# 直接内存存取(DMA)开发文档

## 数字电路期末实验-小组作业

小组成员: 王华强 刘蕴哲 蔡昕

## 综述

本作品实现了一个支持CPU和内存双向传输数据的DMA（满足所有基本要求）。除此之外，该DMA可以满足接受CPU向DMA传输所处理的数据所在的地址和数据长度的要求，并且DMA可根据CPU所决定并输出的数据长度来控制实际传输数据的长度，在传输完毕要求长度的数据则不再传输数据，而是等待CPU下一条包含所处理的数据所在的地址和数据长度的指令，之后再进行要求长度数据的传输。本作品使用了模块化设计，共包括七个模块，各模块之间的调用关系将在后文给出。同时，在testbench中完善了数据测试部分，即将所有CPU和DMA、内存和DMA之间的数据传输过程打印出来，方便使用者在不检查波形的情况下，通过打印的数据传输过程来判断该DMA的功能是否实现。

整个项目由5个模块组成. 其中DMA本身采用了模块化设计, 由两个FIFO模块和外部控制逻辑组成.

扩展版本支持地址控制功能. 在原实现的基础上设计了单独的地址控制模块, 总模块数达到7个. 其余模块也相应的做了大量改动. 但整体结构与基本版本保持一致. 扩展版本相对于基础版本的改动将在扩展部分详细说明.

其中basic版本是基础的要求, 而addr_supported版本是支持地址的版本.

## 基本版本

### 1. 整体概述

完备的注释仅限在支持地址的版本中.

文档中仅仅列出接口定义, 修改位置, 原理说明. 具体的实现方式敬请参阅源代码

基础版本中各个模块的层级关系如下所示:

```
graph TD;
Testbench-->CPU
Testbench-->DMA
Testbench-->MEM
DMA-->FIFO1
DMA-->FIFO2
```

各个模块的简要说明和接口设置如下所示:

### 2. FIFO模块

先入先出存储模块, 额外的功能包括: 支持直接输出full, empty状态(使用组合逻辑), 支持两种写入输出模式, 可以通过workmode端口配置读写模式(4位/8位). 支持地址的版本中添加了直接置满功能.

```
module FIFO(
    input clk,
    input resetn,
    input workmode,
    input input_valid, output_enable,
    input [7:0] fifo_in,
    output reg[7:0] fifo_out,
    output reg output_valid,
    output reg input_enable,
    output empty, //当前fifo已满
    output full   //当前fifo已空
```

```
);
```

workmode在reset的时候设定, 之后一直保持不变.

只有在fifo满以后才能够写入, 在fifo空之后才可以输出.

Fifo状态由以下4个reg控制: position(记录当前写入/输出位置), writehigh(在4位模式时控制高低位),output_valid, input_enable

### 3. DMA模块

整个硬件的主体, 由两个FIFO模块和外部控制逻辑组成. 基础版本支持两个fifo的交替输入输出. 支持地址的版本改动较大, 将在下文详细介绍.

```
module DMA(

input clk,
input resetn,
input mode,              //模式选择:控制DMA的工作方式:内存->CPU 或 CPU->内存

input dma_to_mem_enable, //MEM是否准备好接收数据。
input mem_to_dma_valid, //MEM中传入的数据是否有效。
input dma_to_cpu_enable, //CPU是否准备好接收数据。
input cpu_to_dma_valid, //CPU传入的数据是否有效。

input [3:0] mem_data_out, //内存信号输出
input [7:0] cpu_data_out,  //中央处理器信号输出

output dma_to_mem_valid, //向MEM传出的数据是否有效
output mem_to_dma_enable, //DMA准备好自MEM接收数据
output cpu_to_dma_enable, //DMA准备好自CPU接收数据
output dma_to_cpu_valid, //向CPU传出的数据是否有效

output [3:0] mem_data_in, //内存信号输入
output [7:0] cpu_data_in  //中央处理器信号输入
);

...

FIFO buf1(
    .clk(clk),
    .resetn(resetn),
    .workmode(buf1_mode)
);

FIFO buf2(
    .clk(clk),
    .resetn(resetn),
    .workmode(buf2_mode)
);
```

其中含有两个FIFO模块, 其中一个为输入缓冲器, 另一个为输出缓冲器, 通过input_buf reg来控制.

