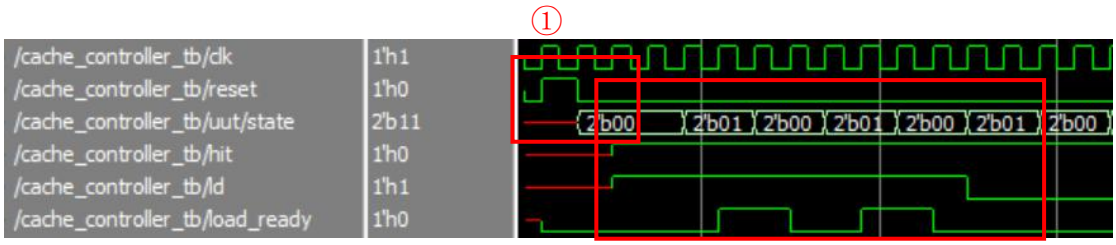

图 10 读命中测试

各个数字标号说明：

① 前提：控制器初始化，reset=0，state = 2'b00 即空闲——Idle state

② ：hi1=1 ， ld=1， 读命中，状态切换，从 00(Idle) → 01(CompareTag) → 00 且输出 load_ready =1 给 CPU

因为读信号一直持续，且 hit 所以又从 00 → 01 ...

直到 ld=0, st=1 变成写操作

(3) write hit 测试

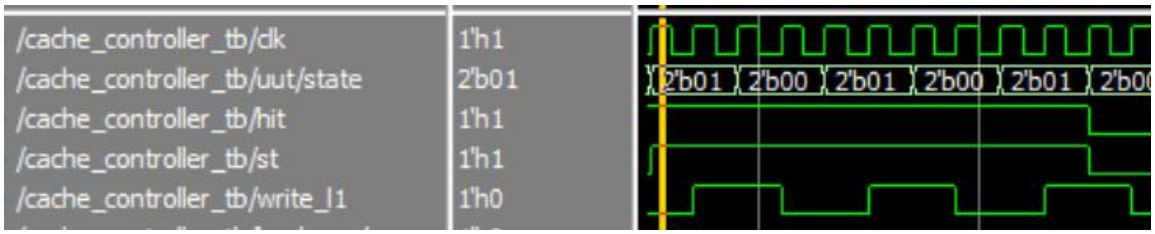

图 11 写命中测试

前提：写操作 st=1 ,且 hit=1

状态 state 改变同上面 read hit

输出信号 write_l1=1

(4) read miss + clean 测试

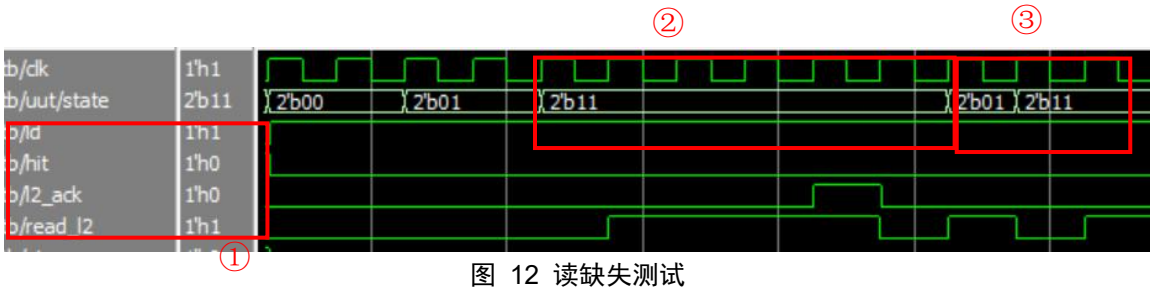

图 12 读缺失测试

①前提：读缺失 ld=1 ,且 hit=0 ， 非脏块。来自 L2 的 l2_ack=0
输出信号： read_l2 ， 只要在 Allocate 状态就等于 1， 从 L2 取数
②：在 Allocate 状态持续 8cycles，取数完成 l2_ack=1
③：由于 hit 由实际数据的 tag 和 valid 来决定，而我写的 verilog 没有涉及到实际数据，这个测试中把 valid 位一直置 0 的，因此 hit 始终为 0，state 一直在 CompareTag 和 Allocate 之间切换，而没有回到 Idle

(5) write miss 测试

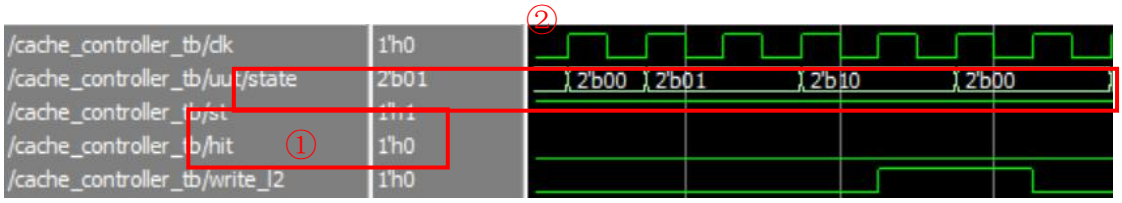

图 13 写缺失测试

①前提：写缺失 st=1 ,且 hit=0 ，dirty 与否无所谓，都是写 buffer。来自 L2 的 l2_ack=0
输出信号： write_l2 写 buffer
②：状态 Idle → CompareTag （发现 miss 后） → WriteBuffer → Idle

四、参考文献

[1] TMS320C621x/C671x DSPTwo-Level Internal MemoryReference Guide
[2] “Cache write policies and performance” Norman Jouppi, *Proc 20th International Symposium on Computer Architecture (ACM Computer Architecture News21:2)*, May 1993, pp. 191-201.