# 异步 FIFO 的设计

CNDLAB 程磊

I．综述

在多时钟域系统的设计中，跨时钟域的设计在所难免，尤其是在设计模块和外围芯片通信中出现。当数据从一个时钟域向另一个异步时钟域传递的过程中，由于不同速率器件间的速率匹配问题，一般不能够通过直接传递保证数据传输的正确、完整性，故需要有一个中间数据缓冲来保证数据能够准确无误的传递。FIFO(First In First Out)是一种队列，数据从一端进队列，另一端出队列，先进入队列的数据先被读取出，这种机制可以保证数据的时序上的正确性。除此之外对 FIFO 的应用不仅仅是跨时钟域接口设计，在同步系统中可以起到防止时钟抖动等因素造成的数据传输错误提高系统稳定性能。

FIFO 有同步和异步之分，同步的 FIFO 的读写时钟为同一时钟信号，而异步 FIFO 的读写时钟相互独立互不影响。由于缓冲区是 RAM，需要引入读写指针。写指针总是指向下一个将要被写入的地址，在 Reset 后归零地址；读指针总是指向当前待读出的指针，在 Reset 后归零。

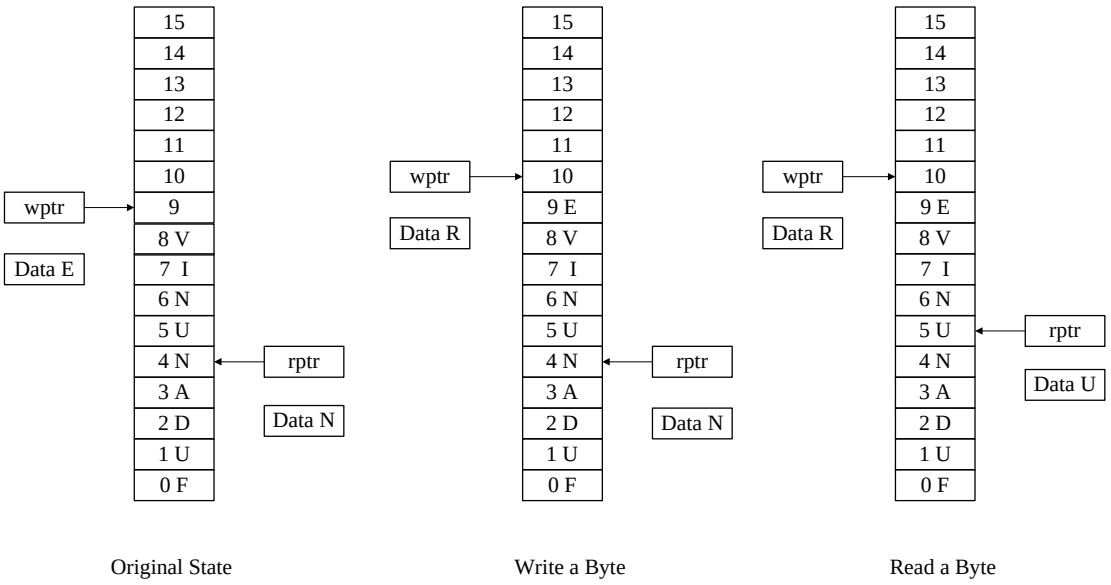


Fig.1 读写指针工作原理

在进行设计时同步和异步 FIFO 的最大区别就在于读写指针是否为同步或异步。同步 FIFO 中 wptr 与 rptr 在同一时钟上升沿变化，而对于异步而言，读指针仅仅在读时钟上升沿变化，写指针仅仅在写时钟上升沿变化。

FIFO 的缓冲区空间是有限的，在设计中需要考虑缓冲区的溢出和空的情况。在缓冲区满时，产生 FULL 信号，以阻塞数据继续向 FIFO 写入。而在缓冲区空时，产生 EMPTY 信号，以阻塞数据继续从 FIFO 中读出。外部的读写控制器将从标志信号 FULL、EMPTY 决定继续写入或读出。这种判断是基于对读指针和写指针指向的地址来判断的。判断标志信号是 FIFO 的设计重点之一。

在同步的 FIFO 中读指针和写指针是同步的，所以在时钟的上升沿时比较判断 FULL 或 EMPTY 时，读指针和写指针指向的地址都是实际指针所指向的；而异步 FIFO 的读写指针是不同步的，这样就会造成在进行判断比较时，无论采用读时钟还是写时钟上升沿去判断

FULL 或 EMPTY 都将不合适。所以在异步 FIFO 中需要一种机制使得能够将读写指针准确传递，一种解决方法是采用握手协议来保证在采样过程中指针不变，但是这样的做法效率很低。另一种是通过读时钟来采样写指针，写时钟来采样读指针，然后通过采样的写指针和读指针判断是否为 EMPTY，采样的读指针和写指针判断是否为 FULL。EMPTY 信号对于读时钟而言是同步的，FULL 信号对于写时钟是同步的。