其运行逻辑如下:

```
graph TD;
reset-->开始输入输出
开始输入输出-->输入buf读入
开始输入输出-->输出buf输出
输入buf读入--输入buf满-->一空一满,交换输出buf和输入buf
输出buf输出--输出buf空-->一空一满,交换输出buf和输入buf
```

```
一空一满,交换输出buf和输入buf-->开始输入输出
```

其中mode在reset时被设定, 之后在运行中保持不变.

DMA的数据输入输出接口直接接到当前输入输出缓冲fifo的对应接口.

实际的硬件示意图如下: 其中实线虚线分别是工作时的两种可能接线

```
graph LR;

信号输入端-.->FIFO1
信号输入端-->FIFO2
FIFO1-->信号输出端
FIFO2-.->信号输出端
```

4. TESTBENCH

在testbench中, CPU, MEM, DMA 三个模块被连接到一起. 支持通过修改MODE来确定数据流的传输方向. 有以下规定:

```
`define MODE 0
或
`define MODE 1

parameter mode_cpu_to_mem 1'b1;
parameter mode_mem_to_cpu 1'b0;
```

testbench中设计了数据输出提示, 成功输入输出的数据会_每八位_的被输出到命令行中. 在带有地址的版本中, testbench有更多的功能, 测试了数据传输的正确性并且检验了dma的状态是否正确, cpu, mem子模块添加了相关的控制逻辑.

**4-1.模拟CPU模块**

此模块支持随机的数据输出, 并在接收输出到有效数据时将其在命令行中显示出来以便于调试.

模拟CPU模块接口定义如下:

```
module CPU(
input clk,
input resetn,
input cpu_to_dma_enable, //DMA准备好自CPU接收数据
input dma_to_cpu_valid, //向CPU传出的数据是否有效
input [7:0] cpu_in_socket,  //CPU的数据传入端口

output reg dma_to_cpu_enable, //CPU是否准备好接收数据。
output reg cpu_to_dma_valid, //CPU传入的数据是否有效。
output reg [7:0] cpu_out_socket //CPU的数据传出端口
);
```

其会随机产生数据并在收到数据时在命令行中显示出结果.

**4-2. 模拟MEM模块**

此模块支持随机的数据输出, 并在接收输出到有效数据时将其在命令行中显示出来以便于调试.

模拟MEM的端口设置如下:

```
module MEM(
input clk,
input resetn,
input mem_to_dma_enable, //DMA准备好自MEM接收数据
input dma_to_mem_valid, //向MEM传出的数据是否有效
input [3:0] mem_in_socket, //CPU的数据传入端口

output reg dma_to_mem_enable, //MEM是否准备好接收数据。
output reg mem_to_dma_valid, //MEM传入的数据是否有效。
output reg [3:0] mem_out_socket //MEM的数据传入端口

);
```

其会随机产生数据并在收到数据时在命令行中显示出结果.

5. 设计特色

- 地址访问
- 模块化设计
- 完善的自动测试模块

6. 电路图

---

## 支持地址的版本

1. 概述

支持地址和长度的版本满足以下逻辑:

```
sequenceDiagram;

地址控制模块->>原有DMA模块: 地址传输完成,启用原有DMA模块
原有DMA模块->>地址控制模块: 传输的数据达到指定长度,移交控制权
原有DMA模块-->>地址控制模块: 传输状态实时反映给地址控制模块
```

为了实现这部分的功能, 在原有的DMA基础上外挂了一个地址控制模块, 使用地址控制模块接管原有DMA的输入输出控制端口.

