

DMA 并行传输机制的设计与实现

余再祥, 陈海波, 孙永节, 吴文俊
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: DMA 的传输控制机制是影响数据传输速度的最直接因素, 经过对传输控制结构和传输协议的分析, 采用了流水线的思想, 设计了支持突发传输、64 位位宽和读写并发的并行传输机制. 通过实际应用表明, 该设计实现了流水化和读写并发, 与串行传输机制相比, 数据传输速度提高了三倍. 该机制对于 DMA 控制器的设计具有一定的指导意义.

关键词: DMA; 传输控制机制; 传输请求; 流水线; 并发

中图分类号: TP331.2

文献标识码: A

文章编号: 1000-7180(2012)02-0001-06

Design and Implementation of the Parallel Transmission Mechanism in the Direct Memory Access Controller

YU Zai-xiang, CHEN Hai-bo, SUN Yong-jie, WU Wen-jun

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract: With the continuous development of DSP (Digital Signal Processor) and its high performance requirement, transfer speed has received increasing attention. The transmission control mechanism of DMA (Direct Memory Access) is the direct factor that affects the data transmission speed. By analyzing the transmission control and transmission protocol, we propose a transfer mechanism which supports double-words access, burst and parallel transmission based on the pipeline strategy. Because of the pipeline and parallel transmission, our proposed mechanism is three times faster than the serial mechanism according to our experimental results. This mechanism will have a significant meaning for the design of DMA controller.

Key words: DMA; Transmission control mechanism; Transfer requests; Pipeline; Parallelism

1 引言

DMA 部件是 DSP 芯片的数据传输控制核心, 可以独立控制内存和各外设接口之间的数据传输, 传输过程不需要 CPU 的干预, 提高了 DSP 的总体性能. 随着 DSP 应用领域的扩展, 对大量数据的实时处理已成为 DSP 的必然发展趋势, 这也对 DMA 的数据传输速度提出了更高的要求^[1].

当前, 提高 DMA 的数据传输速度主要通过两个途径: 增加通道和提高频率. TI 公司最先进的 TMS320C66x 系列多核 DSP 已采用具有 64 个通道和 256 个扩展参数组的 EDMA(加强型 DMA); 在

频率方面, 大多 DMA 部件可以和 CPU 运行在同一频率, 最高可达 1.5 GHz. 但由于传输结构和硬件资源等因素的限制, DMA 部件在传输机制方面的改进比较缓慢, 传统的串行传输控制机制依然广泛使用, 这种机制结构简单, 但传输大量数据时效率较低, 也是影响 DMA 数据传输速度提高的最直接因素^[2].

本文通过对 DMA 传输结构的研究, 采用流水线的思想, 设计了支持突发传输、64 位位宽和读写并发的并行传输控制机制, 数据传输速度比串行机制提高了三倍.

收稿日期: 2011-05-09; **修回日期:** 2011-07-07

基金项目: 教育部长江学者和创新团队发展计划(高性能微处理器创新团队, IRT0614)

2 DMA 的结构和串行传输机制

本文设计的是 EDMA 的传输控制机制, EDMA 的结构如图 1 所示, 主要由参数 RAM(64 个通道)及控制器、传输请求仲裁及多队列、传输控制模块三大部分组成, 相关的端口共有七个: 外存、内存、串口 0、串口 1、L2 cache、HPI 和 PCI。其中, 外存、内存和串口的传输请求是先编程配置参数 RAM 中相关传输参数组, 当同步事件到达后, 再提交给传输请求仲裁及多队列, 称为内部传输请求; 主动设备 L2 cache、HPI 和 PCI 发起的传输请求则称为外部传输请求。传输请求仲裁模块对各传输请求进行仲裁后, 按优先级分配给多队列(共四个队列: Queue0~3, 优先级由高到低), 在队列中, 传输请求按照先进先出的原则等待传输控制模块的处理。传输控制模块直接控制传输请求的执行, 它的结构和性能是影响数据传输速度的关键^[3]。

