DMA项目文档

1. 项目内容

DMA是计算机中不可缺少的一个部分，其主要实现内存中的数据与CPU中数据的交互。支持从内存中取出数据传入CPU中，也支持从CPU中取出数据传入内存中。DMA中需要有2个数据的缓冲区BUF1和BUF2，工作在不同的状态。例如：传输的方向为内存->CPU时，BUF1在接收内存传入的数据，BUF2在向CPU输出数据；只有当BUF1满了，BUF2空了之后，BUF1才和BUF2交换，交换之后，BUF2接收内存传入的数据，BUF1向CPU输出数据；如此交替下去，实现流水。

1. 整体设计

DMA与CPU之间可进行双向数据传输，其基本过程相似，设计时利用复位信号来控制数据传输的方向，每次复位数据传输方向发生改变，初始化的数据传输方向为从MEM到CPU方向。

数据传输过程类似于两个FIFO的联结，设计时参考FIFO的设计，利用三段式状态机实现功能。

DMA的两个BUF在传输过程中要实现交换，根据BUF1和BUF2的缓冲区空、缓冲区满的状态将状态机分为6个状态（利用独热码），实现单方向数据传输时的流水作业。

整体程序分为四段，第一段是时序逻辑，实现次态到现态的迁移；第二段是组合逻辑，实现各个状态的转换；第三段是时序逻辑，用来实现DMA的主体功能；第四段是组合逻辑，用来辅助控制DMA的功能和状态。

1. 模块接口与参数
2. **接口**

Input与Output是参照实验要求的接口，未进行更改，接口及功能如下。

(1)Input接口

**clk**：时钟信号，时序逻辑取其上升沿触发（1bit）；

**rst\_n**：复位信号，在为0时进行复位操作（1bit）；

**mem\_to\_dma\_valid**：mem传入的数据是否有效（1bit）；

**mem\_to\_dma\_enable**：mem是否准备好接受数据（1bit）；

**cpu\_to\_dma\_valid**：cpu传入的数据是否有效（1bit）；

**cpu\_to\_dma\_enable**：cpu是否准备好接受数据（1bit）；

**mem\_data\_out**：mem传出的数据（4bit）在valid和enable同时为1的时候才会被dma读入；

**cpu\_data\_out**：cpu传出的数据（8bit）在valid和enable同时为1的时候才会被dma读入；

1. Output

**dma\_to\_mem\_valid**：dma传入的数据是否有效（1bit）；

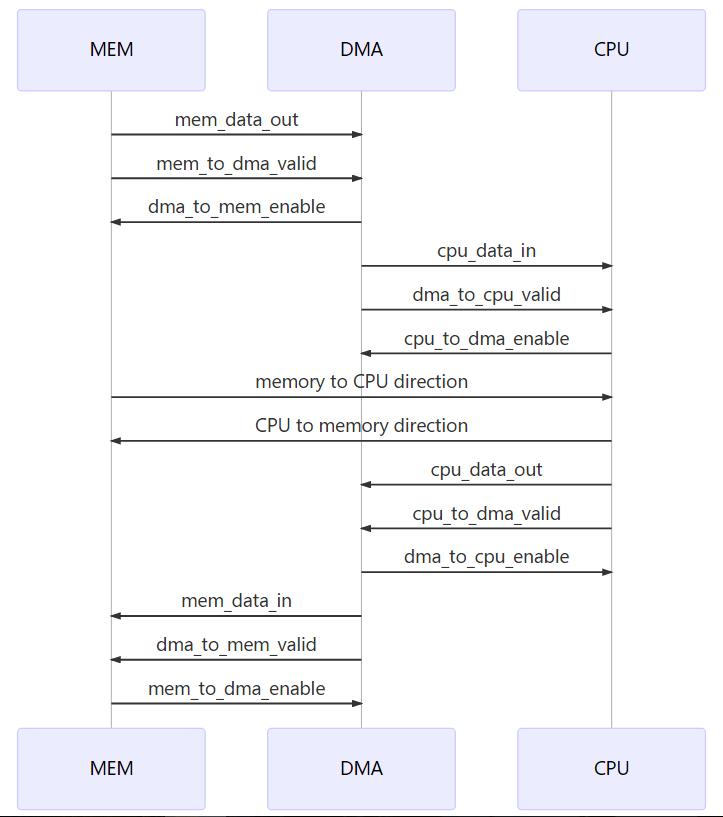
**dma\_to\_mem\_enable**：dma是否准备好接受数据（1bit）；

