(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 101930350 A
(43)申请公布日 2010.12.29

(21)申请号 200910117133.6

(22)申请日 2009.06.24

(71)申请人 合肥力杰半导体科技有限公司
地址 230088 安徽省合肥市高新区天元路 9 号顶间花园巴黎座 1501 室
申请人 龙迅半导体科技（合肥）有限公司

(72)发明人 苏进 陈峰

(51)Int.Cl.
G06F 5/06(2006.01)
G06F 17/50(2006.01)

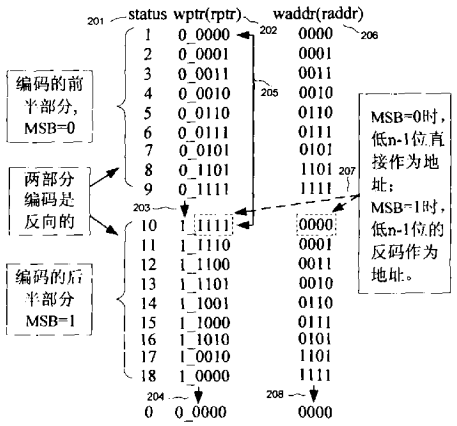
权利要求书 1 页 说明书 5 页 附图 2 页

(54)发明名称

深度不是 2 的幂的异步 FIFO 存储器设计

(57)摘要

深度不是 2 的幂的异步 FIFO 存储器设计属于集成电路领域，用于解决数据在不同时钟域之间的快速传递问题。异步时钟域之间的数据传输一般采用异步 FIFO 存储器来实现。由于异步 FIFO 一般是采用格雷码设计方式，这就要求所设计的 FIFO 深度是必须是 2^n。在大多数情况下，实际需要的深度不会恰巧是 2^n，因此这样的设计要求不仅增大了面积和功耗，而且由于 FIFO 先进先出的工作方式，多余的存储深度势必造成数据输出延迟增大。本发明提出了构造一种单步循环码实现异步 FIFO 的设计思路，使它的设计深度不再拘泥于特定值，不仅节省了芯片面积和功耗，还使数据的延迟（latency）降低了。



CN 101930350 A

1. 构造单步循环码设计深度不是 2 的幂的异步 FIFO 存储器。

2. 直接利用读 / 写指针的低位或低位的反码作为读 / 写地址，无需将指针先转化成二进制码再作为地址。

3. 利用读 / 写指针相等生成满标志，利用读 / 写指针相反生成空标志。

4. 使用这种方法在 FPGA 上生成异步 FIFO 存储器，用于存储、缓冲数据。

深度不是 2 的幂的异步 FIFO 存储器设计

一．技术领域

[0001] 本发明专利属于集成电路领域，用于解决数据在不同时钟域之间的快速传递问题。在含有多个时钟的芯片中，使用异步 FIFO 可以在两个不同时钟系统之间快速地传输数据。在 SoC 系统芯片中，异步 FIFO 存储器已经成为了必不可少的组成部分。在网络接口、图像处理等方面，异步 FIFO 也得到了广泛的应用。另外，使用 FPGA 做数据处理时，也往往在接口部分使用异步 FIFO 来存储、缓冲数据。本设计方法不涉及具体的工艺。

二．背景技术

[0002] 当今集成电路设计的主导思想之一就是同步化设计，即对所有时钟控制器件（如触发器、RAM 等）都采用同一个时钟来控制。但是随着设计规模的不断扩大，更多元件集成在同一裸片上，使裸片尺寸越来越大，这容易造成时钟偏差。在集成电路的设计中，一些新的方法，如整体异步局部同步 (GALS) 结构正在替代通常的同步方法，它不需要整体采用单一时钟因而避免了时钟的不确定性问题，另外在 SoC 芯片中也往往包含多个时钟。但多时钟域带来的一个问题就是，不可避免地要完成数据在不同时钟域之间的传递。如何设计异步时钟域之间的接口电路就成了一个必须考虑的问题。