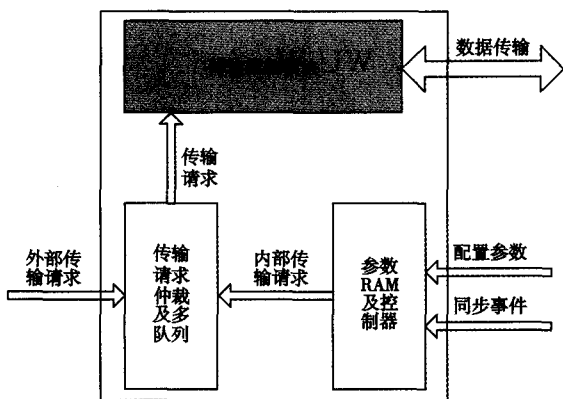


图 1 EDMA 的结构图

串行的传输控制模块主要包含三部分: 队列寄存器组、命令发送模块和数据缓存模块。具体结构如图 2 所示, 队列寄存器组对传输请求队列提交的传输请求进行处理, 提取出源地址、目的地址、传输计数和其他传输控制信息, 生成读和写传输请求, 经过仲裁后, 将满足条件的传输请求提交给命令发送模块进行数据传输, 在传输过程中, 队列寄存器组维护传输参数的更新。命令发送模块收到读请求后, 当源端口不忙时, 向源端口发读命令, 缓存模块接收从源端口读取的数据, 当数据全部接收完毕且目的端口不忙时, 命令发送模块向目的端口发写命令和数据。

在串行的传输控制机制中, 读和写串行工作, 每个数据的传输都要经过读阶段和写阶段两步, 至少需要 3 拍(读阶段包括发读命令和接收数据, 最少需要 2 拍; 写阶段主要是发写命令和数据, 最少需要 1

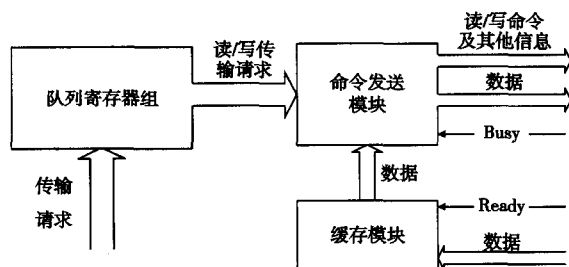


图 2 串行的传输控制模块的结构图

拍)。所以, 要实现 n 个数据的传输, 最少需要 $3n$ 拍才能完成。显然, 这种低效率的串行传输机制已成为 DMA 数据传输速度提高的瓶颈。

3 并行传输机制的设计

对串行传输机制的分析可以看出, 要提高数据传输的速度, 必须在数据传输协议和传输控制结构两个方面都做出改进。

在传输协议方面, 本文的改进主要有两点: 一是增加了突发(Burst)传输协议, 在突发传输时, 只需向源端口发一次读命令, 即可读取 BurstCount 个数据(BurstCount 为一次突发传输的数据个数, 大小由端口决定, 并作为传输协议固定不变), 既节省了开销又提高了传输速度; 二是增大位宽, 支持双字(64 位)传输, 本文在设计中, 主要将内存、外存和 L2 cache 的接口扩展至 64 位, 这三者间的数据交互最为频繁, 增大位宽可以使传输速度加倍。

在传输控制结构方面, 为了提高读和写的并行度, 需要将命令发送模块、缓存模块和读写总线分开。如图 3 所示, 本文设计的传输控制结构主要由队列寄存器组(共有四个: Registers0~3, 优先级由高到低, 与传输请求仲裁及多队列模块中的四个队列一一对应)、源 Pipeline、目的 Pipeline、读命令缓冲区(共有七个, 每个端口各有一个)、写命令缓冲区(共有七个, 每个端口各有一个)、命令仲裁模块、Routing(数据传输旁路)模块七部分组成。