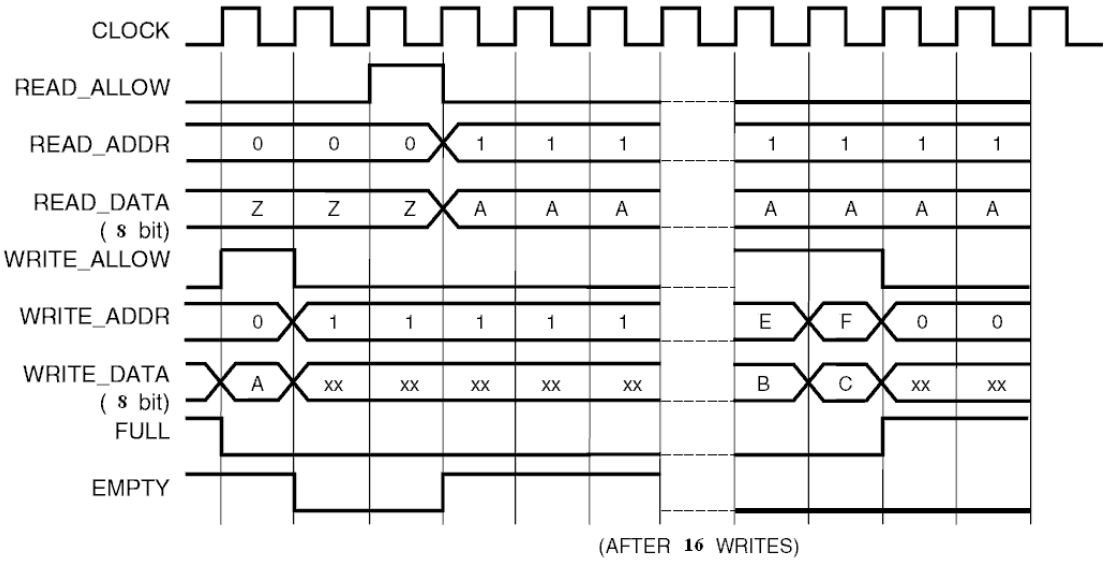


Fig. 2 一种同步 FIFO 的读写时序

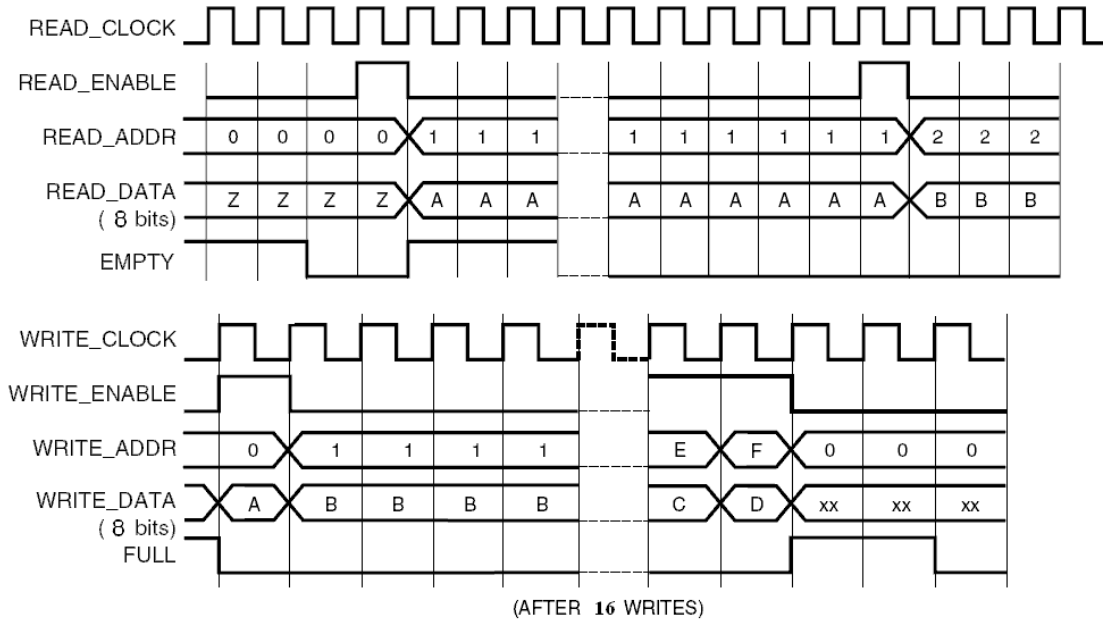


Fig. 3 一种异步 FIFO 的读写时序

II．深入考虑

在前面提到如果打算用写时钟取样读指针或用读时钟来取样写指针，将不可避免的遇到一个亚稳态的问题，它将会导致空、满标志的判断错误，并导致设计的失败。下面分析亚稳态的产生原因和消除方法。

对于采样电路中，一个信号在过渡到另一个时钟域时，如果仅仅用一个触发器将其锁存，那么采样的结果将可能是亚稳态。这也就是信号在跨时钟域时应该注意的问题。如 Fig.4 中时钟 a，b 是异步的，第二个触发器通过时钟 b 采样由寄存器锁存的数据 data_a。显然这种做法是冒险的，我们可以知道触发器工作过程中存在数据的建立(setup)和保持(hold)时间。对于使用上升沿触发的触发器来说，建立时间就是在时钟上升沿到来之前，触发器数据端数据保持稳定的最小时间。而保持时间是时钟上升沿到来之后，触发器数据端数据还应该继续保持稳定的最小时间。我们把这段时间成为 setup-hold 时间。在这个时间参数内，输入信号在时钟的上升沿是不允许发生变化的。如果输入信号在这段时间内发生了变化，输出结果将是不可知的，即亚稳态 (Metastability)。信号 data 经过一个锁存器的输出数据为 data_a。用时钟 b_clk 进行采样的时候，如果 data_a 正好在 b_clk 的 setup-hold 时间内发生变化，此时 data_b 就既不是逻辑"1"，也不是逻辑"0"，而是处于中间状态。经过一段时间之后，有可能回升到高电平，也有可能降低到低电平。输出信号处于中间状态到恢复为逻辑"1"或逻辑"0"的这段时间，我们称之为亚稳态时间。

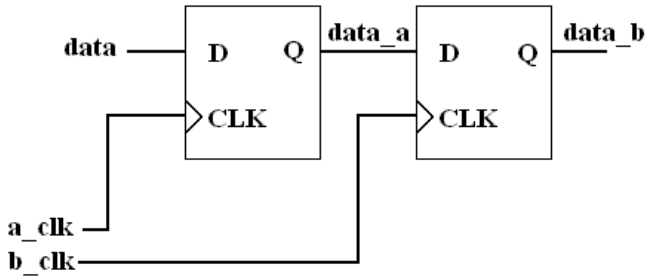


Fig. 4 采样电路

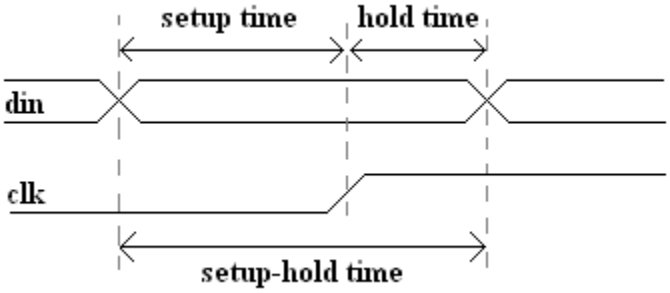


Fig. 5 D 触发器的设定保持特性

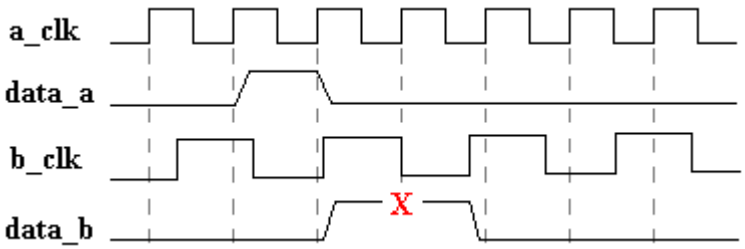


Fig. 6 由采样电路产生的亚稳态时序

为了考量亚稳态发生的几率可以用参数 MTBF(Mean Time Between Failures，平均无故障时间间隔)来描述，MTBF 即触发器采样失败的时间间隔，表示为：

$$MTBF = \frac{\exp(t_{res}/\tau)}{T_0 \times f_{clk} \times f_{data}} \quad \text{Eq. 1}$$