**dma\_to\_cpu\_valid**：dma传入的数据是否有效（1bit）；

**dma\_to\_cpu\_enable**：dma是否准备好接受数据（1bit）；

**mem\_date\_in**：dma传入mem的数据，只有在valid和enable同时为1的时候才会被读入（4bit）；

**cpu\_data\_in**：dma传入cpu的数据，只有在valid和enable同时为1的时候才会被读入（8bit）；



1. **状态**

现态和次态表示使用currentstate（当前状态6bit）和nextstate（下一状态6bit）；

利用独热码和BUF1、BUF2的状态划分6个状态，前三个状态是在向buf1中写入数据，在buf2中读取数据。BUF的空满状态根据有效数据判断，有效数据是指已经写入dma但是还有被读取的数据。

前三个状态是向buf1写入数据，从buf2中读取数据

**S1**:buf1未满（需要写入数据），buf2未空（需要读出数据）；

**S2**:buf1满，buf2未空；

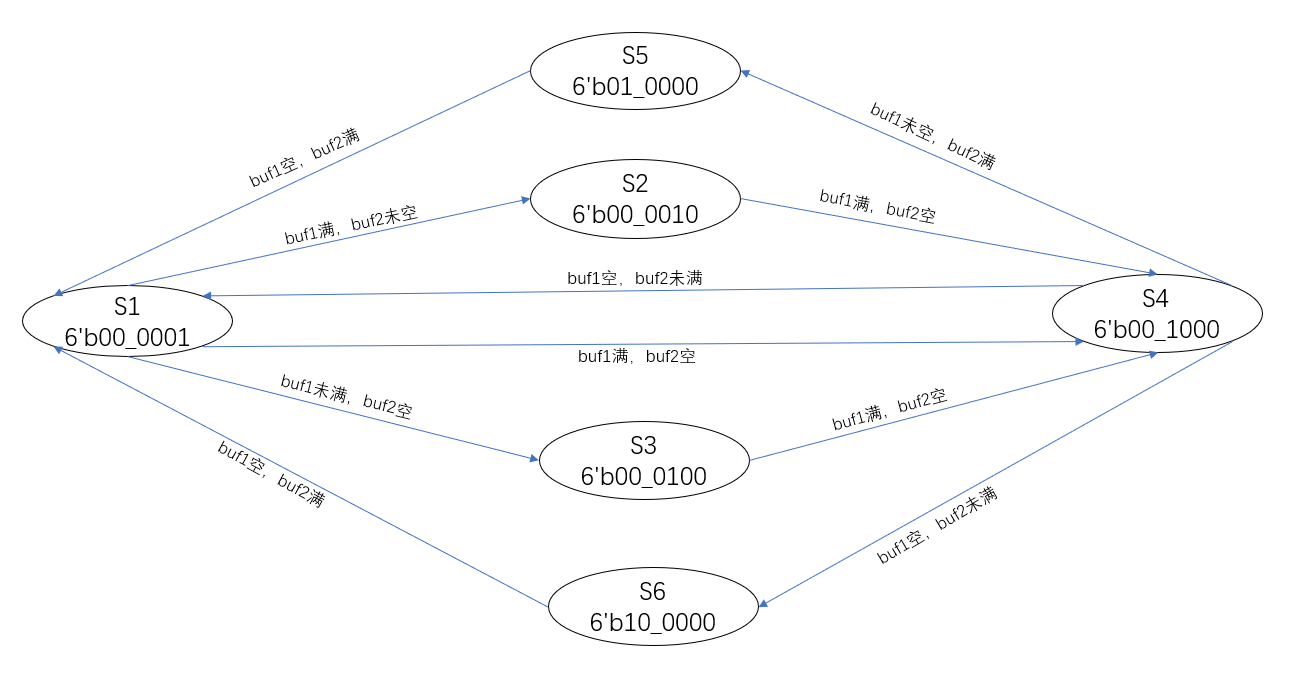
**S3**:buf1未满，buf2空；

后三个状态是在向buf2中写入数据，从buf1中读取数据。

**S4**:buf1未空，buf2未满；

**S5**:buf1未空，buf2满；

**S6**：buf1空，buf2未满；



1. **其他参数**

**传输方向**

**data\_flow\_direction**:控制数据传输方向，为0时从CPU到MEM，为1时从MEM到CPU；每次复位时，data\_flow\_direction信号取反

**DMA缓冲区**

**buf1[7:0][7:0]**:缓存区1（）；

**buf2[7:0][7:0]**:缓存区2（）；

**BUF读出、写入指针**

buf1和buf2的读取指针，指示MEM或CPU读取的位置，每次增到8时自动舍去高位变为0。

**buf1\_read\_ptr**:用来负责buf1输出工作的指针（3bit）；

**buf2\_read\_ptr**:同上，负责buf2（3bit）；

buf1和buf2的写入指针，指示MEM或CPU写入的位置，每次增到8时自动舍去高位变为0。

