一种高性能的异步 FIFO 结构

刘祥远,陈书明
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 针对现有 FIFO 设计方法的不足,本文提出一种新的异步 FIFO 结构——WG-FIFO,采用加权 Gray 码进行指针编码,采用实时状态检测器控制写/读操作.模拟结果表明,在 FIFO 深度为 4 ~ 16 的情况下,该结构与已有的 FIFO 结构相比在性能、面积开销以及写/读操作效率等方面都获得了明显的改善.特别地,当 FIFO 的深度为 8、宽度为 32 时,相比 B-FIFO,WG-FIFO 的最高时钟频率提高 31.6%,单元面积减少 17.1%,且写/读效率最大能提高 47%.

关键词: 先进先出;高性能;格雷码;异步;同步器
中图分类号: TN47 **文献标识码:** A **文章编号:** 0372-2112 (2007) 11-2098-07

A High-Performance Asynchronous FIFO Architecture

LIU Xiang-yuan, CHEN Shu-ming
(*School of Computer Science, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: To cover the shortages of the existing FIFO architecture, a new FIFO architecture, WG-FIFO, is presented in this paper. The WG-FIFO encodes write/read pointer with a weighted-gray code, and controls write/read operation with a real-time global states detector. Simulation results show that the performance, area cost, and write/read operation efficiency are all considerably improved compared with other available FIFO architectures under the depth range of 4 ~ 16. For example, when the FIFO depth is 8 and the data width is 32, compared with B-FIFO, the highest clock frequency is increased by 31.6%, cell area cost is decreased 17.1%, and write/read efficiency is improved 47% at most.

Key words: first-in first-out(FIFO); high performance; Gray code; asynchronous; synchronizer

1 引言

随着 CMOS 工艺缩小到超深亚微米阶段,当前的 VLSI 设计已进入 SoC 时代.一个 SoC 系统中通常含有多个不同的时钟域,如何在这些不同时钟域之间进行可靠的高性能通信就成为设计人员所面临的一个极具挑战性的问题[1].研究人员提出了异步握手、异步包装以及异步 FIFO 等多种解决方案.

采用异步 FIFO 进行跨时钟域通信是非常有效、可靠的方法,在实际芯片设计中得到了广泛应用[2~9].按照指针编码方式来分,异步 FIFO 通常可分成 Binary 码、Gray 码、Hybrid-Gray 码等方式[2~4],但它们不同程度地存在下列问题:

(1)写/读指针编码转换、计算的延时大,限制了写/读操作频率.

(2)跨时钟域传递写/读指针需要大量的同步器(synchronizer),面积开销大.

(3)由于同步器的延迟,导致 FIFO 的状态检测保守,浪费可用空间.

文献[5,6]用写/读令牌环代替写/读指针,简化了"满"/"空"状态的检测,节省了编码转换逻辑以及同步器的延迟和面积开销,但为防止写/读溢出及死锁,又引入了额外的控制逻辑,并且仍然存在可用空间的浪费.文献[7]结合握手协议和行波流水技术实现了一种异步 FIFO,降低了延迟和功耗,但其纯异步电路实现方式增大了设计的复杂度和应用的难度.

因此,本文提出一种新的异步 FIFO 结构:WG-FIFO,采用加权 Gray 码进行指针编码,采用实时的全局状态检测器来控制写/读操作,克服了现有 FIFO 结构的一些缺陷.WG-FIFO 具有以下优点:

(1)写/读指针可直接进行比较与计算,减少了编码转换的延时,提高了性能.

(2)只传递全局状态,不传递写/读指针,减少了同步器的面积开销.

收稿日期:2006-08-14;修回日期:2007-05-18
基金项目:国家自然科学基金(No.60473079,No.60676010);教育部高等学校博士学科点专项科研基金(No.20059998026)

(3)实时检测“满”/“空”/“将满”/“将空”等全局状态,不会浪费可用空间,能高效地支持 Burst 写/读操作.

文献[9]提出过一种异步 FIFO 结构,本文采用的状态检测思想与其类似,但在指针编码方式上与其具有本质的区别.前者采用常规 Gray 码指针,不支持“将满”/“将空”等状态的检测;而 WG-FIFO 采用加权 Gray 码指针,不仅能直接比较出“满”/“空”状态,还可以方便地计算出“将满”/“将空”状态,从而能高效地支持 Burst 写/读操作.

2　WG-FIFO 结构

2.1　总体结构

每个 FIFO 都包含两个端口:写端口与读端口,它们可以分别工作在不同的时钟域.图 1 给出了本文设计的 WG-FIFO 的总体结构,它包含存储体、写控制器、读控制器以及全局状态检测器四个部分.存储体储存写入 FIFO 的有效数据;写/读控制器接收外部的写/读请求,从存储体中存/取数据,并控制写/读指针的移动;全局状态检测器根据写/读指针的变化,实时检测当前 FIFO 的“满”/“空”/“将满”/“将空”状态.