[0003] 异步 FIFO(First In First Out) 是解决这个问题一种简便、快捷的方案。异步 FIFO 是一种先进先出的电路，使用在时钟频率不同的数据接口部分，用来存储、缓冲在两个异步时钟之间的数据传输。现在的异步 FIFO 存储器一般都是拘泥于格雷码设计方式，它有着自身的缺点。由于格雷码是一种具有反射特性和循环特性的单步自补码，它的码长 2n 决定了存储器的深度一定要是 2n-1，即若实际需要深度为 9 的存储器，则只能设计出深度为 16 的存储器来替代。由于 FIFO 先进先出的工作机制，大的 FIFO 这不仅造成电路面积和功耗的增加，还使得数据的输出延迟 (latency) 增大。

三．发明内容

[0004] 构造出一种具有单步特性和循环特性的编码方式来设计异步 FIFO 存储器，舍弃格雷码的反射和自补特性，这样新的编码的码长就不再要求是 2n。利用这种设计思路，就可以设计出一个任意深度的异步 FIFO。同时利用编码的低 n-1 位产生地址，避免了这种编码和二进制码的转换，也简化了电路结构。

四．附图说明

[0005] 图 1 异步 FIFO 存储器的电路框图，地址生成电路，空 / 满控制电路，双端口存储单元，两级同步器等。

[0006] 图 2 是本发明提出的用于设计深度为 9 的异步 FIFO 的编码。读 / 写指针采用五位编码，共有 18 个状态，分为上下两个码区，分别作为地址奇 / 偶次循环编码。低四位用来产生地址（最高位为 0 时低四位直接作为地址，为 1 时用低四位的反码作为地址）。

[0007] 图 3 所示本发明方案中提出的生成满标志的逻辑电路。满标志是在写时钟域中用

写指针 wptr 和同步的读指针 wq2_rptr 比较产生的，当二者互为反码时，FIFO 存储器处于满状态。其中 wq2_rptr 是 rptr 经两级同步器同步到写时钟域中的写指针。

[0008] 图 4 所示本发明方案中提出的生成空状态的逻辑电路。空标志是在读时钟域中用读指针 rptr 和同步的写指针 rq2_wptr 比较产生的，当二者相同时，FIFO 存储器处于空状态。其中 rq2_wptr 是 wptr 经两级同步器同步到读时钟域中的写指针。

[0009] 图 5 所示是异步 FIFO 的双端口存储介质及其附属电路。

五．具体实施方式

[0010] 以下内容具体的说明了本发明在实际应用中的原理和一种可实施的方案。本发明不仅仅限于以下所描述的应用及设计方案，如果对该领域了解并有足够的电路设计专业知识的人，很容易将本专利推广并应用于其它深度存储器的设计当中去。实现的具体方法也许会对编码的方式有所改变，但基本原理不变。在有的附图中采用数字标示以帮助描述各部分的原理及相互之间的关系。

[0011] 发明的内容主要是利用图 2 所提到的编码方式设计深度不是 2n 的异步 FIFO 存储器，以及图 3 和图 4 所示的生成空 / 满信号的电路。

[0012] 图 1 所示是异步 FIFO 的整体结构图。异步 FIFO 的整个系统可分为两个完全独立的时钟域——读时钟域和写时钟域；FIFO 的存储介质为一个双端口 RAM，可以同时进行读 / 写操作。在写时钟域部分，由写地址产生逻辑生成写控制信号和写地址；读时钟部分由读地址逻辑产生读控制信号和读地址。在空 / 满标志是通过读 / 写指针相互比较产生的。

[0013] 由于 FIFO 的空 / 满信号是由读 / 写指针的比较产生的，而读 / 写指针又不在同一个时钟域中，因此在产生空标志时要求写指针先从写时钟域传递到读时钟域中再和读指针比较，产生满标志要求读指针先从读时钟域传递到写时钟域中再和写指针比较。由于亚稳态的影响，不能将多位数据直接同步到另一时钟域中，那样采样到的可能是一个和原先数据毫无相关的值。最常见的方法是先将数据转化成格雷码再同步。由于格雷码每次只有一位翻转，因此最多只有一位发生亚稳态。即使翻转的那一位在采样时发生亚稳态，采样的结果也只可能有两种：一是翻转前的数据，二是翻转后的数据。以满状态为例，满标志的产生是写指针追上读指针的结果，若采样的结果是第一种情况，即采样数据是翻转前的值，写指针与之比较如果相等就将满标志置位，FIFO 存储器表现为满状态，实际上此时 FIFO 存储器并未满，它还有一个字节的存储空间。但是这不会引起 FIFO 存储器的操作错误，在下一个时钟周期 FIFO 存储器又会正常工作，它只是引起了 FIFO 存储器工作效率的降低，最重要的一点是它不会引起 FIFO 存储器的误操作。若采样的结果是第二种情况，则不会出现任何问题。