**buf1\_write\_ptr**:负责buf1写入dma的操作的指针（3bit）；

**buf2\_write\_ptr**:同上，负责buf2（3bit）；

MEM的数据是4bit传输，buf1和buf2的每个寄存器是8bit，需要有一个信号来标志读写的是高位还是低位。

**buf1\_write\_low**:为1代表buf1在写入时写的是低位，每次操作后取反（1bit）；

**buf1\_read\_low**:同上，从buf1读取数据时用（1bit）；

**buf2\_write\_low**:同上，buf2写入数据时用（1bit）；

**buf2\_read\_low**:同上，buf2读取数据时用（1bit）；

**计数**

buf1和buf2中的数据个数（按4bit计）要进行计数，便于输入输出和状态转换

**counter\_buf1**:用来计数buf1中的有效数据（5bit）；

**counter\_buf2**:同上，计数buf2中的有效数据（5bit）；

**辅助功能**

DMA是两个同步的FIFO，要实现数据传输过程中buf1和buf2的瞬间翻转并接受输入和输出功能，设置了下面接个参数来及时控制信号的改变，避免数据传输的延迟。

**buf1willfull**:当buf1在填数据的时候用，当buf1在这个时钟上升沿录入了最后的数据，这个值就设为1，用来帮助转换状态（去除状态转换的那一格的延迟）（1bit）；

**buf2willempty**:当buf2在弹出数据的时候用的，弹出最后一个数据的时钟上升沿时置为1（1bit）；

**buf1willempty**:同上（1bit）；

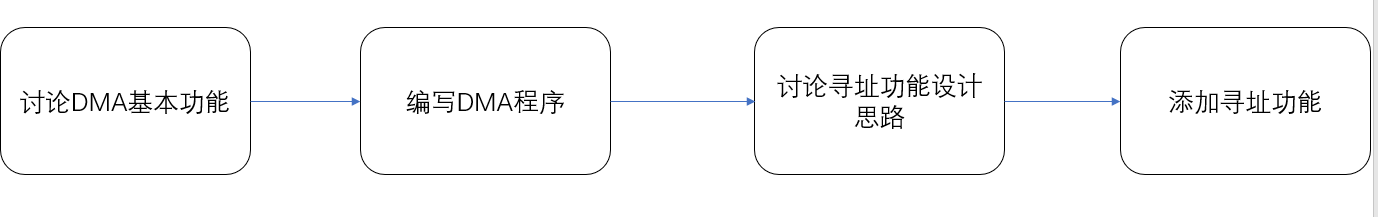
**buf2willfull**:同上（1bit）；

**cpu\_in\_finish**:在mem到cpu的方向，如果读入已经读完了，但是还没有输出完就会进入S2或者S5，这时候最后一个数据一进入dma的输出端口状态就会改变但是最后一个值还没有被cpu读进去，等最后一个值被录进去了，这个值就会为1，从而把valid拉低（1bit）；