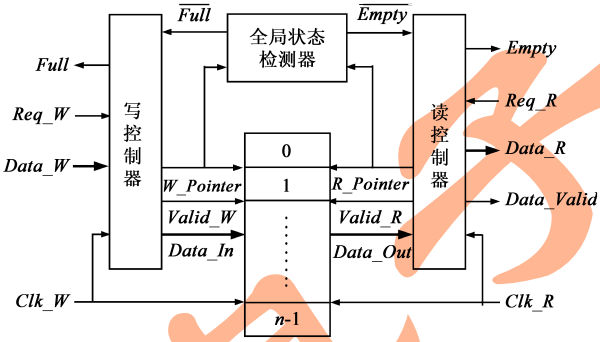


图 1　WG-FIFO 的总体结构

2.2　加权 Gray 码指针

写/读指针的编码通常采用 Binary 码或 Gray 码,但 Binary 码递增时不是一位变化的,需要先转换成 Gray 码才能进行跨时钟域的传递;而 Gray 码及其变种 Hybrid-Gray 码都是无权重的,需要先转换成 Binary 码后才能计算指针的差值.这些额外的码制转换增加了较大的延迟和面积开销,在 FIFO 深度较小的情况下显得成本过高.

为此,我们提出一种加权 Gray 码循环编码:Weighted-Gray 码,并进而提出了新型的异步 FIFO 结构——WG-FIFO.设编码位数为 n,则由自然数 k 计算对应加权 Gray 码 WG_k 的递推公式如(1)所示,它的周期为 $2n$,且每次递增时只有一位发生变化.

$$\begin{aligned} &WG_0 = 0 \\ &WG_k = (WG_{k-1} \ll 1) + \overline{WG_{k-1}[n-1]},\ k = 1,2,\cdots \end{aligned} \tag{1}$$

用 n 位加权 Gray 码来表示[0, $n-1$]中的自然数,则每个自然数都对应了两个按位相反的编码值.由加权 Gray 码 WG 计算其对应自然数 k 的公式如下,

$$k = \begin{cases} \sum_{i=0}^{n-1} WG[i], & \text{if } WG[n-1] = 0 \\ \sum_{i=0}^{n-1} \overline{WG[i]}, & \text{if } WG[n-1] = 1 \end{cases} \tag{2}$$

假设 FIFO 深度为 n,则写/读指针须采用 n 位加权 Gray 编码.图 2 给出了 n 为 8 时的示例,每个单元都对应两个相反的编码值,分别表示指针奇/偶数次经历该单元.写/读指针初始值为“全 0”,任一时刻对两个指针进行位异或运算,就能判断出 FIFO 的当前状态.结果中为 1 的位就表示 FIFO 中的对应单元存在有效数据,有多少位为 1 就有多少个对应的有效数据可读,反之,有多少个 0 就表示有多少个对应的空余单元可写.如果结果为“全 1”,表示写指针比读指针多循环移动了一次,FIFO 处于“满”状态;如果结果为“全 0”,则表示读写指针经历了相同次数的循环移动,FIFO 处于“空”状态.

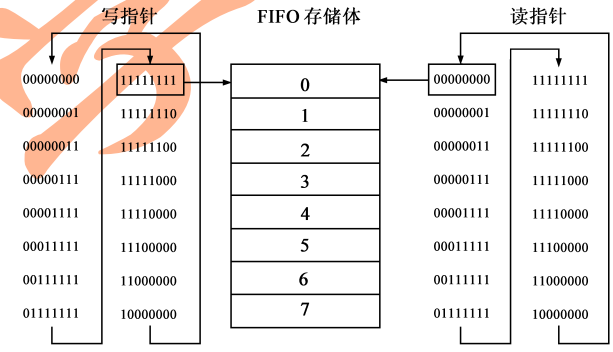


图 2　FIFO 深度为 8 时的加权 Gray 码指针

加权 Gray 码既有 Gray 码每次递增变化一位的特性,又有 Binary 码带权重、容易计算的优点,因此无须转换码制就可以在不同时钟域之间直接进行写/读指针的传递和计算.

2.3　实时全局状态检测器

全局状态检测器根据写/读指针的当前值计算出 FIFO 的当前状态.“满”状态的输出传递到写控制器,而“空”状态的输出传递到读控制器.由于 FIFO 具有高度并发的操作:写/读端口可能在两个不同时钟控制下同时写/读 FIFO,改变 FIFO 的状态.因此,两个全局状态信号(“满”和“空”)的输出必须使用同步器进行同步,分别由写时钟和读时钟控制.