上式$t_{res}$中为分辨时间，$\tau$ 和$T_0$为触发器器件特性触发器特性相关参数，$f_{clk}$是采样时钟频率，$f_{data}$是异步信号的触发频率。如果 MTBF 很大，就认为这个设计在实际工作中是能够正常运行的，不会因为亚稳态导致整个系统的失效。当触发器处于亚稳态，且处于亚稳态的时间超过了一个时钟周期，这种不确定的状态还会影响到下一级的触发器，最终导致连锁反应，从而使整个系统功能失常。为了避免进入亚稳态，应当使参数 MTBF 尽可能大。通常采用的方法是双锁存器法，即在一个信号进入另一个时钟域 Fig. 7 双锁存器法之前，将该信号用两个锁存器连续锁存两次(如图 Fig. 7 所示)，由前给出的 MTBF 公式可以得到两级触发器的 MTBF 公式：

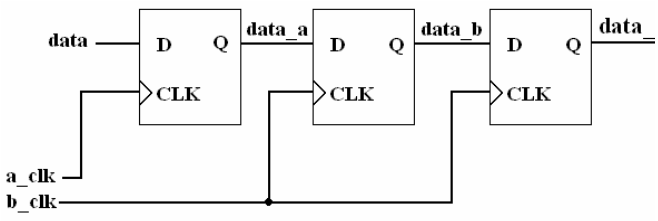


Fig. 7 双锁存器法

$$MTBF_2 = \frac{\exp(t_{res}/\tau)}{T_0 \times f_{clk} \times f_{data}} \times \exp\left(\frac{1/f_{clk} - t_{pd}}{\tau}\right) \quad \text{Eq. 2}$$

一般而言$t_{res} >> \tau$ 所以 exp 的增长比分母的平方增长快，所以在其他参数不变的条件下 Eq.2 比 Eq.1 来得大，data_c 进入亚稳态的几率就更加减小。依次类推，多增加锁存器的级数可以进一步降低出现亚稳态的几率，但是作为平衡考虑一般采用 2 级 MTBF 就已经足够大了。

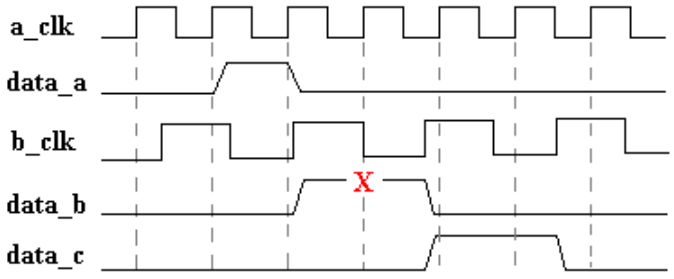


Fig. 8 双锁存器采样异步信号时序

引入两级锁存器的同时带来了对输入信号的一级延时，再如 a_clk 的频率比 b_clk 的频率高，将可能出现因为 dat 变化太快，而使 b_clk 无法采样的问题。即在信号从快时钟域向慢时钟域过渡的时候，如果信号变化太快，慢时钟将可能无法对该信号进行正确的采样，所以在使用双锁存器法的时候，应该使原始信号保持足够长的时间，以便另一个时钟域的锁存器可以正确地对其进行采样。但是在这里的 FIFO 设计中并不需要过于关心这个两问题，原因将在后面分析。

采用双锁存器可以大大降低亚了稳态的出现几率，但是完全消除亚稳态是不可能的，一旦发生亚稳态那么读写指针所读取的数据就会发生错误。读写指针在每次写入或读取之后会

加一以使指针指向下一个地址，一般我们会采用 Binary code。但是采用 Binary code 是相邻的两个数之间可能有若干位 bit 改变（7->8 == 0111->1000 四位全部改变），这样增加了产生亚稳态的几率。一种很好的解决方法是采用 Gray code 来对地址进行编码。Gray code 是一种最小距离编码方式，既相邻的两数时间仅有 1bit 的差别，那么在 FIFO 应用中每次仅有一位改变，

| Order | Binary Code | Gray Code | Convert Rules |
|---|---|---|---|
| 0 | 0000 | 0000 | G->B |
| 1 | 0001 | 0001 | $B_n = G_n$<br>$B_i = G_i \oplus B_{i+1}$<br>$\forall i \neq n$ Eq.3 |
| 2 | 0010 | 0011 | |
| 3 | 0011 | 0010 | |
| 4 | 0100 | 0110 | |
| 5 | 0101 | 0111 | |
| 6 | 0110 | 0101 | |
| 7 | 0111 | 0100 | |
| 8 | 1000 | 1100 | B->G |
| 9 | 1001 | 1101 | $G_n = B_n$<br>$G_i = B_i \oplus B_{i+1}$<br>$\forall i \neq n$ E.q 4 |
| 10 | 1010 | 1111 | |
| 11 | 1011 | 1110 | |
| 12 | 1100 | 1010 | |
| 13 | 1101 | 1011 | |
| 14 | 1110 | 1001 | |
| 15 | 1111 | 1000 | |

Table. 1 Binary Gray Code 对照以及转换方法

读写指针采用了 Gray code 那么地址译码将使用 Gray code，判断空、满就和采用 Binary code 就不一样了，接着我们详细分析。

为了最大限度利用 RAM 资源，FIFO 一定是环形队列。对于环形队列判断其是否空还是满的准则是：当读指针赶上写指针时，队列为空；当写指针赶上读指针时，队列为满。显然只要是读写的指针的地址 n 位全都相同就会有空或满的状态，显然这样并不能区分空或满的状态。则当对 $2^n$ 个地址进行编码时，如果我们再增加一位，即利用 n+1 位对 $2^n$ 个地址进行编码时，MSB 可以表示是读指针赶上写指针还是写指针赶上读指针。如果是 Binary code 的方式，以 n=3 为例(如 Fig. 9)给 $2^3$ 个地址编码，多加一位，则一共有 $2^{3+1}$ 个编码，从 0000 到 0111，从 1000 到 1111。我们可以看到尽管 MSB 有变化，但是其余的位是相同的在地址译码的过程中，存在周期性只需要取后三位就是地址了。

对于判断空、满则在 rptr(MSB-1:0)=wptr(MSB-1:0)条件下 rptr(MSB)=wptr(MSB)则代表为队列为空，$\text{rptr(MSB)} \neq \text{wptr(MSB)}$代表为队列为满；

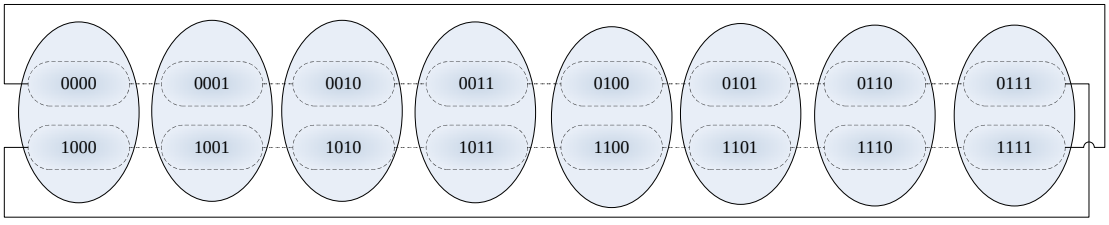


Fig. 9 MSB-1:0 的周期性

而在 Gray code 的方式下，由于 Gray Code 是一种带有镜像对称的编码，故情况比较复杂。增加一位后我们可以发现剩下的几位并不像 Binary Code 那样可以重复出现，而是关于中间对称。如 Fig. 10 中，(2:0)关于中心对称，分成的两部分则(1:0)关于中心对称，既然这样，我们可以知道对 n+1 bit 采用 Gray Code，n-1 bit 的编码带有周期为 $2^n$ 周期性。但是仅是 n-1 bit，而我们需要的是 n bit 的地址。