本文的传输机制, 实现了三个层次的并行:

(1) 四个队列寄存器组并行工作: 分别将来自对应队列的传输请求分为读请求和写请求并行处理。

(2) 源 Pipeline 和目的 Pipeline 并行工作: 源 Pipeline 对读请求进行仲裁后送入对应的读命令缓冲区; 目的 Pipeline 对写请求进行仲裁后送入对应的写命令缓冲区。

(3) 读命令缓冲区和写命令缓冲区并行工作:

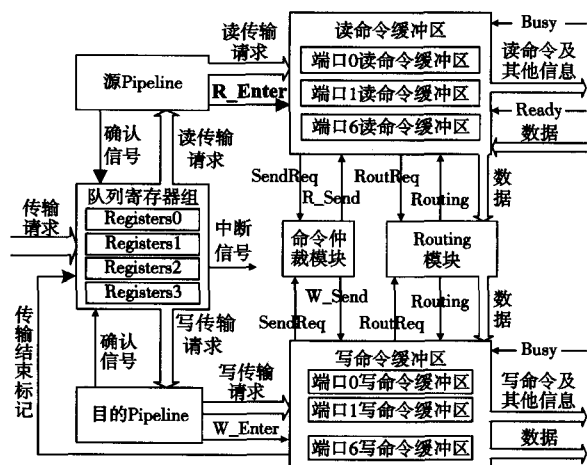


图 3 并行传输控制结构的结构图

每个读命令缓冲区内有一条读流水线,每个写命令缓冲区内有一条写流水线,读请求进入读流水线执行读操作,写请求进入写流水线执行写操作,所有的读流水线和写流水线并行执行。

此外,命令仲裁模块负责对所有读写命令进行仲裁,分配读写总线. Routing 模块相当于数据旁路,将读流水线从源端口读取的数据发给对应的写流水线写往目的端口.

本文的并行传输机制虽然增加了一套总线和控制复杂度,但通过多个层次的并行,实现了读写并发,充分利用了资源,加快了传输请求的响应速度,增大了数据传输的吞吐率,提高了传输的速度和效率。

3.1 队列寄存器组

四个队列寄存器组与多队列模块中的四个队列一一对应,对每个队列提交的传输请求,都有一组寄存器进行实际控制.四个寄存器组的优先级不同,但结构一致,功能相同,可以并行工作,主要任务如下:

(1) 从传输请求中提取出源和目的地址、传输计数、数据大小及地址刷新模式等参数,并根据源和目的端口确定传输位宽及一次传输的数据个数,分别生成读和写传输请求。

(2) 若读传输计数非零,则向源 Pipeline 发读传输请求,当收到确认信号时,表明该传输请求已发往读命令缓冲区,则更新源地址和读传输计数,并生成下一个读传输请求;若读传输计数为零,则等待传输结束。

(3) 若写传输计数非零,则向目的 Pipeline 发写传输请求,当收到确认信号时,表明该传输请求已发往写命令缓冲区,则更新目的地址和写传输计数,

并生成下一个写传输请求;若写传输计数为零,则等待传输结束.

(4) 当收到来自写命令缓冲区的传输结束标记时,表明当前传输已完成,发中断信号并清零相关参数.若传输请求队列中仍有待处理的传输请求,则队列寄存器组接收并处理新的请求.

队列寄存器组是数据传输的总体控制模块,将读传输和写传输分开控制,是实现读写并发的前提。此外,四个队列寄存器组并行工作,可以同时处理四种不同优先级的传输请求,既加快了对各端口传输请求的响应速度,也保证了命令缓冲区、读写总线等资源得到充分利用,提高了数据传输的效率。

3.2 源和目的 Pipeline

源和目的 Pipeline 是由组合逻辑实现的传输请求仲裁机制和传输请求通路组成,负责对四个寄存器组提交的传输请求进行仲裁,将满足条件的传输请求发给对应的命令缓冲区进行数据传输.