最简单的同步器设计是采用多级同步寄存器,通常两级就够了.然而,同步器所增加的延迟可能导致 FIFO 的“上溢”或“下溢”.例如,当 FIFO 变“空”时,读操作将在两个时钟周期后被停止,而在停止之前的一个时钟周期,读端口可能读走一个无效的“空”单元,发生

“下溢”.

一种保守的解决方法是将“满”/“空”状态的检测分布到写/读控制器中,并将读/写指针经两级同步器同步后分别传递到写/读控制器[2~4].这时,“满”信号是由写指针与延迟后的读指针进行比较产生的,而“空”信号是由读指针与延迟后的写指针进行比较产生的,因此,FIFO 的“满”/“空”判断总是保守的.写控制器可能在 FIFO 还有一定空间时停止写数据,但不会在 FIFO 已满的情况下继续写数据;读控制器可能在 FIFO 还有有效数据时停止读数据,但不会在 FIFO 已经空的情况下继续读数据.这种保守的状态判断会损失 FIFO 空间的可用性,对于容量较小的 FIFO 而言是无法接受的.例如,一个深度为 8 的 FIFO,最坏情况下,写控制器判断 FIFO 为“满”时,实际上是 3 个周期前的状态,如果这 3 个周期内都有读操作发生,则 FIFO 里实际只存有 5 个有效数据.

另一种方法是改变 FIFO 的“满”/“空”定义[5],即预测 FIFO 的“满”/“空”.当 FIFO 中只剩下一个或没有空单元时认为是“满”,而只有一个或没有有效单元时认为是“空”.提前预测“空”状态在某些情况下,可能发生 FIFO 含有一个有效数据,但读请求仍然被阻塞的情况.为此,论文[5]还采用了复杂的“Bi-modal”检测机制来防止这种死锁的发生,增加了额外的开销.

上述问题的产生归根结底就在于同步器的延迟使写/读控制器不能及时获知 FIFO 的“满”/“空”状态.

FIFO 的“满”/“空”状态是由读、写两个指针经过比较后产生的,但分别只被写/读控制器读取.我们注意到:只有读指针的变化才可能使 FIFO 进入“空”状态,而写指针的变化只可能使 FIFO 退出“空”状态.换句话说,就是“空”状态的进入实际上与读时钟是同步的,而“空”状态的退出与读时钟是异步的.因此,读时钟控制的同步器需要同步的是“空”状态的退出,而不是“空”状态的进入.同样,“满”状态也存在类似的情况.

根据上述现象,本文设计了图 3 所示的全局状态检测器.读、写指针经位异或运算后产生 FIFO 的状态标识(FS),将其分别与“全 0”和“全 1”进行相等比较,实时产生一个“空”信号 $\overline{True\_Empty}$ 和一个“满”信号 $\overline{True\_Full}$.再将这两个信号分别连接到各自同步器的异步复位端,则可以产生一个与读时钟同步的“空”状态 $\overline{Empty}$ 以及一个与写时钟同步的“满”状态 $\overline{Full}$.

当 FIFO 进入“空”状态时($\overline{True\_Empty}$ 变为 0),读控制器能够马上获知该状态($\overline{Empty}$ 变为 0),从而在下一个读时钟周期就能够停止读操作;而当 FIFO 退出“空”状态时($\overline{True\_Empty}$ 变为 1),读控制器在两个读时钟周期后才能获知新状态($\overline{Empty}$ 变为 1),然后再在下一个读时钟周期才开始允许读操作.“满”状态 $\overline{Full}$ 的检测类似.因此,这个实时的全局状态检测机制既能及时防止 FIFO 的“上溢”/“下溢”,又不会损失可用空间,从而使异步 FIFO 能够高效地支持连续写/读操作.

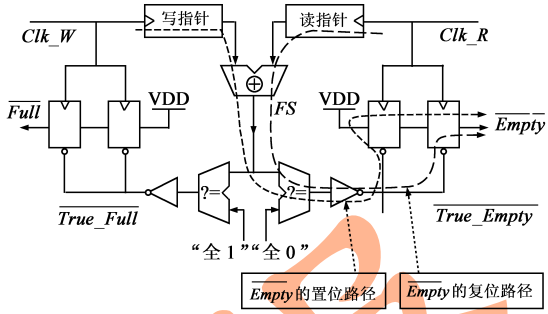


图 3 全局状态检测器

2.4 亚稳态的消除

两个时钟域之间的信号传递总是存在一定的风险,比如信号变成亚稳态.本节以“空”状态 $\overline{Empty}$ 为例,来分析图 3 中的同步机制是如何消除亚稳态的.