[0014] 由此看出，若读 / 写指针采用格雷码方式，就可以解决指针在读 / 写时钟域之间的传输问题。缺点是，格雷码指针只能用于实现大小为 2 的幂次的 FIFO 存储器。如果需要设计的 FIFO 存储器的存储容量为 9 个字节，若采用格雷码指针就必须设计成存储容量为 16 个字节的 FIFO 存储器，有 7 个字节的存储空间是多余的。由于 FIFO 是先进先出的工作方式，多出的存储空间不仅造成设计面积和功耗的增加，还使数据的输出延迟时间增大了，对于 9/16 的设计，数据输出延迟了 7 个时钟周期。这是采用格雷码设计的弊端所在。

[0015] 对读 / 写指针采用格雷码的编码方式的原因是由于它的单步循环特性，因此我们

可以创造出一种也具有单步循环性的编码，而舍弃它的反射和自补特性，其码长就不再是2n，这样就可以设计出任意深度的FIFO了。

[0016] 图2是深度为9的FIFO指针编码，它采用的是5位编码。码的总长度是18，分为上下两个部分，分别作为地址奇偶次循环编码。指针每历经一次存储循环，FIFO地址完成两次循环。编码的相邻两个码组之间只有一位不同，即从上一个状态到下一个状态的跳转只会有一位翻转，因此它具有单步特性；在204处，状态由18跳转到0状态，也是只有一位翻转，可见这种编码也具有循环性。用这种码作为读/写指针状态空间，就可以解决FIFO深度问题。

[0017] 编码的下半部分是上半部分的反码，如图2中的205处的双向箭头所示，状态10是状态1的反码。每一部分编码的最高位是相同的，在203处翻转。用这种编码作为读/写指针的状态值来设计异步FIFO，每一次的读/写操作都使读/写指针从上一个状态翻转到下一个状态。FIFO的地址是利用读/写指针的低n-1产生的，当最高位为0时，低n-1直接作为地址；当最高位为1时，将低n-1先反向再作为地址。如207处所示，MSB = 1，将低四位4′ b1111取反再作为地址，地址为4′ b0000。读/写指针从状态9跳转到10时，地址恢复到初始值。值得注意的是，读/写地址不是二进制的，它仅仅是一种编码，已经没有数量大小的含义，只是表示不同地址的代号而已。

[0018] 按照这种思路，可以设计出不同深度的FIFO存储器，解决了采用格雷码设计深度是2n的要求。

[0019] 下面代码是用verilog HDL实现的写指针和写地址：

[0020] parameter C_A1 = 5′ b0_0000；

[0021] parameter C_A2 = 5′ b0_0001；

[0022] parameter C_A3 = 5′ b0_0011；

[0023] parameter C_A4 = 5′ b0_0010；

[0024] parameter C_A5 = 5′ b0_0110；

[0025] parameter C_A6 = 5′ b0_0111；

[0026] parameter C_A7 = 5′ b0_0101；

[0027] parameter C_A8 = 5′ b0_1101；

[0028] parameter C_A9 = 5′ b0_1111；

[0029] parameter C_B1 = ~ C_A1；

[0030] parameter C_B2 = ~ C_A2；

[0031] parameter C_B3 = ~ C_A3；

[0032] parameter C_B4 = ~ C_A4；

[0033] parameter C_B5 = ~ C_A5；

[0034] parameter C_B6 = ~ C_A6；

[0035] parameter C_B7 = ~ C_A7；

[0036] parameter C_B8 = ~ C_A8；

[0037] parameter C_B9 = ~ C_A9；

[0038] always@(posedge wclk or negedge wrst_n)