源 Pipeline 的仲裁机制为:判断所有读传输请求的源端口对应的读命令缓冲区是否有空闲位置,若有,将该读请求发往对应的读命令缓冲区(如果同时有多个传输请求要进入同一个读命令缓冲区,则高优先级的传输请求优先发送),并向该读请求的队列寄存器组发确认信号;否则,该传输请求继续等待.目的 Pipeline 的仲裁机制的原理与源 Pipeline 相同.

源和目的 Pipeline 的设计是为了将繁琐的传输请求仲裁工作从队列寄存器组中分离出来,简化了队列寄存器组的设计,降低了传输请求仲裁的复杂度.并且,源和目的 Pipeline 并行工作,将读和写传输请求分开处理,读传输请求向读命令缓冲区的发送可以同步、超前或滞后于写传输请求向写命令缓冲区的发送.这样的设计可以使读和写命令缓冲区得到充分利用,也掩盖了读和写传输请求的准备时间,提高了传输请求的处理速度.

3.3 读和写命令缓冲区

每个端口在 EDMA^[4]中均有高速的循环 FIFO 结构的读命令缓冲区和写命令缓冲区,结构如图 4 所示.

读命令缓冲区由 m 个 Cmd_Buffer(称为命令及数据缓存, m 的值由该命令缓冲区对应的端口决定)和四个指针(REnter:指向待接收读命令的 Cmd_Buffer, RSend:指向待发送读命令的 Cmd_Buffer, RReady:指向待接收读取的数据的 Cmd_Buffer, RRout:指向待 Routing 读取的数据的 Cmd_Buff-

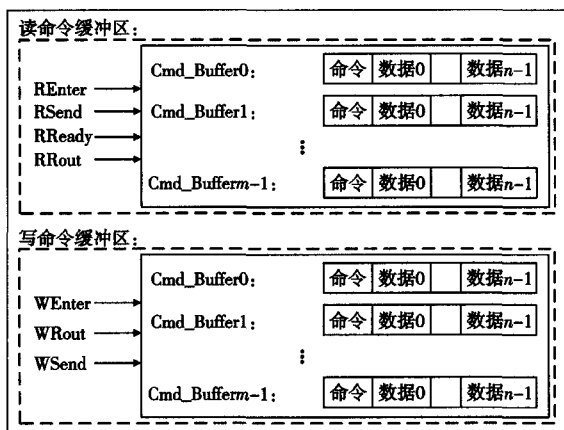


图4 读和写命令缓冲区的结构图

er)组成,主要负责从源 Pipeline 接收读请求,生成读命令送入读流水线进行读操作,并将读取的数据经 Routing 模块发往写命令缓冲区。

写命令缓冲区由 m 个 Cmd_Buffer 和三个指针 (WEnter: 指向待接收写命令的 Cmd_Buffer, WRout: 指向待接收 Routing 的数据的 Cmd_Buffer, WSend: 指向待发送写命令的 Cmd_Buffer) 组成,主要负责从目的 Pipeline 接收写请求,生成写命令送入写流水线进行写操作,要写往目的端口的数据从读命令缓冲区获得,当最后一条写命令发送完毕后,向队列寄存器组发传输结束标记。

指针的设计是为了保证命令在缓冲区内能够按流水线方式有序执行,每个指针在初始时均指向 Cmd_Buffer0,采用循环方式,指针加 1,则自动指向下一个 Cmd_Buffer。

读命令缓冲区内读操作流水线的主要步骤:

(1) 当收到来自源 Pipeline 的 R_Enter(读请求允许进入)信号时,接收读传输请求参数,将生成的读命令存至 REnter 指向的 Cmd_Buffer,并向命令仲裁模块发 SendReq(请求发送命令)信号,REnter 加 1。

(2) 当收到来自命令仲裁模块的 R_Send(读命令可以发送)信号时,将 RSend 指向的 Cmd_Buffer 中的读命令发往源端口,命令发送完毕,RSend 加 1。