由前文分析可知,只有读指针的变化才可能使 $\overline{Empty}$ 变为 0,而写指针的变化只能使其变为 1,所以“空”状态 $\overline{Empty}$ 的产生存在复位与置位两条路径,如图 3 中虚线所示.在将 $\overline{Empty}$ 复位时,异步复位信号 $\overline{True\_Empty}$ 的下跳沿变化与读指针的变化同步,所以 $\overline{True\_Empty}$ 的下跳沿与读时钟边沿存在一个固定的组合逻辑延时,可以认为它与读时钟是同步的.在置位 $\overline{Empty}$ 时,信号 $\overline{True\_Empty}$ 的上跳沿变化与写指针的变化同步,所以它对于读时钟而言是异步的.此时两级同步寄存器开始起作用,由于第二级同步寄存器的输入输出均为 0,不管 $\overline{True\_Empty}$ 信号的上跳沿与读时钟上升沿的相位关系如何,输出信号 $\overline{Empty}$ 都不会产生亚稳态,维持为 0[10],经过两个读时钟周期后,$\overline{Empty}$ 才会被同步置 1.可见,$\overline{Empty}$ 的变化总是与读时钟同步的,并且不论信号是 0 还是 1,都至少会维持一个读时钟节拍.因此,只要 $\overline{Empty}$ 的复位路径以及后续组合逻辑的总延迟满足读时钟的单周期时序约束,就能保证读端口不出现亚稳态问题.

同样,“满”状态 $\overline{Empty}$ 的变化总是与写时钟同步,因此写端口也不会存在亚稳态问题.

2.5 毛刺的过滤

全局状态信号的产生采用了异步逻辑,如果同步器复位信号上的负脉冲宽度达到寄存器异步复位的最小脉冲宽度要求,将导致该状态信号被复位.数字系统中的信号很容易受噪声影响而产生毛刺,因此必须对复位信号线上的毛刺进行过滤,以防止全局状态信号被错误复位.

由于同步器复位信号上的高电平需要保持两个时

钟周期以上才能使输出信号由 0 变为 1,所以窄的正脉冲毛刺不会对全局状态信号的产生造成任何影响.真正有害的是负脉冲毛刺,本章采用一个由延迟单元和 De Morgan 等价或门构成的过滤电路[10]来过滤同步器复位信号上的窄负脉冲,如图 4 所示.

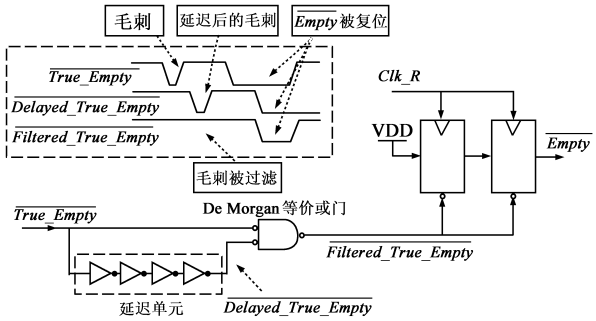


图 4　毛刺的过滤电路

过滤电路在过滤掉有害负脉冲毛刺的同时,也可能会过滤掉由 FIFO 读写指针变化所产生的正常窄负脉冲.例如,FIFO 中只有 1 个有效数据时,读、写操作都是允许的,且信号 $\overline{True\_Empty}$ 和 $\overline{Empty}$ 均为 1.如果读端口先来一个读操作,则读指针追上写指针,$\overline{True\_Empty}$ 变为 0;紧接着写端口来一个写操作,则写指针又超过读指针,$\overline{True\_Empty}$ 又变回 1.正常情况下,状态检测器会先把 FIFO 状态 $\overline{Empty}$ 复位为 0,并阻塞读请求,过两个读时钟周期后再把 $\overline{Empty}$ 置为 1,允许读请求来读取该数据.但是,如果这个读后写操作发生的间隔很小,则 $\overline{True\_Empty}$ 信号上的负脉冲宽度就会很窄,当它小于过滤电路中延迟单元所限定的宽度时,就会被当作毛刺过滤掉.幸运的是,将这种正常的窄负脉冲过滤掉不但不影响 FIFO 的读写控制,反而还能够提高读端口的性能,因为此时 FIFO 中已经有一个新数据可读,所以没必要阻塞后续的读请求.

同样,也采用类似的过滤电路来过滤 $\overline{True\_Full}$ 信号上的窄负脉冲.

2.6　“将满”/“将空”状态的检测

采用 Burst 传输可以显著提高系统的通信效率,所以 FIFO 的状态检测器往往需要产生“将满”/“将空”状态信号来表征 FIFO 还能支持的 Burst 写/读操作个数.