地址控制模块的 **状态图** 如下图所示:

```
graph TD;
reset-->等待地址输入;
等待地址输入--地址输入-->等待地址输出;
等待地址输出--地址输出:设置len计数器,进入数据操作状态-->数据操作状态;
数据操作状态--输入部分len计数器完成:强制置满fifo-->等待数据操作完成;
数据操作状态--输出部分len计数器完成:强制清空fifo-->等待数据操作完成;
等待数据操作完成--输入输出均完成-->等待地址输入;
```

注意:

- 在等待地址输出的同时dma和cpu之间已经可以开始传输数据
- 在地址输出之后dma和mem之间可以开始传输数据

各模块间的层级关系如下:

```
graph TD;
Testbench-->CPU
Testbench-->MEM
Testbench-->DMA_with_address_control
DMA_with_address_control-->DMA_original_version
DMA_with_address_control-->ADDRESS_TRANSMITER
DMA_original_version-->FIFO1
DMA_original_version-->FIFO2
```

新的端口设置如下所示

```
module DMA_withaddr(

//数据部分----------------------------------------------------------

input clk,
input resetn,
input mode,              //模式选择:控制DMA的工作方式:内存->CPU 或 CPU->内存

input dma_to_mem_enable, //MEM是否准备好接收数据。
input mem_to_dma_valid, //MEM中传入的数据是否有效。
input dma_to_cpu_enable, //CPU是否准备好接收数据。
input cpu_to_dma_valid, //CPU传入的数据是否有效。

input [3:0] mem_data_out, //内存信号输出
input [7:0] cpu_data_out,  //中央处理器信号输出

output dma_to_mem_valid, //向MEM传出的数据是否有效
output mem_to_dma_enable, //DMA准备好自MEM接收数据
output cpu_to_dma_enable, //DMA准备好自CPU接收数据
output dma_to_cpu_valid, //向CPU传出的数据是否有效

//地址控制部分------------------------------------------------------

input address_in_valid,     //CPU传入给DMA地址值有效
input address_out_valid,   //DMA处于可以传出地址状态
input [31:0]len_in,     //接收传入的长度
input [31:0]addr_in,    //接收传入的地址

output address_out_enable,  //DMA传出地址值可被MEM接收
output address_in_enable,  //CPU传出地址值可被DMA接收, DMA处于可以接受地址状态
output [3:0] mem_data_in, //内存信号输入
output [7:0] cpu_data_in,  //中央处理器信号输入
output [31:0] address_reg,   //地址暂存器
output [31:0] len_reg        //长度暂存器 unit: bit

);
```

2. 原DMA模块的端口设置

在此前连接在原DMA上的CPU, MEM的控制端口由地址控制模块接管.

逻辑如下所示:

输入部分:

输入长度未达到len, 仍可输入:

```
input mem_to_dma_valid, //MEM中传入的数据是否有效。 接通
```

```
input cpu_to_dma_valid, //CPU传入的数据是否有效。    接通

output mem_to_dma_enable, //DMA准备好自MEM接收数据   接通
output cpu_to_dma_enable, //DMA准备好自CPU接收数据   接通
```

这里的接通是指由DMA自行控制;

输入长度达到len, 输入端关闭:

```
input mem_to_dma_valid, //MEM中传入的数据是否有效。  置为0
input cpu_to_dma_valid, //CPU传入的数据是否有效。    置为0

output mem_to_dma_enable, //DMA准备好自MEM接收数据   置为0
output cpu_to_dma_enable, //DMA准备好自CPU接收数据   置为0
```

输出部分:

输出长度未达到len, 仍可输出:

```
input dma_to_mem_enable, //MEM是否准备好接收数据。   接通
input dma_to_cpu_enable, //CPU是否准备好接收数据。   接通
output dma_to_mem_valid, //向MEM传出的数据是否有效   接通
output dma_to_cpu_valid, //向CPU传出的数据是否有效   接通
```