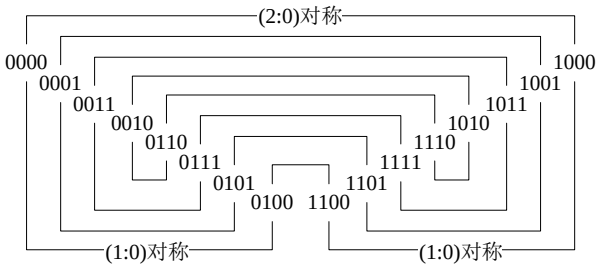


Fig. 10 Gray Code 镜像对称特性

MSB 和 MSB-1 可以进行异或运算作为新的 n bit 地址的最高位来获得 n bit 地址(实质上是将+1 bit 的 Gray code 转换成 n bit 的 Gray Code)。如 Fig. 11 中，按照前面所说的方法，将内外圈进行转换，在转换后的图中内外圈的数字完全一样，这样就产生的 n bit 的周期为 $2^n$ 的地址。

此时由于 n+1 bit 缩减为 n bit，不能够用缩减以后的地址来判断是空还是满，但是可以用转换以前的 MSB 来判断。把转换前的 MSB 和转换后的(MSB-1:0)合并成一个新的指针。这样的表示方法即可以起到判断空满状态，指针中直接包含译码的地址。利用这种方法，空满的判断和采用 Binary Code 方法一样了。在 rptr(MSB-1:0)=wptr(MSB-1:0) 条件下，rptr(MSB)=wptr(MSB) 则代表为队列为空，rptr(MSB) ≠ wptr(MSB) 代表为队列为满。这样的做法中还存在一个问题，可以发现在边界处的变化出现了存在 2 bit 的变化过程，如 1100->0000，0100->1000，这是一种折中的方法，但可以通过整体系统结构的设计来解决这个问题，我们将在后面讨论。

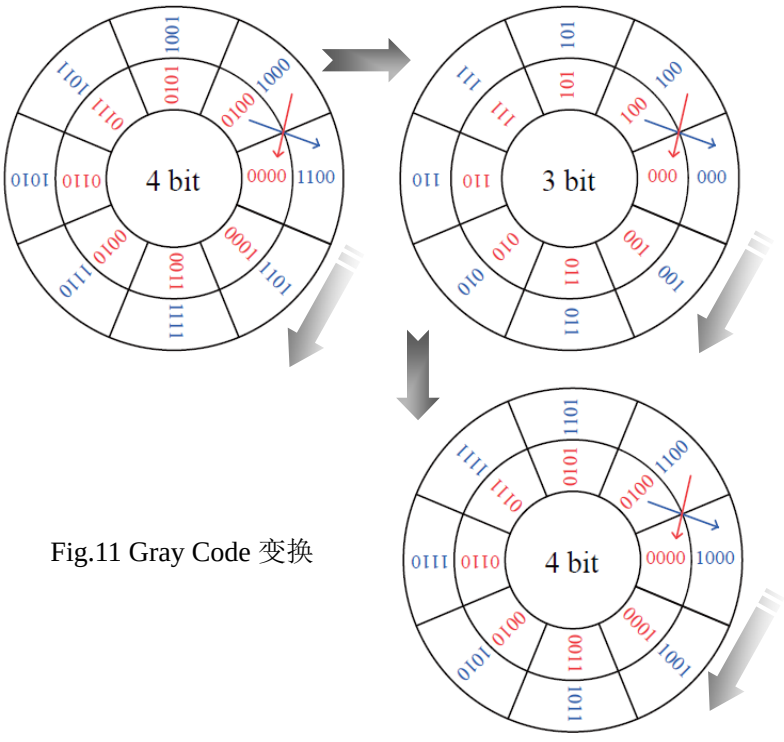


Fig.11 Gray Code 变换

采用 Gray Code 也必须使用 Gray Code 的累加器，然而 Gray Code 的累加器的实现并不容易。在 Fig. 12 中，采用 Binary Code 累加，将得到的 Binary Code 转换成为 Gray Code。加法器的两个加数中，一个是经 Gray Code 转换成为 Binary Code 的数，另一个 1 bit 可以认为是累加器的使能端。当其为 0 时，相加后数字不改变，相当于累加器暂停累加；当其为 1 时，相加后数字增加 1，相当于处于累加状态。控制的使能信号和满或空标志共同作用，当使能为 0 时，数据不写入(或不读出)，则内部的累加器暂停累加，读(写)指针不发生改变。但是当数据写满(读空)指针也不能发生改变，故累加器也需要暂停累加。由前面所提到的 Gray Code 改进方法可以在此图中体现出来，gnext 是 Binary Code 经过转换后成为 Gray Code，将其最高两位进行异或运算成为 msbnext 通过 clk 时钟同步锁存，那么在使用地址译码时仅需要将这个得到的结果 addrmsb 和 ptr(n-2:0)组成新的序列就是正确的了。