通常大多数嵌入式微处理器的 DMA 都支持个数为 4 的 Burst 写/读操作.因此,我们定义“将满”状态表示 FIFO 中还剩下不到 4 个空余单元,不允许写端口再往 FIFO 连续写 4 个(或 4 个以上)数据以防发生阻塞;“将空”状态表示 FIFO 中还剩下不到 4 个有效数据,不允许读端口再从 FIFO 连续读 4 个(或 4 个以上)数据以防发生阻塞.

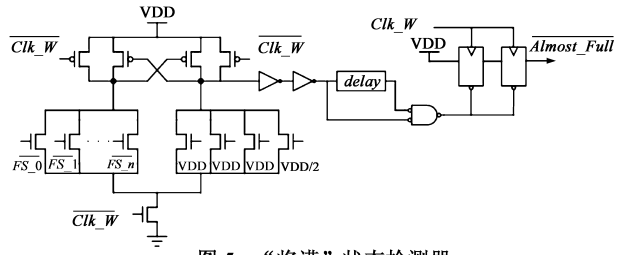


图 5　“将满”状态检测器

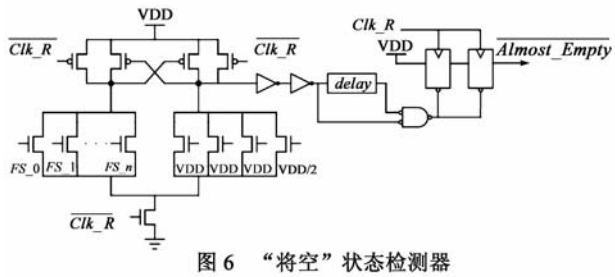
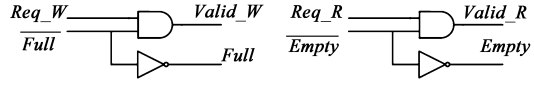


图 6　“将空”状态检测器

根据加权 Gray 码的特性,“将满”状态 $\overline{Almost\_Full}$ 和“将空”状态 $\overline{Almost\_Empty}$ 可分别用图 5 和图 6 中的预充比较电路及同步器来产生.如果图 3 的 *FS* 信号中 0 的个数小于 4,则 $\overline{Almost\_Full}$ 为 0,否则为 1;如果 *FS* 中 1 的个数小于 4,则 $\overline{Almost\_Empty}$ 为 0,否则为 1.它们的同步与过滤机制与“满”/“空”状态相同,不再赘述.

2.7　写/读控制器

如图 7 所示,写/读控制器根据 FIFO 是否处于“满”/“空”状态来过滤外部的写/读数据请求:写控制器通常使能写请求,但 FIFO 为“满”时使其失效;读控制器通常使能读请求,但 FIFO 为“空”时将其阻塞.



(*a*)写控制器　　(*b*)读控制器

图 7　写/读控制器

2.8　写/读时序

FIFO 的写时序波形如图 8 所示,FIFO 中还剩 4 个空余单元,而写端口连续不断的发出写请求,则连续写完 4 个单元后,FIFO 马上进入“满”状态并将写请求阻塞.接着,读端口读走一个有效数据,FIFO 在两个写时钟周期后才退出“满”状态,并开始允许下一次写请求.如果FIFO的写指针与读指针变化的间隔非常小,则产

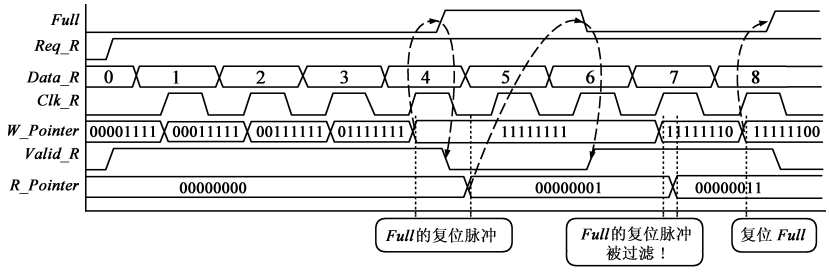


图 8　WG-FIFO 的写时序

生的复位负脉冲会被过滤电路过滤掉，FIFO 就不会进入“满”状态，并允许下一次的写请求.在写控制器及全局状态检测器的控制下，写端口正确的向 FIFO 写入了 6 个数据单元.

类似的，读时序如图 9 所示.由此可见，WG-FIFO 能够高效地支持连续的写/读请求，并且不会产生“上溢”/“下溢”以及同步错误.