[0039] if(! wrst_n)begin

```
[0040]          wptr <= C_A1 ;
[0041]      end
[0042]      else if(wr_en&& ~ full)begin
[0043]          case(wptr)
[0044]            C_A1 :begin wptr <= C_A2 ;end
[0045]            C_A2 :begin wptr <= C_A3 ;end
[0046]          C_A3 :begin wptr <= C_A4 ;end
[0047]          C_A4 :begin wptr <= C_A5 ;end
[0048]          C_A5 :begin wptr <= C_A6 ;end
[0049]          C_A6 :begin wptr <= C_A7 ;end
[0050]          C_A7 :begin wptr <= C_A8 ;end
[0051]          C_A8 :begin wptr <= C_A9 ;end
[0052]          C_A9 :begin wptr <= C_B1 ;end
[0053]          C_B1 :begin wptr <= C_B2 ;end
[0054]          C_B2 :begin wptr <= C_B3 ;end
[0055]          C_B3 :begin wptr <= C_B4 ;end
[0056]          C_B4 :begin wptr <= C_B5 ;end
[0057]          C_B5 :begin wptr <= C_B6 ;end
[0058]          C_B6 :begin wptr <= C_B7 ;end
[0059]          C_B7 :begin wptr <= C_B8 ;end
[0060]          C_B8 :begin wptr <= C_B9 ;end
[0061]          C_B9 :begin wptr <= C_A1 ;end
[0062]          default :begin wptr <= C_A1 ;end
[0063]        endcase
[0064]    end
[0065]  always@(wptr)
[0066]    if( ! wptr[4])
[0067]          waddr = wptr[3:0] ;
[0068]    else
[0069]          waddr = ~ wptr[3:0] ;
```

[0070] 读指针和读地址的实现与此类似。

[0071] 当读地址和写地址相等也就是指向同一个内存位置的时候，FIFO 可能处于空或满两种状态。因此需要通过一种方法判断或区分 FIFO 是处于空状态还是满状态，也就是究竟是写地址追上了读地址，还是读地址赶上了写地址。可以通过判断读 / 写指针循环的次数来区分空和满状态，循环次数相同时为空状态，循环次数不同时为满状态。

[0072] 图 3 是生成满标志的逻辑电路。满标志是在写时钟域中通过写指针与同步到写时钟域中的读指针比较得到的。当读 / 写地址相同而读 / 写指针循环的次数不同时，说明是写地址追上了读地址，这时 FIFO 处在满状态。从图 2 中的编码可以看出当读 / 写指针相反时，它们处在不同的编码区，读 / 写地址循环的次数不同，但读 / 写地址是相同的，此时 FIFO

处在满状态。因此，可用读 / 写指针的相应位异或的与来生成满信号，其中 wq2_rptr 是经过两级同步器同步到写时钟域的读指针。采用 verilog HDL 生成满标志显得尤为简单 ：

[0073] assign full = (wptr == ~ wq2_rptr) ;

[0074] 图 4 是空标志的逻辑电路。空标志是在读时钟域中通过读指针与同步到读时钟域中的写指针比较得到的。当读 / 写指针相同时，读和写的循环的次数相同且读 / 写地址也相同。因此，可用读 / 写的相应位异或的或非来生成空信号。其中 rq2_eptr 是同步到读时钟域的写指针。

[0075] 采用 verilog HDL 描述空标志为 ：

[0076] assign empty = (rptr == rq2_wptr) ;

[0077] 空 / 满信号的撤销和实际 FIFO 的工作情况相比有一定的延迟。例如满信号是利用写指针和同步到写时钟域中的读指针比较产生的，而读指针同步到写时钟域需要一定的时间，因此同步的读指针 wq2_rptr 并不能反映当前读指针 rptr 的真实值，这可能导致在 FIFO 存储器已经不再处于满状态时，满标志却要在两个写时钟周期之后才能撤销。这种现象好比 FIFO 动态缩小了一般，但是这没有多少坏处。空 / 满信号的置位和 FIFO 存储器的实际情况相比没有延迟。在 FIFO 存储器将满的情况下，由于读指针不发生变化，在写时钟域得到的是当前的读指针值，能够马上判断出 FIFO 存储器是否为满。因此异步 FIFO 可以正确地置位满标志和空标志，但撤销满标志和空标志却是保守的。保守的空 / 满状态判断不会造成 FIFO 的误操作。