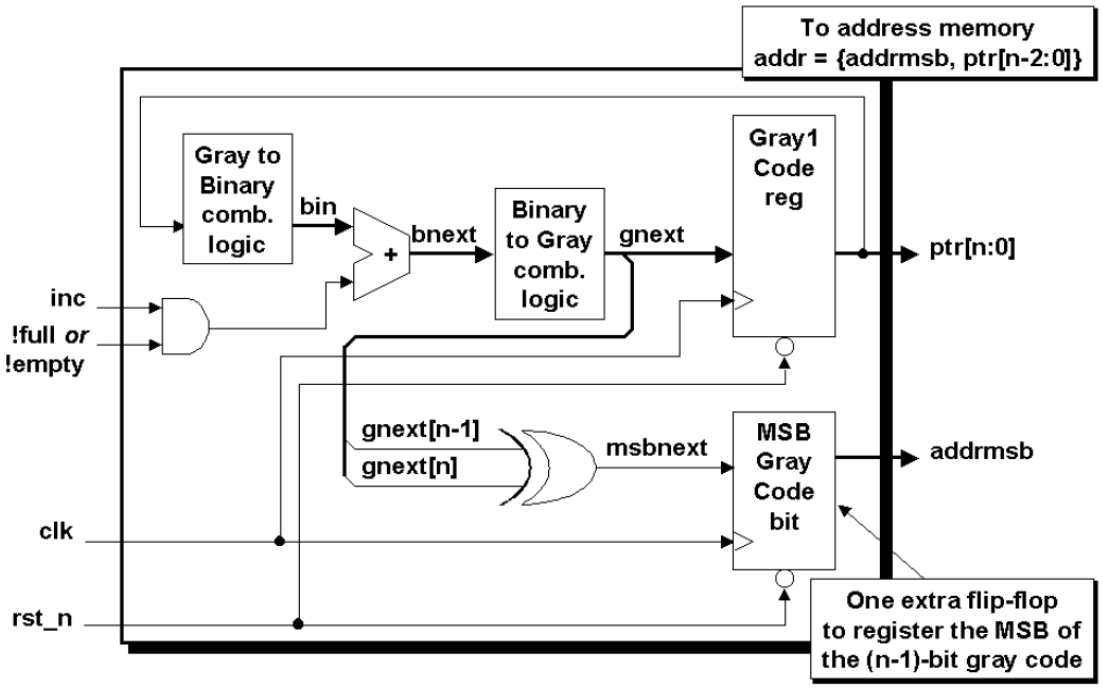


Fig. 12 Gray Code 累加器方案一

还有一种改进的方案值得讨论一下，在上图中加法器的 bin 数据其实是经过 Binary code 到 Gray code 转换经过锁存后再由 Gray Code 到 Binary code 转换，需要用到 2 个转换逻辑电路。如果把寄存器位置改放到加法器的输出位置，把 Binary code 到 Gray code 转换逻辑置于锁存器输出的位置，那么加法器的 bin 输入可以直接从寄存器输出得到。这种改进带来的另一个好处就是不需要产生和寄存 addrmsb 的逻辑，因为 MSB Binary Code 就是 addrmsb。从对于外部电路看来，这种改变带来的唯一区别就是 ptr[n:0]在时钟 clk 上升沿后到改变的 Delay 增加了，但是只要这个 Delay 较短这并不影响整个 FIFO 系统，因为 ptr[n:0]是被一个异步的时钟所采样，增加的 Delay 不产生影响。由 Binary Code 到 Gray Code 转换公式可以的得到如下代码：

```
rgnext = (rbnext>>1) ^ rbnext;
```

这个单元的最大延时为一级异或门的延时。以上改进的方案实现可以见 Fig. 13。

当然，如果希望拥有一个深度为32字节(小规模的)的FIFO时，可在格雷码编码的状态机中手工编写计数器代码或是用查找表的方法来实现都是不错的选择。

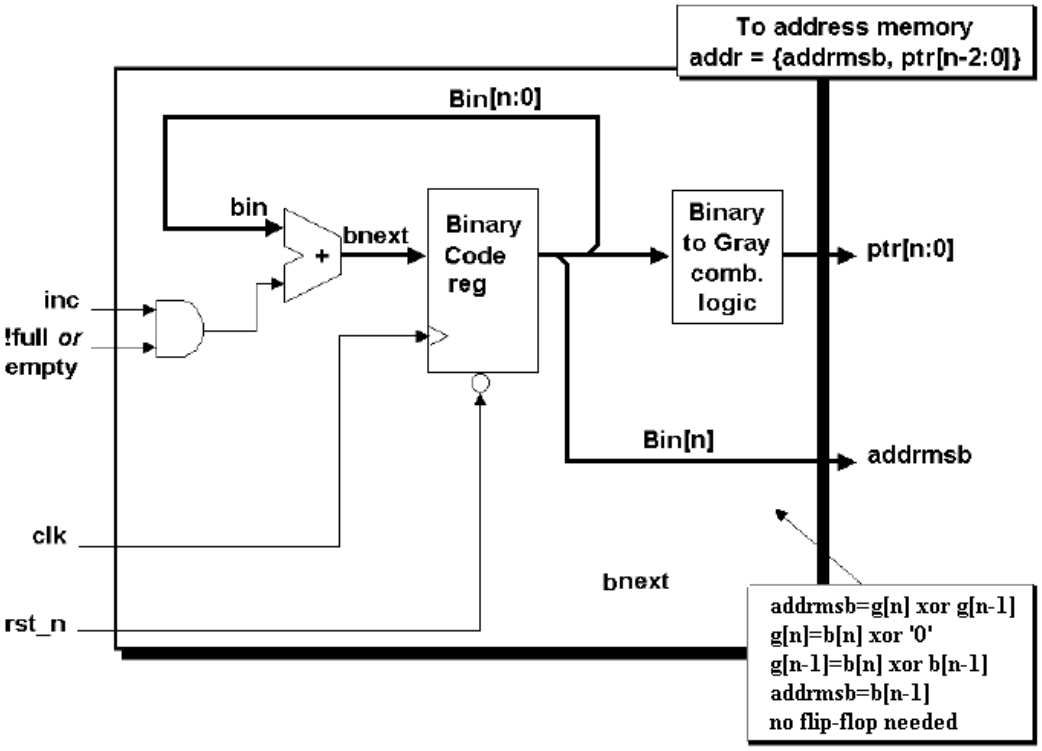


Fig. 13 Gray Code 累加器方案二

III．整体结构

异步 FIFO 整体结构可以分为四大模块：写指针控制(FIFO wptr&full)、读指针控制(FIFO rptr&empty)、双端口 RAM(FIFO Memory Dual Port RAM)和同步采样模块(sync_r2w sync_w2r)。