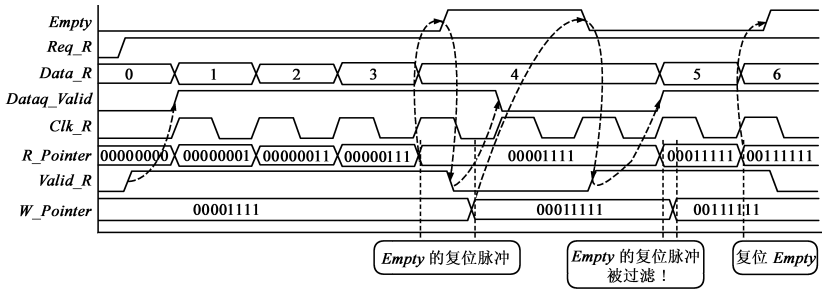


图 9 WG-FIFO 的读时序

3 实验与比较

为了验证本文提出的 FIFO 设计方法，本文在 1.8V 0.18μm CMOS 工艺下进行了模拟.我们用 Verilog 语言设计描述了五种不同 FIFO：Binary 码 FIFO (B-FIFO)[3]、Gray 码 FIFO (G-FIFO)[2]、Hybrid-Gray 码 FIFO (HG-FIFO)[4]、加权 Gray 码 FIFO (WG-FIFO)以及采用令牌环代替指针的 FIFO (TR-FIFO)[5]，并且比较了它们在不同位宽、深度条件下的性能、面积以及写/读效率.需指出的是，WG-FIFO 的“将满”/“将空”检测器中的比较电路是用全定制方法实现的.

3.1 性能

在相同的时序约束条件下，对这五种 FIFO 进行了自动综合，然后用静态时序分析工具分析了最坏 PVT 条件下所能达到的最高写/读时钟频率，如表 1 所示.

表 1 最坏 PVT 条件下不同 FIFO 的最高写/读时钟频率

数据位宽	FIFO 深度	最高写/读时钟频率(MHz)				
		B-FIFO	G-FIFO	HG-FIFO	TR-FIFO	WG-FIFO
8	4	640	621	575	709	763
	8	575	467	565	645	654
	16	538	413	565	534	606
	32	493	391	498	455	565
16	4	645	549	602	690	725
	8	549	446	595	641	685
	16	535	405	568	521	617
	32	457	402	532	390	529
32	4	633	529	595	684	719
	8	493	431	565	613	649
	16	495	417	529	543	575
	32	420	348	508	387	515

从表 1 可以看出，由于 WG-FIFO 无须转换指针编码就能计算出全局状态，所以它的写/读时钟频率比其他 FIFO 更高，并且 FIFO 深度越小，所能达到的时钟频率越高.特别地，当深度为 8、位宽为 32 时，WG-FIFO 的最高写/读时钟频率达到 649MHz，相比 B-FIFO、G-FIFO、HG-FIFO 和 TR-FIFO 分别提高了 31.6%、50.6%、14.9%和 5.9%.

需要指出的是，B-FIFO、G-FIFO、HG-FIFO 及 WG-FIFO 各自的读/写性能较为平衡，写/读时钟的频率相差很小；而 TR-FIFO 读端口的数据输出采用三态总线设计，随着深度增大性能下降很快，因此它的性能受制于读端口，表 1 中列出的是它的读时钟频率.

3.2 面积

根据综合出的逻辑网表，我们统计并比较了五种 FIFO 的面积开销，如表 2 所示.

表 2 不同 FIFO 的单元面积开销

数据位宽	FIFO 深度	单元面积开销($\times 10^4 \mu m^2$)				
		B-FIFO	G-FIFO	HG-FIFO	TR-FIFO	WG-FIFO
8	4	0.94	0.83	0.80	0.71	0.77
	8	1.61	1.50	1.51	1.73	1.59
	16	2.99	2.47	2.80	3.42	3.01
	32	4.79	4.39	5.07	7.17	5.97
16	4	1.44	1.42	1.20	1.23	1.25
	8	2.74	2.32	2.42	2.64	2.54
	16	4.93	4.21	4.89	4.86	4.74
	32	8.43	7.63	9.30	10.8	9.19
32	4	2.68	2.39	2.31	2.15	2.17
	8	5.02	4.19	4.12	5.19	4.16
	16	9.32	7.49	8.43	9.18	8.32
	32	15.8	14.3	15.7	18.3	16.4

WG-FIFO 的指针编码长度与 FIFO 深度 n 相同，而 B-FIFO、G-FIFO 及 HG-FIFO 的指针长度仅为 $\log_2 n + 1$；但前者无需采用多级同步器来传递写/读指针，所以在 n 较小的情况下，指针编码所引起的面积增加可以通过同步器的面积节省而获得补偿.从表 2 可以看出，FIFO 深度为 4 的时候，WG-FIFO 的面积开销最小，随着深度增大到 32，它的面积开销逐渐超过其他三种 FIFO.但在深度不大于 16 的情况下，WG-FIFO 的面积开销仍然要比 B-FIFO 的面积开销小.

