FIFO最小深度计算

FIFO的深度问题是为了解决数据传输中的读写速率不匹配的问题，如果一个FIFO的写入速率比读出速率快，那么没来得及及时读出的数据就需要存储在FIFO中等待被读出。其实这就是一个比较简单的数学问题，可以联想到小学题目中给游泳池一边灌水一边放水（？）。

# 基础题

一般通常会见到题干是这样的：“一笔数据的个数N是120个，FIFO的写入时钟频率fw为80MHz，FIFO的读出时钟频率fr为50MHz，求FIFO的最小深度。”

其实这道题就是在问一个游泳池需要多大才能一边灌水一边放水，水还不能溢出游泳池。那么我可以直接用数据总量120，除以写入速率80MHz，可以得到一个灌水时间 T = 120 / 80MHz 。那么在这么长时间中，我们能放多少水呢？就是 n = T \* 50MHz = 120 / 80MHz \* 50MHz = 75 。所以我们至少需要一个 （120 - 75 = 45）这个大的一个游泳池才能满足要求，即就是我们所需要的FIFO最小深度为 45 。这里可以提炼出公式：

最小深度：d = N - N \*(fr / fw)

有时出题的老师会再绕一个弯，加考一些基础知识。例如：一笔数据是5kbit，FIFO的宽度是1byte，balabala... 在这里用数据总量除以FIFO宽度就是数据个数了。即一笔数据的个数为 5kbit / 8bit = 640 个。

# 附加题1

因为异步FIFO需要处理异步界面，不能像放水一样那么迅捷，所以出题时可能会提到“当写入 3 （任意数）个数据后才能开始读出”，那么我们的 FIFO 最小深度就必须加上这个缓冲量，即FIFO最小深度为 45 + 3 = 48 。

# 附加题2

“ 若每隔一个周期写一次数据，每隔三个周期读一次数据，请计算最小深度。”

每隔一个周期写一次数据，意味着每两个周期才写一次数据，意味着我们的写入速率从原先的80MHz降为了40MHz。以此类推，读出速率从50MHz降成了12.5MHz 。这样一来再套用公式 最小深度 d = N - N / fw \* fr = 120 - 120 / 40 \* 12.5 = 82.5 。即可得FIFO的最小深度为 83 。

从这道题可以看出，读写时钟频率和读写速率是两回事，以此题为蓝本的变种类型题还有很多，但是万变边不离其宗，关于FIFO最小深度的计算我们主要关注读写速率即可。

# FIFO设计中的深度计算

写时钟频率 w\_clk,

读时钟频率 r\_clk,

写时钟周期里，每B个时钟周期会有A个数据写入FIFO

读时钟周期里，每Y个时钟周期会有X个数据读出FIFO

则，FIFO的最小深度是？

计算公式如下：

fifo\_depth = burst\_length - burst\_length \* X/Y \* r\_clk/w\_clk

例举说明：

如果100个写时钟周期可以写入80个数据，10个读时钟可以读出8个数据。

可以认为写操作是突发写的。然后要考虑突发的情况，一般情况下，数据传输是：空闲—Burst突发—空闲—Burst突发—空闲—Burst突发。但是我们在计算中，需要考虑最极端的情况，即空闲—Burst突发—Burst突发—空闲—Burst突发—空闲。

以最极端的情况空闲—Burst突发—Burst突发—空闲来计算。

此时考虑背靠背（20个clk不发数据＋80clk发数据＋80clk发数据＋20个clk不发数据的200个clk）

令wclk＝rclk ，考虑背靠背（20个clk不发数据＋80clk发数据＋80clk发数据＋20个clk不发数据的200个clk）代入公式可计算FIFO的深度

fifo\_depth = 160-160\*(8/10)\*(rclk/wclk)=160-128=32

异步FIFO

# 一、FIFIO简介

FIFO是一种现先进先出的数据缓冲器，特点是没有外部的读写地址。由于没有外部的地址信号，所以只能顺序的读写，而不能跳读。FIFO的读写是根据满和空信号设计写使能和读使能来写/读FIFO，当FIFO满的时候不可以往里面写、当FIFO空的时候不能读数据。读FIFO时，内部的读指针自动的加一，当写FIFO时写指针自动的加一。