(3) 当收到来自源端口的 Ready(数据准备完毕)信号时,接收数据存至 RReady 指向的 Cmd_Buffer,并向 Routing 模块发 RoutReq(请求 Routing 数据)信号,数据接收完毕,RReady 加 1。

(4) 当收到来自 Routing 模块的 Routing 信号时,将 RRout 指向的 Cmd_Buffer 中的数据发往写

命令缓冲区,数据发送完毕,RRout 加 1。

写命令缓冲区内写操作流水线的主要步骤:

(1) 当收到来自目的 Pipeline 的 W_Enter(写请求允许进入)信号时,接收写传输请求参数,将生成的写命令存至 WEnter 指向的 Cmd_Buffer,并向 Routing 模块发 RoutReq 信号,WEnter 加 1。

(2) 当收到来自 Routing 模块的 Routing 信号时,从读命令缓冲区接收数据存至 WRout 指向的 Cmd_Buffer,并向命令仲裁模块发 SendReq 信号,数据接收完毕,WRout 加 1。

(3) 当收到来自命令仲裁模块的 W_Send(写命令可以发送)信号时,将 WSend 指向的 Cmd_Buffer 中的写命令和数据发往目的端口,命令发送完毕,WSend 加 1。

将读写命令缓冲区分开是为了提高读和写的并行度,在读操作流水线中,只要有数据返回,就可以向 Routing 模块发 RoutReq 信号,当收到 Routing 信号时,将数据送往写操作流水线写往目的端口,即读和写可以同时进行,写的时间被掩盖,提高了传输效率。并且,所有的读流水线和写流水线均可以并行执行,成倍的提高了数据传输的速度和吞吐率。

3.4 命令仲裁模块

命令仲裁模块由组合逻辑实现的读写命令仲裁机制组成。当收到 SendReq 信号时,即对所有待发送命令(读命令到达 Cmd_Buffer 即为待发送命令,写命令及数据均到达 Cmd_Buffer 才为待发送命令)进行仲裁,将总线使用权交给满足条件的命令进行数据传输^[5]。

命令仲裁模块的仲裁步骤为:先判断所有待发送命令的端口是否空闲,若空闲,则为该命令置预发送标记;否则,该命令继续等待。再对所有预发送的命令按照高优先级优先(优先级最高的命令优先发送)和写优先(同一优先级的读写命令访问同一端口时,写命令优先发送)两个原则进行比较,将读总线使用权交给满足条件的读命令,并向该读命令所在的读命令缓冲区发 R_Send 信号;将写总线使用权交给满足条件的写命令,并向该写命令所在的写命令缓冲区发 W_Send 信号。

将命令仲裁模块单独设计,对所有待发送命令统一仲裁,分配读写总线,既提高了命令仲裁的效率,又保证了数据传输的正确性和有序性。

3.5 Routing 模块

Routing 模块由组合逻辑实现的仲裁机制和数据通路组成。当收到 RoutReq 信号时,即开始仲裁:

当读命令缓冲区中读命令读取的数据已返回,且与该读命令对应的写命令已到达写命令缓冲区,则 Routing 条件满足,向读和写命令缓冲区发 Routing 信号,将数据从读命令缓冲区发往写命令缓冲区。