[0078] 图 5 所示的是 FIFO 的存储介质、地址译码电路和读写控制电路。

[0079] 本专利描述的实施方法，已经在 65nm 工艺和 0.18um CMOS 工艺流片并验证成功，并且在 FPGA 上也验证成功。

[0080] 对本专利的侵权，一般可以对其实施电路的分析来判断，在无法得到其电路的情况下，可以对其芯片进行解剖、拍照等反向分析方法来判断。对于 FPGA 的侵权，可以通过其烧制的程序来判断。可能侵权的机构包括各种有厂、无厂的芯片设计公司，研究机构、学校等。

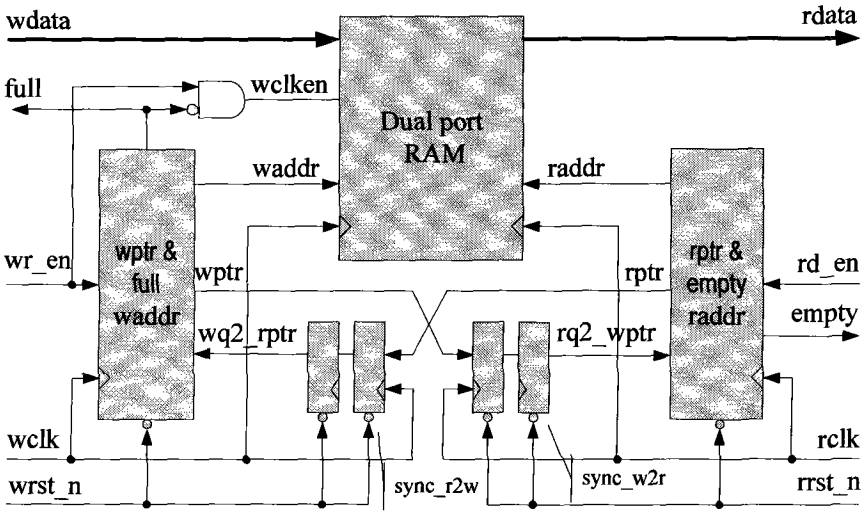

wdata
rdata
full
wclken
Dual port RAM
waddr
raddr
wr_en
wptr & full waddr
wptr
rptr
rptr & empty raddr
rd_en
empty
wq2_rptr
rq2_wptr
wclk
rclk
wrst_n
sync_r2w
sync_w2r
rrst_n