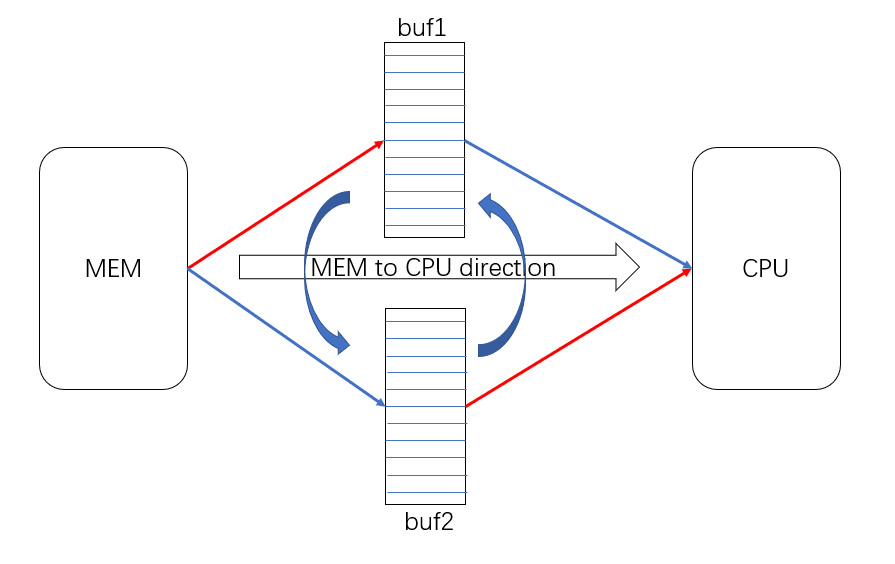
**mem\_in\_finish**:同上，用于cpu到mem的方向（1bit）；

1. 流程图

小组合作过程



DMA设计



以MEM到CPU方向为例，初始化时buf1和buf2均为空，进行红色箭头方向传输，buf2不输出，buf1满了之后buf1和buf2进行调转，进行蓝线方向输出。

1. 实现基本功能

本项目的基本功能部分实现了一个残缺版的DMA，只有当一个buf数据存满并且另外一个buf完全输出时，两个buf的功能才会转换。

1. 亮点
2. 每次复位时改变数据的传输方向，实现手动控制MEM到CPU和CPU到MEM方向的数据传输。
3. 利用三段式状态机实现DMA的功能，将组合逻辑和时序逻辑分离开，更方便程序的优化和维护，更符合思维习惯。
4. 利用独热码，实现状态的转换，容错性高。
5. 实现无延迟状态转换，简单来说就是一个缓冲区满时，瞬间交换状态并同时输出，不会出现延迟，为了实现这个功能我们对最初版本的DMA程序做了比较大的修改。

比如说，在mem到cpu 的方向，最初版本的DMA从S3到S4，当buf1满了之后还会等一个时钟上升沿到来才会把dma\_to\_cpu\_valid拉高，然后把buf1的最初的数据输出。新版本的DMA在buf1填满的那个时钟上升沿就能把dma\_to\_sth\_valid拉高，并且把buf1的最初的数据放在cpu\_data\_in里等待cpu的读取，这个功能是使用buf1willfull、buf1willempty、buf2willfull、buf2willempty控制状态跳变来实现的。比如还是从mem到cpu的方向，在S3状态，buf1在一个时钟上升沿写入了最后的数据，那么就把buf1willfull拉高，然后在第二块的状态转换的组合逻辑中是有buf1willfull的，它的拉高会让nextstate瞬间变成S4，而第四段的组合逻辑再根据状态S1将dma\_to\_cpu\_valid瞬间拉高，那么如果下个上升沿cpu\_to\_dma\_enable正好也是1（因为cpu\_to\_dma\_enable为了保证鲁棒性，在testbench里面设的是随机值），就可以直接让cpu取到数据了。