Routing 模块作为数据旁路,是连接读操作流水线和写操作流水线的桥梁,是实现读写并发的关键,保证读和写流水线能够有序工作,保证整个传输能够高效进行。

4 分析及验证

本文设计的传输控制机制执行一次数据传输的状态机如图 5 所示。

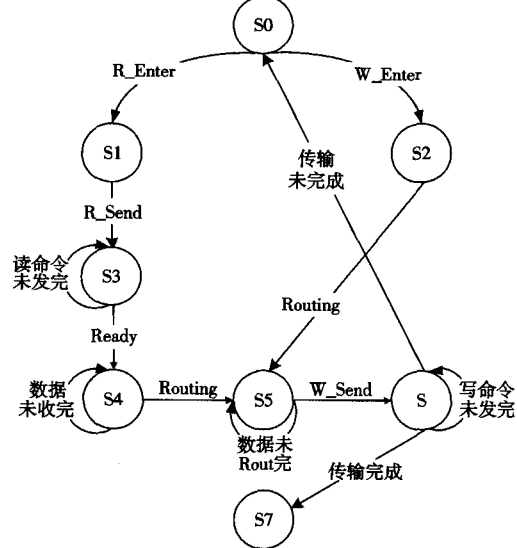


图 5 数据传输的状态机

S0:初始状态,队列寄存器组收到传输请求后,分别生成读和写传输请求,若收到 R_Enter 信号,进入 S1 状态;若收到 W_Enter 信号,则进入 S2 状态。

S1:接收读命令状态,读命令缓冲区通过源 Pipeline 接收读传输请求,生成读命令,并向命令仲裁模块发 SendReq 信号,若收到 R_Send 信号,进入 S3 状态。

S2:接收写命令状态,写命令缓冲区通过目的 Pipeline 接收写传输请求,生成写命令,并向 Routing 模块发 RoutReq 信号,若收到 Routing 信号,进入 S5 状态。

S3:发送读命令状态,向源端口发读命令,若收到 Ready 信号,进入 S4 状态。

S4:接收数据状态,从源端口接收数据,并向 Routing 模块发 RoutReq 信号,若收到 Routing 信号,进入 S5 状态。

S5:Routing 数据状态,写命令缓冲区通过 Routing 模块从读命令缓冲区接收数据,并向命令仲裁模块发 SendReq 信号,若收到 W_Send 信号,进入 S6 状态。

S6:发送写命令和数据状态,向目的端口发写命令和数据,发送完毕,若整个传输请求尚未完成,则进入 S0 状态,继续处理新的请求;否则,进入 S7 状态。

S7:结束状态,整个传输请求已完成,发中断信息并清零相关参数。

由图 5 的状态机可以看出,本文的传输控制逻辑要完成一次 n 个数据的普通传输(非突发且非 64 位位宽传输),最少需要 $n+4$ 拍即可完成,如表 1 所示。

表 1 本文的传输机制传输 n 个数据的节拍数

节拍	1	2	3	...	n	$n+1$	$n+2$	$n+3$	$n+4$
读操作的流水线	a1	a2	a3	...	a_n				
		b1	b2	b_3	...	b_n			
			c1	c_2	c_3	...	c_n		
				d_1	d_2	d_3	...	d_n	
写操作的流水线		e1	e2	e_3	...	e_n			
			f1	f_2	f_3	...	f_n		
				g_1	g_2	g_3	...	g_n	

其中, a_i 表示接收读命令 i , b_i 表示发送读命令 i , c_i 表示接收读命令 i 读取的数据 $Data_i$, d_i 表示 Routing 数据 $Data_i$, e_i 表示接收写命令 i , f_i 表示写命令 i 等待数据 $Data_i$, g_i 表示发送写命令 i 和数据 $Data_i$,与传统传输机制相比,数据传输速度提高了近两倍。此外,本文的传输机制若采用突发传输,则需要的节拍数与普通传输相同,但仅需发送 $(n/BurstCount+n\%BurstCount)$ 个读命令;若采用 64 位位宽的传输,则最少需要 $(\lceil n/2 \rceil+4)$ 拍即可完成,比传统传输机制的速度提高了近五倍。

以上分析了本文的并行传输机制执行一个传输请求的流程和节拍数,而实际上,本文的传输机制可以同时处理多个传输请求,最多可以使七对读写流

流水线同时工作,即七个传输请求并行传输,在传输效率和性能方面都有很大提高。

本文以 NC-Sim 为平台对并发传输控制机制在整个 DSP 系统中的工作情况进行了验证,图 6 中给出了从外存到内存传输 16 个 32 位数据(按 64 位位