输出长度达到len, 输出端关闭:

```
input dma_to_mem_enable, //MEM是否准备好接收数据。   置为0
input dma_to_cpu_enable, //CPU是否准备好接收数据。   置为0
output dma_to_mem_valid, //向MEM传出的数据是否有效   置为0
output dma_to_cpu_valid, //向CPU传出的数据是否有效   置为0
```

输入长度达到len时, 将DMA中当前输入FIFO直接置满;

输出长度达到len时, 将DMA中当前输出FIFO直接置空;

## 3. 各模块变化

### 3-1. DMA模块(DMA)

额外引出了输入FIFO置满, 输出FIFO置空引脚. 通过直接置满置空, 可以在数据传输达到要求的长度之后直接截断, 从而加速数据传输速度.

### 3-2. FIFO模块(FIFO)

额外引出了置满, 置空引脚. 通过直接置满置空, 可以在数据传输达到要求的长度之后直接截断, 从而加速数据传输速度.

同时又有空满显示引脚, 方便外侧模块直接读取到其状态.

### 3-3. TESTBENCH

#### 整体设计(TEST)

较之前相比, 调整了CPU, MEM模块使之支持地址操作. 地址部分支持随机长度和指定长度.

#### 模拟CPU, MEM模块(CPU,MEM)(鲁棒性测试)

地址控制, 数据传输部分的所有数据全部随机给出, 在传出/收到有效数据时在命令行中显示出来.

在上交的版本中为了测试鲁棒性采用这种TESTBENCH, 正常工作的情况下, 运行逻辑如下所示:

#### 模拟CPU模块(CPU)

正常的运行逻辑如下图所示.

```
graph TD;
reset-->等待地址传出
等待地址传出--地址传出-->数据传输
数据传输--数据传输完毕,达到预定的长度-->等待地址传出
```

增添了地址控制的相关引脚, 新的端口设置如下所示.

```
module CPU(

//数据部分
input clk,
input resetn,
input cpu_to_dma_enable, //DMA准备好自CPU接收数据
input dma_to_cpu_valid, //向CPU传出的数据是否有效
input [7:0] cpu_in_socket,

output reg dma_to_cpu_enable, //CPU是否准备好接收数据。
output reg cpu_to_dma_valid, //CPU传入的数据是否有效。
output reg [7:0] cpu_out_socket,
//------------------------------
//地址控制部分
input addr_out_enable,  //DMA准备好自CPU接收地址
output reg addr_out_valid,  //CPU正在传出有效的地址

output reg [31:0] addr_out, //地址输出
output reg [31:0] len_out  //数据长度输出

);
```

#### 模拟MEM模块(MEM)

正常的运行逻辑如下图所示.

```
graph TD;
reset-->等待地址传入
等待地址传入--地址传入-->数据传输
数据传输--数据传输完毕,达到预定的长度-->等待地址传入
```

增添了地址控制的相关引脚, 新的端口设置如下所示.

```
module MEM(

//数据部分
input clk,
input resetn,
input mem_to_dma_enable, //DMA准备好自MEM接收数据
input dma_to_mem_valid, //向MEM传出的数据是否有效
```

```
    input [3:0] mem_in_socket,

    output reg dma_to_mem_enable, //MEM是否准备好接收数据。
    output reg mem_to_dma_valid, //MEM传入的数据是否有效。
    output reg [3:0] mem_out_socket,
    //-------------------------------------
    //地址控制部分
    input addr_in_valid,  //DMA传出有效的地址
    output reg addr_in_enable,  //MEM可以接收地址

    input [31:0] addr_in, //地址输入
    input [31:0] len_in //数据长度输入

    );
```

## Acknowledge

感谢 ram wire 创作出这么赞的曲子, 听着歌写代码真是太棒了!

## Contribution

- 统筹: 王华强
- 架构: 王华强
- FIFO: 刘蕴哲
- DMA(两个版本): 王华强, 蔡昕
- TESTBENCH, 整体测试: 蔡昕
- 文档: 王华强, 蔡昕
- 注释: 王华强, 蔡昕, 刘蕴哲

## Reference

- 不存在的

---