什么是异步FIFO，什么又是同步FIFO？

异步FIFO简单的来说就是读写时钟不相同，同步FIFO就是读写的时钟相同。

# 二、异步FIFO的用途

1、使用异步FIFO可以在两个不同的时钟域之间快速而方便的传输数据，起到跨时钟域处理的作用。经常用于处理跨时钟域的问题。

2、对于不同宽度的数据接口也可以采用FIFO进行缓冲，如8位输入，16位输出。（注：本文只简介输入输出位宽相同的情况）

# 三、FIFO的常见参数

wfull: 满标志， 表示FIFO已经满，不能再写入数据。

rempty：空标志，表示FIFO已经空，不能再读取数据。

wclk: 写时钟

rclk: 读时钟

winc: 写使能

rinc: 读使能

wdata：写数据

rdata: 读数据

wrst\_n: 写复位

rrst\_n：读复位

# 四、读写指针的工作原理

读指针：总是指向下一个将要被写入的单元，复位时指向第一个单元。

写指针：总是指向当前要被读出的数据，复位时指向第一个单元。

也就是说，复位时读写指针都指向第一个单元。并且向FIFO写入一个数据，写指针加1。从FIFO中读出一个数据，读指针加1。

# 五、FIFO满空标志的产生

FIFO设计的关键是如何产生可靠的读写指针和满空信号。FIFO读写指针的工作原理如上第四点所述。

那么剩下的就是要讨论如何产生FIFO的满空信号了。

FIFO什么时候为空呢？我们来思考一下，假设我从第一个单元写入数据，那么写指针从地址0—>1,读指针不变，此时FIFO中有一个数据。接着我把这个数据读出来，读指针从0—>1。此时写指针为1，读指针也为1,FIFO中没有数据了，因此FIFO为空。

从中可以发现，判断FIFO为空很简单，只要读写指针相等就是空。但是事情好像也没那么简单，再想一下，假设一开始就复位，读写指针都在0地址，然后一直往FIFO中写入数据，当写满FIFO的时候，写指针刚好转了一圈回到了0地址，此时读写指针也相等，但是这时候FIFO是满的。因此得到下面的判空和判满条件。

判空：读指针追上写指针的时候，两者相等，为空。

判满：写指针追上读指针的时候，两者相等，为满。

突然发现两者相等的话不是空就是满，区别就是谁追上谁而已了。那么如何来区别是谁追上谁呢？

# 六、如何判断读写指针相等时，为空还是为满呢？

答案就是在表示读写指针的数据位宽上再加1位来区分是满还是空。比如FIFO的深度位8，那么需要3位二进制数来表地址，则需要再最高之前再加一位，变成4位。一开始读写都是0000，FIFO为空。当写指针增加并越过最后一个存储单元的时候，就将这个最高位取反，变成1000。这时是写地址追上了读地址，FIFO为满。同理，当读地址越过最后一个存储单元的时候把读地址的最高位也取反。可以看到，当最高位相同，并且剩下的位也相同的时候FIFO为空；当最高位不同，并且剩下的位相同时，为满。

# 七、异步时钟域下如何判断时空还是满？

在上述六中已经解释了如何判断FIFO的满空。但是如果在不同时钟域下，显然需要将读写指针进行同步化才可以进行判断。具体就是在判断空的时候，需要将写地址同步到读时钟域下进行判断。同理，在进行判断满的时候需要将读时钟域中的读指针同步到写时钟域进行判断。

# 八、使用格雷码来表示地址

其实在读时钟域中读指针的增加仍然是自然二进制，同理在写时钟域中写地址的增加也是按照自然二进制变化的。但是在将读指针发送到写时钟域下进行同步时，如果仍然采用自然二进制，那么就会面临地址同时有多位变化的情况。比如0111->1000,一次就变了四位。在数电的学习中我们知道，这种情况是要尽量避免的，因为这样容易引起亚稳态或者是毛刺（具体是亚稳态还是毛刺我还不太确定）。

那么问题又来了，采用格雷码又要如何判断满和空呢？首先还是要记住：在读指针和写指针相等的时候进行判断。

举个例子：

假如T1时刻，读指针的自然二进制为0111，写指针的自然二进制位1000。

简化如下：

T1：读：0111（bin） 0100（grey）

写：1000（bin） 1100（grey）

可以看到，此时此时格雷码 的最高位不同，剩下的都相同。那么可以判为满吗？显然是不行，可以冥想的知道，此时刚刚向FIFO中写入一个数，怎么就满了呢。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 十进制 | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 二进制码 | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 |
| 格雷码 | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 |