图 1

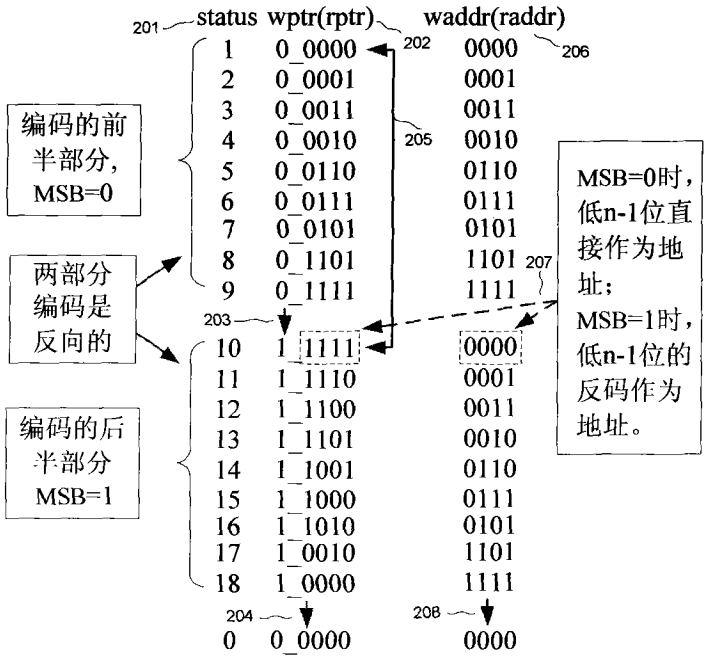

201
status wptr(rptr)
waddr(raddr)
202
206
205
编码的前半部分，MSB=0
两部分编码是反向的
编码的后半部分 MSB=1
203
207
MSB=0时，低n-1位直接作为地址；MSB=1时，低n-1位的反码作为地址。
204
208
1 0_0000 0000
2 0_0001 0001
3 0_0011 0011
4 0_0010 0010
5 0_0110 0110
6 0_0111 0111
7 0_0101 0101
8 0_1101 1101
9 0_1111 1111
10 1_1111 0000
11 1_1110 0001
12 1_1100 0011
13 1_1101 0010
14 1_1001 0110
15 1_1000 0111
16 1_1010 0101
17 1_0010 1101
18 1_0000 1111
0 0_0000 0000

图 2

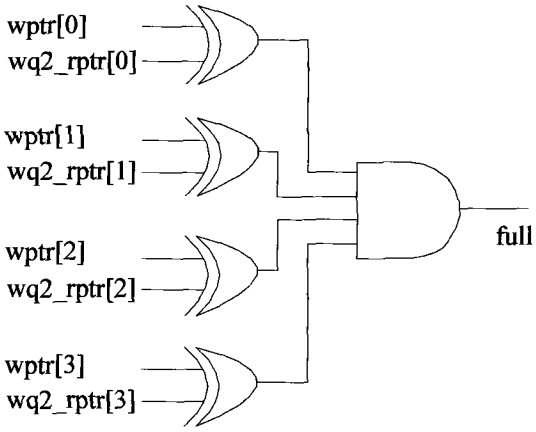

wptr[0]
wq2_rptr[0]
wptr[1]
wq2_rptr[1]
wptr[2]
wq2_rptr[2]
wptr[3]
wq2_rptr[3]
full

图 3

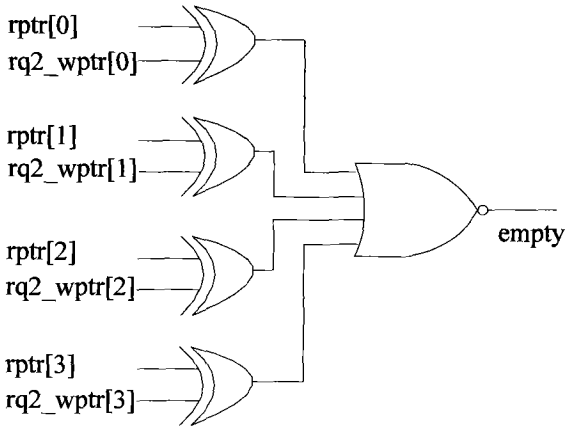

rptr[0]
rq2_wptr[0]
rptr[1]
rq2_wptr[1]
rptr[2]
rq2_wptr[2]
rptr[3]
rq2_wptr[3]
empty

图 4

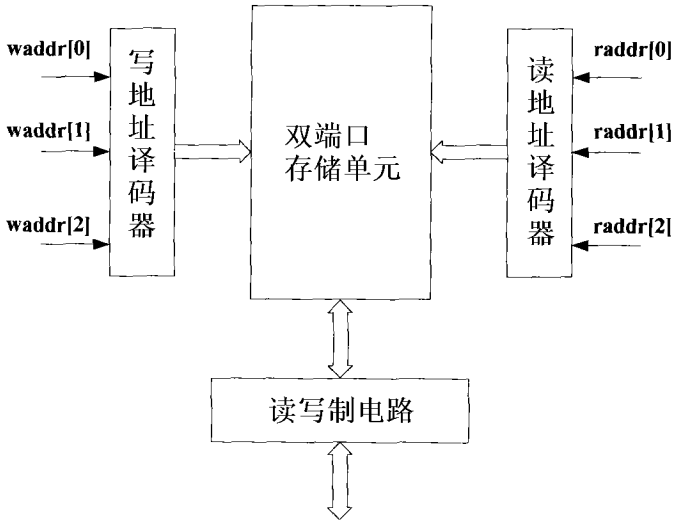

waddr[0]
waddr[1]
waddr[2]
写地址译码器
双端口存储单元
读地址译码器
raddr[0]
raddr[1]
raddr[2]
读写制电路

图 5