宽的突发传输)的验证波形,传输情况与分析结果一致,需要 12(即 $16/2+4$)拍即可完成。经过综合验证表明,本文设计的传输控制机制功能正确,满足了设计需求,实现了流水化和读写并发,数据传输速度比串行机制提高了三倍左右。

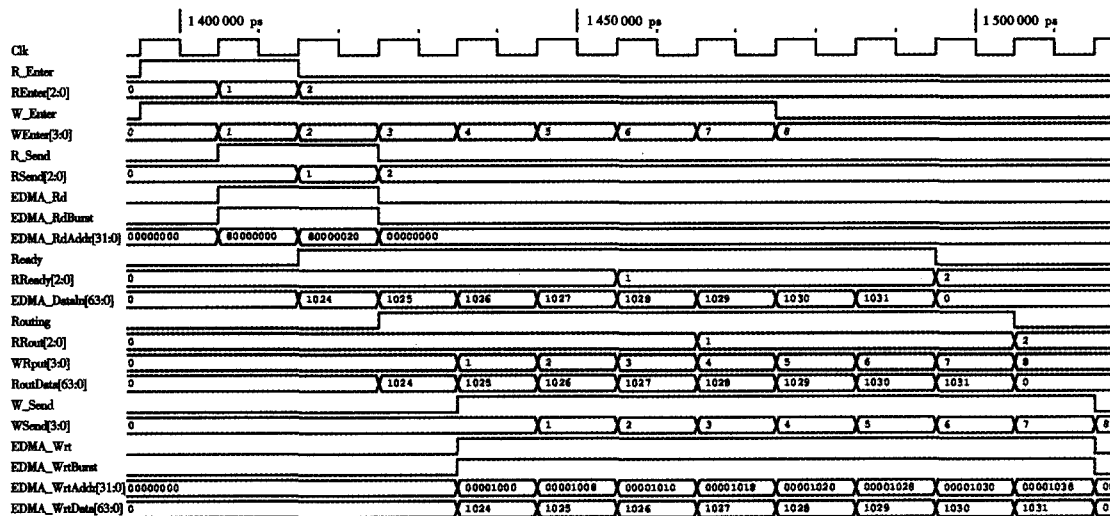


图 6 验证波形

5 结束语

本文针对如何提高 DMA 部件数据传输速度的问题,采用了流水线的思想,设计了支持突发传输、64 位位宽和读写并发的并行传输控制机制,并在我所自主研发的一款高性能 DSP 中得到了实际应用。结果表明,本文的设计将读写请求分开处理,虽然增加了一套总线和控制复杂度,但实现了流水化和读写并发,数据传输速度比传统传输机制提高了三倍,使该款 DSP 芯片的综合性能有较大提升。

参考文献:

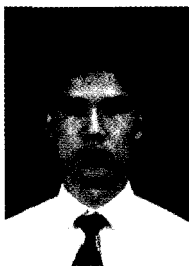
- [1] 薛炜,李广军. 增强型 eDMA 的结构与典型应用[J]. 单片机与嵌入式系统应用,2009(1):12-14.
- [2] 时磊,张铁军. 高速多通道 DMA 控制器的设计与实现[J]. 为计算机应用,2010(5):50-53.
- [3] Texas Instruments Inc. TMS320C6000 DSP enhanced direct memory access (EDMA) controller reference guide[M]. Literature Number: SPRU234C, 2006.
- [4] Alexander P, Kemalov. Algorithms for DMA communications[C] // Proceedings of the 5th international conference on Computer systems and technologies. New York, 2004.

- [5] Grosso M, Perez W J H. Functional test generation for DMA controllers[J]. IEEE Xplore digital library, 2010,11(8):1-6.

作者简介:



余再祥 男,副研究员.研究方向为大规模集成电路和高性能微处理器设计。



陈海波 男,硕士研究生.研究方向为 VLSI 设计。

孙永节 男,研究员.研究方向为超大规模集成电路和微处理器设计。

吴文俊 男.研究方向为 VLSI 设计。