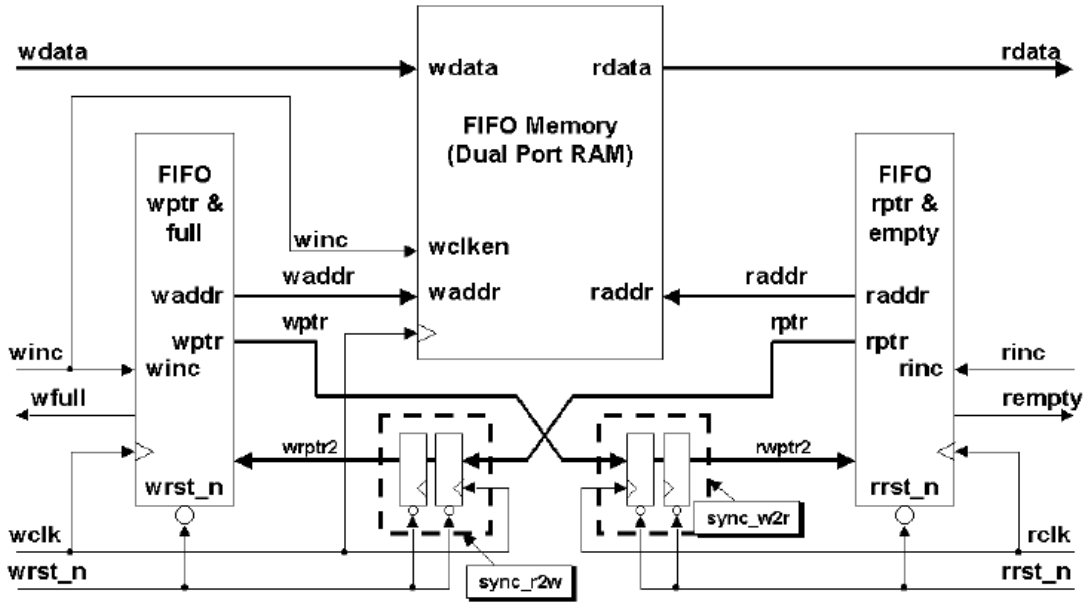


Fig. 14 一种异步 FIFO 整体结构

此结构的中读写指针通过两个同步采样模块交叉传递，使得产生 FULL 和 EMPTY。wptr 和 rptr 是异步的，试分析一种情形，当 rclk 频率远大于 wclk，则 rptr 的更新速度远大于 wclk，那么从 rptr 数据更新后，到经 sync_r2w 同步的 wrptr2 和 wptr 产生 FULL 的时刻，rptr 更新了许多次，即又从 FIFO 中读取了好几个数据，显然此时 FIFO 并不满。相反的，有 EMPTY 产生时，FIFO 并不空的情况。这是一种保守的做法，即等效于提前产生空或满的标志，以保证数据不丢失，这样付出的代价是实际 FIFO 的空间变小了。更一般地情形，是前文提到同步采样模块中会有一个时钟周期地 Latency，那么当假设当写时钟上升沿后产生 FULL 时，是由于写指针赶上了一个同步的读指针，那么 FULL 的信号是准确和即时的。但是在这个系统中，比较输出 FULL 中的读指针是被同步采样的异步信号。当读指针更新一次后，FIFO 不再是满的了，但是 FULL 产生的电路需要在写时钟两个周期后才能将读指针更新到写时钟域中，在此之间除时钟上升沿时刻以外的改变都将无法被同步模块探测到 (如 Fig.15) 。

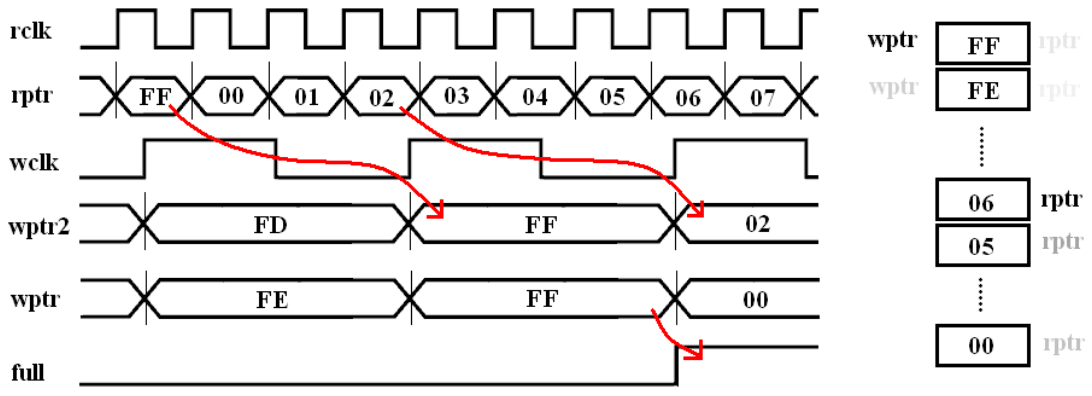


Fig. 15 异步和同步数据对 FULL 影响

在读写时钟差不多的情况下，FIFO 的空间也不可能 100%地利用，理想地情况是可利用的空间比总量小一，我们一般不计较。在有些 FIFO 的设计中不仅会有 FULL 和 EMPTY 还会有如 Almost FULL，Almost EMPTY 信号输出。这一类输出的处理方法就是把读来的指针多加几位进行，在同产生 FULL 和 EMPTY 的方法一样对两个指针进行比较。

waddr，wptr，raddr，rptr 这四个信号都是 Gray Code，注意到前文提到需要采用改进型 Gray Code 编码方式中提到会有两位同时变化的情况，在整体设计中我们依旧在外部的指针传递中采用 Gray Code，能够保证指针信号被同步模块是每次仅有 1bit 的变化，而在读写控制模块内部采用改进型 Gray Code 编码，即将输入的 wrptr2 或 rwptr2 进行从 Gray Code 到改进型的转换，是传递后转换，而不是在输出 wptr，rptr 时进行转换后传递。

异步 FIFO 中复位采用异步 Reset。且在 Reset 后各个信号初始值的确定没有多少问题，例如指针值 reset 后无论是多少都无碍大妨，在环形的队列中，只要读写指针 Reset 后相同就可以了。FIFO 的关键输出 FULL 和 EMPTY 在 reset 后为了保守考虑都为’1’较为合理。

IV．测试和结果

测试一个异步的 FIFO 设计正确性几乎很困难，原因就在于即使 FIFO 指针在 RTL 的行为仿真是十分理想的，但是如果错误地 RTL 映射到实现的器件，关键路径的时序约束设计不合理将会导致致命的错误。

在 RTL 级仿真中，如果前面提到的控制指针的 Gray Code 累加器中 Binary－count 将会同步地改变所有的比特位想要查找出发生同步或者比较错误。在没有寄生参数延时的门级仿真中，如果在时钟上升下降沿的不同，也仅仅有很小的几率能够发现问题的存在。而对于更高速的设计，信号上升下降的延时之间的区别更加小了，发现问题的可能性也降低了。所以实际的 FIFO 设计问题最好能够采用带有寄生参数的门级仿真，但即使是这样，一旦发现问题，有时候也能难再现错误。

基于以上考虑可以对 FIFO 进行功能上的仿真，再采用 FPGA 来验证测试 FIFO，来模拟在真实器件中 FIFO 的特性。采用 Xilinx Spartan 3 系列 xc3s400-4pq208，综合工具 XST。综合的结果报告中：

Timing Summary:----------
Speed Grade: -4
Minimum period: 9.302ns (Maximum Frequency: 107.504MHz)
Minimum input arrival time before clock: 9.416ns
Maximum output required time after clock: 12.762ns
Maximum combinational path delay: No path found
其中
Timing constraint: Default period analysis for Clock 'wclk'
Clock period: 9.302ns (frequency: 107.504MHz)
Total number of paths / destination ports: 1402 / 667
---
Timing constraint: Default period analysis for Clock 'rclk'
Clock period: 8.912ns (frequency: 112.205MHz)
Total number of paths / destination ports: 260 / 21

在更为详细的报告中可以看到对于 FULL 和 EMPTY 信号通路是关键路径决定整体 FIFO 最高工作时钟，如果需要进一步提高 FIFO 的速度其重点是要对 FULL 和 EMPTY 加以改进