TR-FIFO 的令牌长度也与 FIFO 深度相同，但其读端口的数据输出采用三态总线，其负载随深度的增大而增加，所需三态驱动门的面积也急剧增大.因此，深度较小($n < 8$)时，TR-FIFO 的面积开销与 WG-FIFO 相当，但深度较大($n > 8$)时，它的面积开销要比 WG-FIFO 大得多.

特别地，当深度为 8、位宽为 32 时，WG-FIFO 的单元面积开销为 41600μm^2，相对于 B-FIFO、G-FIFO、HG-FIFO 和 TR-FIFO 分别减少了 17.1%、0.7%、－0.9%和

19.8%.

3.3 效率

最高工作频率并不能完全反映 FIFO 的真实性能,因此,我们给出一个与频率大小无关的衡量指标,即在 Burst 传输情况下,平均每个写/读时钟周期执行的合法写/读操作的个数(Operations per Cycle,简称为 OPC).OPC 反映了 FIFO 写/读操作的实际效率,理想情况下它的值为 1,但实际上 FIFO 的写/读操作受制于较慢的一方,当读端口的时钟频率高于写端口时,FIFO 的 OPC 由写端口决定,反之,OPC 由读端口决定.

通过模拟,我们比较了这些 FIFO 在"将满"/"将空"状态控制下的 OPC.G-FIFO 和 HG-FIFO 只支持深度为 2^m 的设计,并且与 B-FIFO 具有相同的 OPC;而文献[5]中没有给出 TR-FIFO 的"将满"/"将空"状态检测机制.所以,表 3 只列出了 B-FIFO 及 WG-FIFO 的 OPC 比较结果.从表 3 可以看出,由于 WG-FIFO 的状态检测及时,没有同步延迟,没有空间损失,所以它的 OPC 更高.

当深度为 4 时,"将满"/"将空"状态实际上对应"空"/"满"状态,B-FIFO 和 WG-FIFO 都要等到 FIFO "满"/"空"时才能允许发出 Burst 读/写 4 个数据单元的操作,读、写是交替进行的,因此 OPC 达不到 1,但由于 WG-FIFO 的状态检测比 B-FIFO 快,所以它的 OPC 要比后者更高.

当深度为 8 时,WG-FIFO 看到的可用单元数为 8,可以不间断的支持个数为 4 的 Burst 写操作,读、写是并发的,每个写时钟周期都能执行 1 个合法的写操作,OPC 达到 1.而 B-FIFO 只能看到 6 个可用单元,所以经常停止发出后续的 Burst 写请求,当读、写时钟同频时,OPC 达到 0.68 的最低值.提高读时钟频率能弥补空间损失,当读时钟频率提高到写时钟的 2 倍时,B-FIFO 的 OPC 才能达到 1.

当 FIFO 深度大于 10 之后,B-FIFO 的空间损失不会影响连续的 Burst 写/读操作,它与 WG-FIFO 的 OPC 均为 1.

特别地,在 FIFO 的读、写时钟同频且深度分别为 4、6、8 和 10 的情况下,WG-FIFO 的 OPC 比 B-FIFO 的 OPC 分别提高了 46%、84%、47%和 19%.

综上所述,在 FIFO 深度较小的情况下,WG-FIFO 的 OPC 比其他 FIFO 都更高.当 FIFO 性能受制于读端口时,OPC 的结果与表 3 类似,不再赘述.

表 3　性能受制于写端口时不同 FIFO 的 OPC

读、写时钟周期的比值	FIFO 性能受制于写端口时的 OPC(Operations per Cycle)														
	深度 = 4			深度 = 6			深度 = 8			深度 = 10			深度 > 10		
	B-FIFO	WG-FIFO	提高	B-FIFO	WG-FIFO	提高	B-FIFO	WG-FIFO	提高	B-FIFO	WG-FIFO	提高	B-FIFO	WG-FIFO	提高
0.2	0.52	0.68	31%	0.8	1	25%	1	1	0	1	1	0	1	1	0
0.3	0.52	0.68	31%	0.68	1	47%	1	1	0	1	1	0	1	1	0
0.4	0.52	0.68	31%	0.68	1	47%	1	1	0	1	1	0	1	1	0
0.5	0.52	0.68	31%	0.64	1	56%	1	1	0	1	1	0	1	1	0
0.6	0.52	0.68	31%	0.56	1	79%	0.9	1	11%	1	1	0	1	1	0
0.7	0.45	0.59	31%	0.56	1	79%	0.87	1	15%	0.98	1	2%	1	1	0
0.8	0.42	0.58	38%	0.56	1	79%	0.8	1	25%	0.98	1	2%	1	1	0
0.9	0.41	0.57	39%	0.52	0.92	77%	0.78	1	28%	0.92	1	9%	1	1	0
1	0.35	0.51	46%	0.44	0.81	84%	0.68	1	47%	0.84	1	19%	1	1	0

3.4 缺点

相对于其他 FIFO,WG-FIFO 在深度较小($4 \leqslant n \leqslant 16$)的情况下具有较强的优势,但还存在下列缺点需要改进:

(1)全局状态的复位路径及其后续组合路径的总延时必须满足读/写时钟域的时序约束,这在一定程度上限制了最高读/写时钟频率.