因此必须考虑用别的办法来比较。方法就是：

判满：格雷码的最高位和次高为不同，剩下的都同，就是满。

判空：格雷码完全相同，就是空。

【重点】因为当只有一个比特发生改变时，即使在中间状态抽样，其结果也不外乎两种：递增前原指针和递增后新指针。显然递增后新指针是最新情况的反映，如果抽样到这个指针，那么和我们的设计预期是一致的，如果抽样到递增前的原指针，会有什么结果呢？假设现在抽样读指针，那么最坏的情况就是把“不满”判断成了“满”，使得本来被允许的写操作被禁止了，但是这并不会对逻辑产生影响，只是带来了写操作的延迟。同样的，如果现在抽样写指针，那么 最坏的情况就是把“不空”判断成了“空”，使得本来被允许的读操作被禁止了，但是这也不会对逻辑产生影响，只是带来了读操作的延迟。

同步FIFO

# 一、什么是同步FIFO？

这部是很简单的问题吗？同步FIFO就是读写时钟域都是同一个时钟的FIFO。也就是说同步FIFO不用考虑跨时钟域的问题。同步FIFO和异步FIFO的根本区别就是读写时钟的不一样。

# 二、同步FIFO的常见参数

clk：读写时钟

rst\_n：同步复位信号

w\_en：写使能信号

r\_en：读使能信号

wdata：输入数据信号

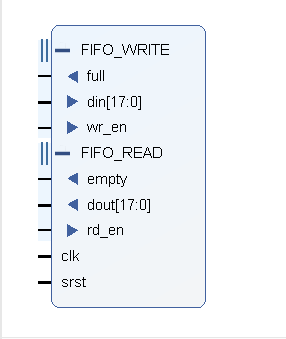
rdata：读出数据信号

wfull：满信号

rempty：空信号

# 三、同步FIFO建模

同步FIFO的模型可比异步FIFO的模型简单多了。同步FIFO的模型一般如下所示。（懒得画图了，直接用Xilinx的模型）



可以看到我们设计的端口信号和XIlinx的同步FIFO的模型的端口信号是一致的。FIFO的读写行为如下：

当需要写入数据时：需要判断两个信号，一个是wfull信号，另一个是写使能信号w\_en。只有FIFO不满并且写使能有效的时候才可以写入数据。用逻辑来表达就是：

(!wfull&&w\_en==1’b1),并且此时的数据要和写使能对齐。用代码来表示就是：

always@(\*) begin

if(rst\_n==1'b0)

wdata = 0;

else if(!wfull&&w\_en==1)

wdata = i;

else

wdata <= 0;

end

如果不满足这个条件，那么写入数据就会有错误。

当需要读出数据的时候：需要判断两个信号，一个是rempty信号，另一个是r\_en信号。只有当FIFO不空，并且读使能有效的时候才可以读取数据。用逻辑来表示就是：

（!rempty&&r\_en==1'b1）

读使能没有那么讲究了，只要不空随时可以读。

always@(posedge clk or negedge rst\_n) begin

if(rst\_n==1'b0)

r\_en =1'b0;

else if(!rempty)

r\_en = 1'b1;

else

r\_en = 1'b0;

end

# 四、地址跳变

FIFO设计最难的地方就在于如何设计读写指针，反正我在调试的时候遇到了很多问题。

总之记住:

FIFO每读一个数据，读指针就加一。FIFO每写一个数，写指针就加一。

FIFO的读和写指针永远都是指向下一个即将要读或者写的单元。

# 五、判空和判满

同步FIFO的判空和判满也是和异步FIFO一样的，区别在于异步FIFO在各自的时钟域使用格雷码进行判空和判满；而同步FIFO只有一个时钟域，判空和判满是使用的自然二进制地址。

FIFO的读指针和写指针相同时不是空就是满，同步FIFO同样需要一个额外的位来区别是空还是满。

判空：assign rempty\_val = (rbinnext==wbinnext);

判满：assign wfull\_val = (rbinnext=={~wbinnext[ADDR\_SIZE], wbinnext[ADDR\_SIZE-1:0]});

注意：这里使用的是rbinnext和wbinnext进行判空和判满。因为我们说过了，FIFO的指针总是指向下一个即将要写入或者读取的数据，因此使用的是对应的next信号。而把真正需要送到RAM的地址用：

assign raddr = rbinnext[ADDR\_SIZE-1:0];

assign waddr = wbinnext[ADDR\_SIZE-1:0];