数据写入读出的正确性取决于对 FIFO 模块中双端口 RAM 的操作。

但是前面提到，仅仅是看仿真结果是很难发现问题，基于此本文设计的一种测试思路是设计一个发送一个接收的控制器，发送控制器用于发送连续循环的数据从 00～FF 到 FIFO，而接收模块用于从 FIFO 读出数据并进行数据的校验。发送的是连续的数据码流，那么如果接收到的数据流中检测出数据的不连续，那么就有错误 ERROR 信号产生。发送和接收模块分别使用不同频率不同相位的时钟，然后采用 Xilinx Chipscope 对 FPGA 内中的信号进行测试。如果在 Chipscope 中未发现 ERROR 信号为高时，不能够完全说明不产生 ERROR 因为 Chipscope 的采样深度有限，且只能采用全局时钟作为采样时钟。可以将 ERROR 从 FPGA 管脚引出，用示波器的单次触发功能对其进行测量，如果示波器长时间地没有被触发，则表明 FIFO 工作一切正常。

从 Chipscope 中记录的数据中来看，如果读写时钟的稳定性很好时，FULL 和 EMPTY 的信号具有周期性，且有如下的关系

$$f_{FULL} = \frac{f_{wclk} - f_{rclk}}{2}, \quad f_{wclk} > f_{rclk} \qquad f_{EMPTY} = \frac{f_{rclk} - f_{wclk}}{2}, \quad f_{wclk} < f_{rclk} \qquad \text{Eq. 5}$$

必然地，由于写入读出的速度之差，当写时钟大于读时钟时，FIFO 一定会被写满，然后产生 FULL，反之写时钟大于读时钟时，FIFO 一定会被读空，然后产生 EMPTY．当读写时钟相同时 FULL 和 EMPTY 的频率不仅取决读写时钟之间的相位差，还取决于系统的特性很难加以计算出。而这个相位差具有一定的随机性，所以 FULL 和 EMPTY 很难观测出周期性，而更带有一定的随机性。在 FIFO 复位以后，读写指针都归零，RAM 内数据为空，一旦 FIFO 满或空后就进入一个稳定的状态，驰豫时间取决于读写时钟频率之差的。