1. DMA能够实现把MEM或者CPU输入的数据完全读出再进行下一步操作。这是在实现无延迟是加入的变量cpu\_in\_finish和mem\_in\_finish实现的，因为我们的dma的状态是没有延迟的，组合逻辑也是用的nextstate的，所以当写入dma已经结束但是还没有被mem（或者cpu）读取干净，就会在把最后一个数据放入mem\_data\_in(或者cpu\_data\_in)的时候状态就会有所改变，但是mem和cpu是在时钟上升沿才会读取mem\_data\_in和cpu\_data\_in的数据，所以我要等最后一个数据被mem或者cpu读取之后再将valid置0，那么这个dma的功能就完善了。
2. 程序的鲁棒性好。
3. 缺点

程序设计风格和硬件设计不是特别符合，习惯于C语言的软件设计，编码风格有待改进。

八、附加功能——寻址的实现

一、功能简述

DMA主要实现的是内存中数据和CPU中数据的交互，而在实际操作中，用户会给予CPU一个命令，含有对所需要数据的要求，而CPU会解析这条命令，向内存传输用户所需的数据长度，而内存接到命令之后会不断向CPU传输对应长度的数据，并输出数据的地址。由于系统内部存在一个输出数据的长度上限（简称burst），当长度超出时，会拆分成两端或者多段输出。

二、整体设计

根据已经实现的dma功能，在其中加入输入接口Length，随机以表示用户的需求，两个和地址有关的参数valid和enable，分别随即表示长度的有效性和此时是否能读入用户需求的长度。之后运用对输入和输出数据的分别统计，来表示此时是否超出burst，当输入已达到上限值而输出未到时，及时跳转状态，停止输入，而等待输出结束后进入下一轮输入输出。因此增添了一个参数address\_in\_enable来控制输入状态。

三、增添的接口与参数

1、接口

Input与Output是参照实验要求的接口，未进行更改，接口及功能如下。

1. Input接口

length：用户所需数据长度，在testbench中进行随机读取（11bit）；

address\_valid：length传入的数据是否有效（1bit）；

（2）Output接口

address\_enable：此时mem是否可以接受cpu传输的长度数据（1bit）；

1. 其他参数

address\_in\_enable：表示此时是否可以从内存向dma输入数据（1bit）

counter\_address\_in：统计已经从内存输入dma的数据数（11bit）

counter\_address\_out：统计已经从dma输出到cpu的数据数（11bit）

return\_S3：当其为1的时候回到S3（1bit）

burst\_will\_finish：表示burst是否将要取满（1bit）

四、程序设计思路

1、首先随机读入length和address\_valid，当address\_valid和address\_enable均为1时表示此时length可以读入，用户需求有效并且内存可以满足此需求。这时将counter\_address\_in的值置为0，而counter\_address\_out的值置为length/4（由于length表示的是bit，而counter表示的是位）。

2、在第三段时序逻辑中，加入对于counter\_address\_in和counter\_address\_out的判断，若输入和输出都已经完成，则address\_enable将被置为1，而address\_in\_enable也将被置为1；否则，若counter\_addreass\_in输入完成，但是counter\_address\_out不为0，即输出未完成，address\_in\_enable将被置为0。同样，在第三段时序逻辑中，每一次有效数据的输入，counter\_buf+1时，counter\_address\_in将为同时+1，而counter\_buf-2时，counter\_address\_out将为同时-2。

3、在第二段组合逻辑实现状态变换中，加入了新的判断条件，即当输入已经完成而输出未完成时，将为跳转为S2或S5来进行只输出不输入的操作。还有当一次length操作完成时，将会回到S3，通过return\_S3参数控制

五、亮点

1、可以实现较长length的有效稳定输入，并且同样保证了无延迟。