(2)由于加权 Gray 码指针位数与 FIFO 深度相同,深度较大($n > 16$)时面积开销大,且性能下降.

(3)加权 Gray 码指针不能直接作为存储器地址,如果存储体为存储器类型,则必须将其转换成 Binary 码才能进行存储器的读、写操作.

4 结束语

本文提出一种新型的异步 FIFO 设计结构——WG-FIFO,采用加权 Gray 码指针编码以及实时全局状态检测机制,具有简洁、可靠及高效的特点.模拟结果表明,与已有的其他几种 FIFO 结构相比,在 FIFO 深度为 4 ~ 16 的情况下,WG-FIFO 在性能、面积开销以及 OPC 等方面都获得了较大的改善,是一种高性能的异步 FIFO 结构.特别地,当 FIFO 的深度为 8、宽度为 32 时,相比 B-FIFO、G-FIFO、HG-FIFO 和 TR-FIFO,WG-FIFO 的最高时钟频率分别提高了 31.6%、50.6%、14.9%和 5.9%,单元面积分别减少了 17.1%、0.7%、-0.9%和 19.8%,并且在读、写时钟同频时,OPC 比 B-FIFO、G-FIFO 及 HG-FIFO 提高了 47%.

参考文献:

[1] Clifford E Cummings. Synthesis and scripting techniques for

designing multi-asynchronous clock designs[A]. Proceedings of Synopsys Users Group Conference[C]. Boston, USA, 2001.
[2] Clifford E Cummings. Simulation and synthesis techniques for asynchronous FIFO design[A]. Proceedings of Synopsys Users Group Conference[C]. San Jose, USA, 2002.
[3] 汪东,马剑武,陈书明. 基于 Gray 码的异步 FIFO 接口技术及其应用[J]. 计算机工程与科学,2005,27(1):58 – 60.
WANG Dong, MA Jian-wu, CHEN Shu-ming. The interface technology of asynchronous FIFOs based on gray code and its application[J]. Computer Engineering & Science, 2005, 27(1): 58 – 60. (in Chinese)
[4] Edward Paluch. Synthesis optimized universal synchronous/asynchronous generic FIFO design[A]. Proceedings of Synopsys Users Group Conference[C]. San Jose, USA, 2003.
[5] Tiberiu Chelcea, Steven M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols[A]. Proceedings of the 38th Design Automation Conference[C]. Las Vegas, Nevada, USA, 2001. 21 – 26.
[6] Xin Wang, Tapani Ahonen, Jari Nurmi. A synthesizable RTL design of asynchronous FIFO[A]. Proceedings of International Symposium on System-on-Chip[C]. Tampere, Finland, 2004. 123 – 128.
[7] Jeong-Gun Lee, Suk-Jin Kim, Kiseon Kim. Handshake-wave combined approach with runtime reconfiguration for designing a low latency asynchronous FIFO[A]. Proceedings of IEEE Asia-Pacific ASIC Conference[C]. Fukuoka, Japan, 2004, 2004. 188 – 191.
[8] Kim D, Kim M, Sobelman G E. Asynchronous FIFO interfaces for GALS on-chip switched networks[A]. Proceedings of International SoC Design Conference[C]. Seoul, Korea, 2005. 186 – 189.
[9] Clifford E Cummings, Peter Alfke. Simulation and synthesis techniques for asynchronous FIFO design with asynchronous pointer comparisons[A]. Proceedings of Synopsys Users Group Conference[C]. San Jose, USA, 2002.
[10] Clifford E Cummings, Don Mills, Steve Golson. Asynchronous & synchronous reset design techniques[A]. Proceedings of Synopsys Users Group Conference[C]. San Jose, USA, 2003.

作者简介:

刘祥远 男,1977 年生于江西会昌,1999 年毕业于国防科技大学计算机系,获学士学位,2002 年毕业于国防科技大学计算机学院,获工学硕士学位,2007 年毕业于国防科技大学计算机学院,获工学博士学位,主要研究方向为超深亚微米 VLSI 设计理论与技术.
E-mail:liu_xiangyuan@hotmail.com

陈书明 男,1961 年生于安徽六安,1993 年毕业于国防科技大学计算机系,获工学博士学位,现为国防科技大学计算机学院教授、博士生导师.主要研究方向为高性能微处理器设计和超深亚微米 VLSI 设计理论与技术.E-mail:smchen@nudt.edu.cn