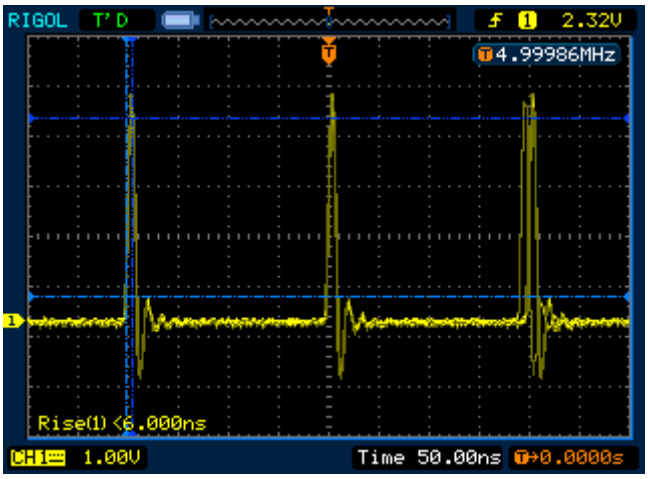


Fig. 16 读 110MHz 写 120MHz 下的 FULL 信号

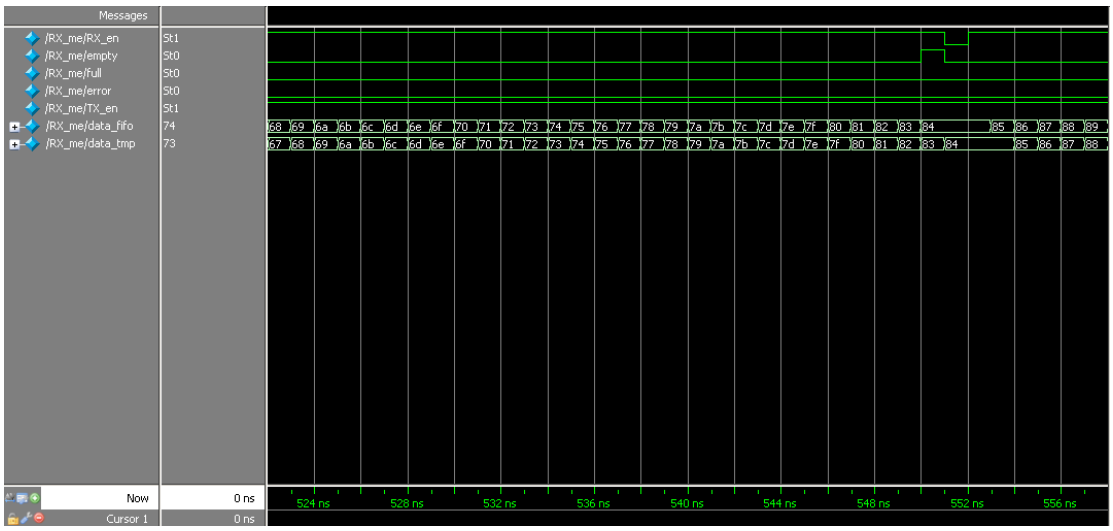


Fig. 17 读空后的情形

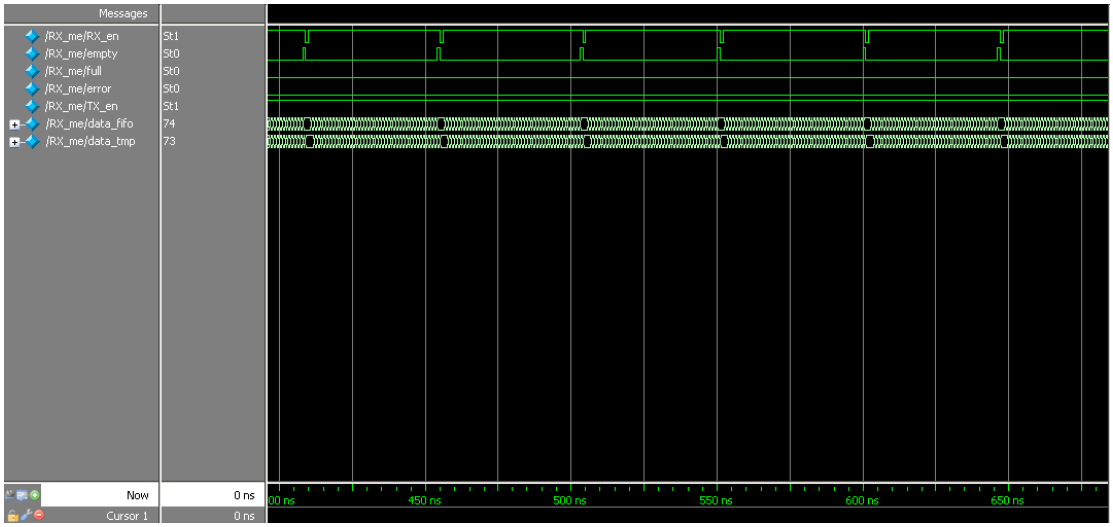


Fig. 18　EMPTY 的周期性

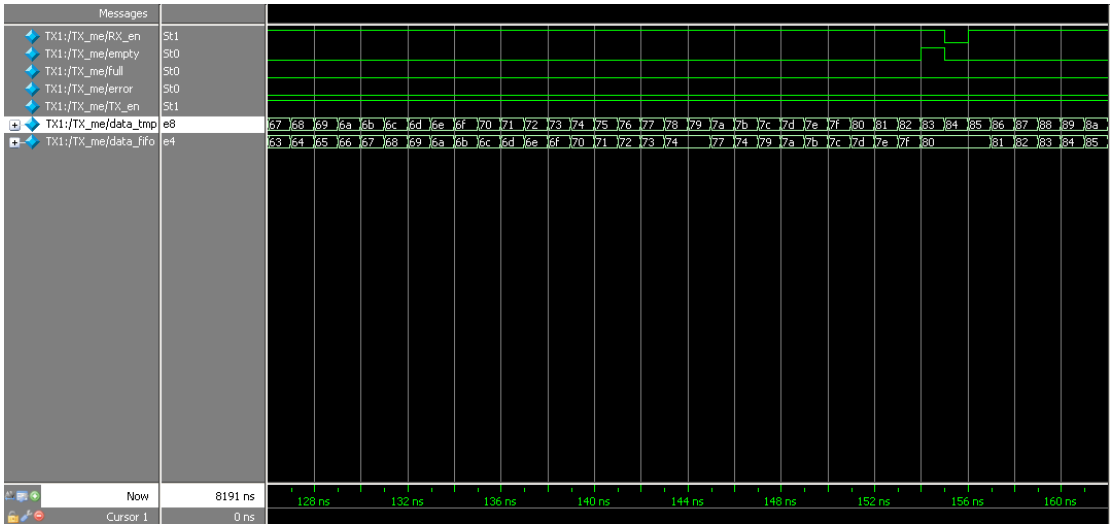


Fig. 19 发送端数据流

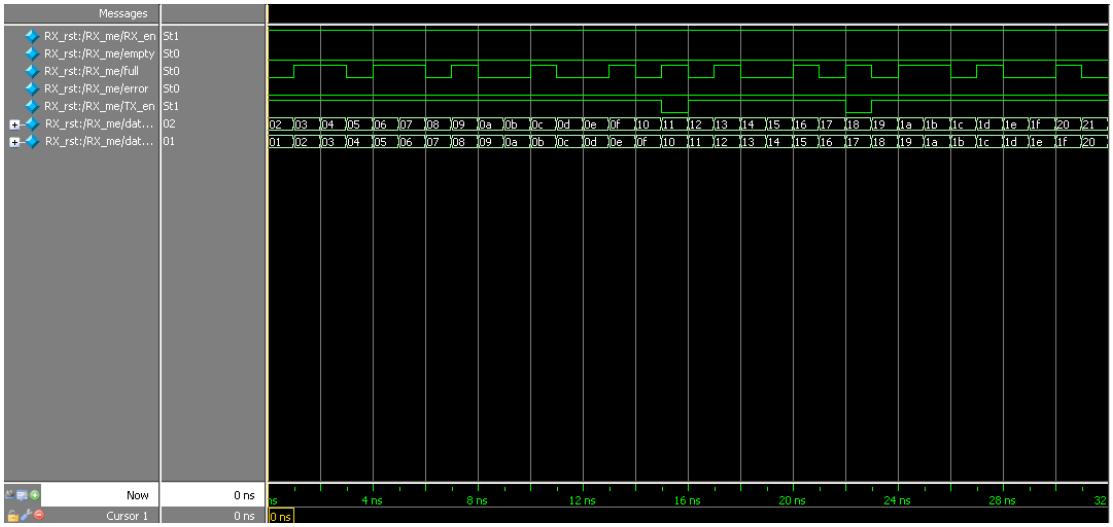


Fig. 20　Reset 后 FULL 的随机性

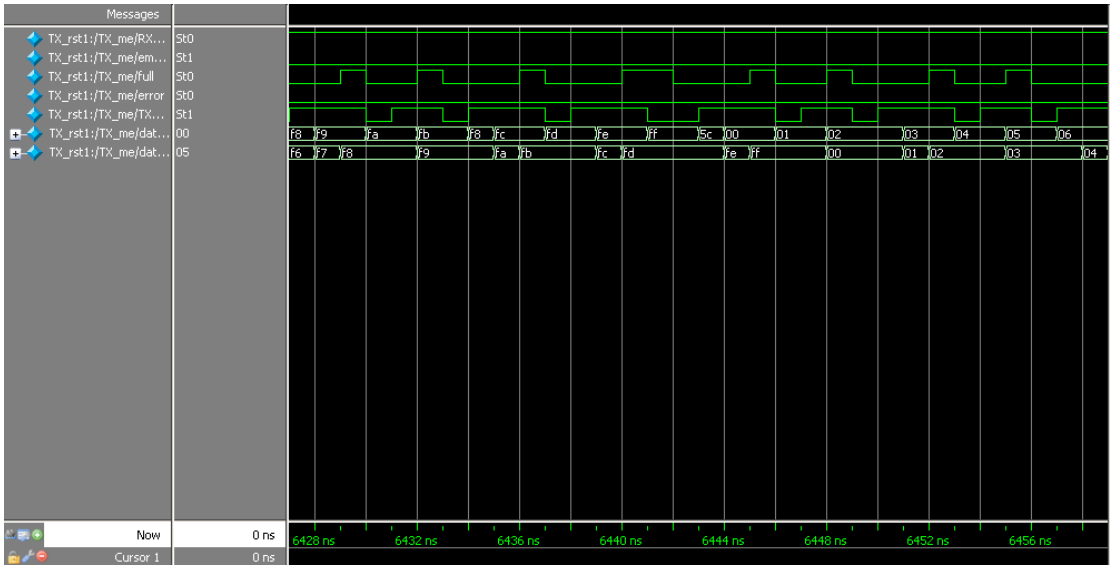


Fig. 21 Reset 后 FULL 的随机性

在验证 FIFO 功能正确性时，手工验证时，只要观察 RX 模块中 data_tmp 是否连续，或者直接观察 ERROR 信号是否为'1'。由于 Chipscope 只能用一个时钟采样信号，所以没有办法同时采样异步的发送和接收端数据信号，所以只能分成发送、接受两次测试。注意以上的波形图中 TX 代表发送端，RX 代表接受端故，在 TX 中 EMPTY 和 RX_EN 是异步的采样的数据是不可信的，在 RX 中 FULL 和 TX_EN 是异步的采样的数据是不可信的。图中信号 data_fifo 始终表示的是从 FIFO 到 RX 端的数据，所以在一 TX 时钟采样时是不可信的。

(详细实现 FIFO 代码，测试模块代码以及测试数据结果见附件)