FIFO 控制

Our goal: help making good designs

1. 前言

在熟悉 SRAM 功能与控制时序后，我们来看另外一个在数字 IP 设计中经常使用的模块：FIFO。SRAM 用于数据的暂时存储，读写地址都由 controller 控制，可以随机访问任意地址。FIFO 是 first in first out 的缩写，其读写有特定的顺序：先进先出。基于“先进先出”这个数据访问顺序的限制，FIFO 的功能与控制时序跟 SRAM 相比是不同的。当然，FIFO 也有数据缓存的能力。

2. FIFO 结构框图

我们以同步 FIFO 为例，来阐述 FIFO 的功能与控制时序。一个同步 FIFO 其 IO 通常如下：

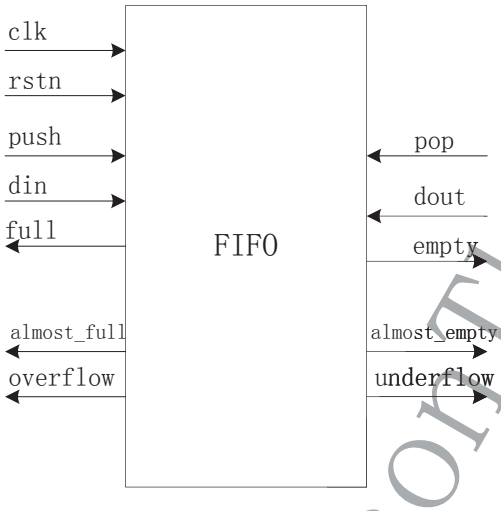
必要的 IO 接口：

Name	Direction	Bits	Description
clk	I	1	时钟输入。
rstn	I	1	复位输入，low active。
push	I	1	FIFO 写使能，high level active。
din	I	DWIDTH	FIFO 写数据。
full	O	1	FIFO 满标志，high active。
pop	I	1	FIFO 读使能，high level active。
dout	O	DWIDTH	FIFO 读数据。
empty	O	1	FIFO 空标志，high active。

可选的 IO 接口：

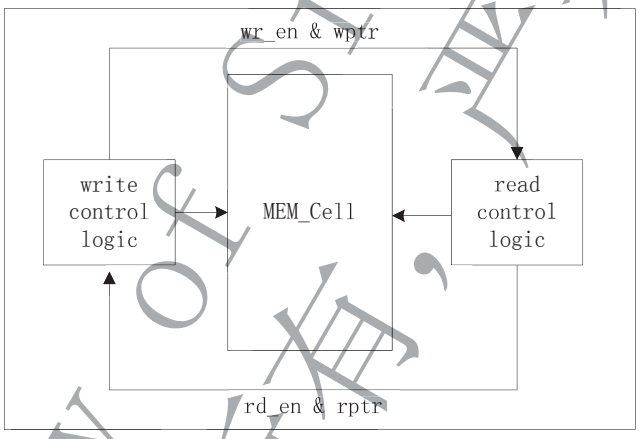
almost_full	O	1	FIFO 即将满标志，high active。
overflow	O	1	FIFO 写溢出，high active。
almost_empty	O	1	FIFO 即将空标志，high active。
underflow	O	1	FIFO 读读下溢，high active。

同步 FIFO 的 block diagram 如下：



3. FIFO 内部结构

一个 FIFO，其内部结构大致如下：

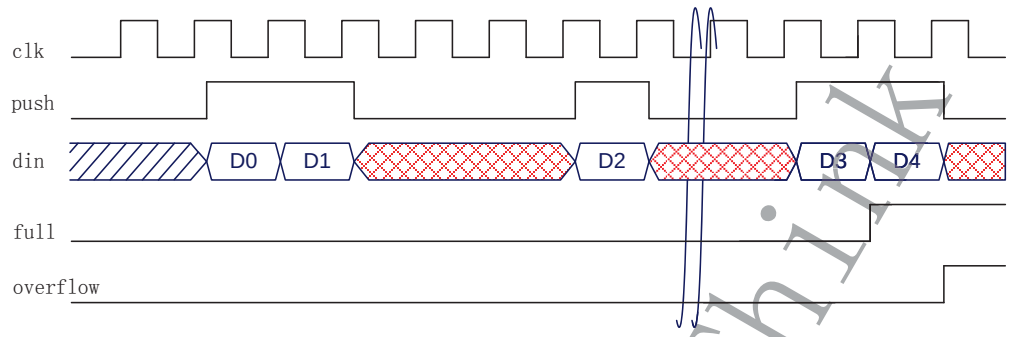


其中：

1) MEM_cell: 是其数据存储部分。根据 FIFO 深度（最大缓存数据的个数）不同，可以选用 register file 或 SRAM macro 来实现。
2) write control logic：把 PUSH 的 din 写入 MEM_cell，并且产生 PUSH 对应的 flags。
3) read control logic：把 POP 需要的 dout 从 MEM_cell 读出，并且产生 POP 对应的 flags。
4) read/write control logic 会有一些信号的交互，来协助产生 PUSH/POP 对应的 flags。这些交互信号，通常是：wr_en/write pointer(wptr)/rd_en/read pointer(rptr)。

4. FIFO 控制时序

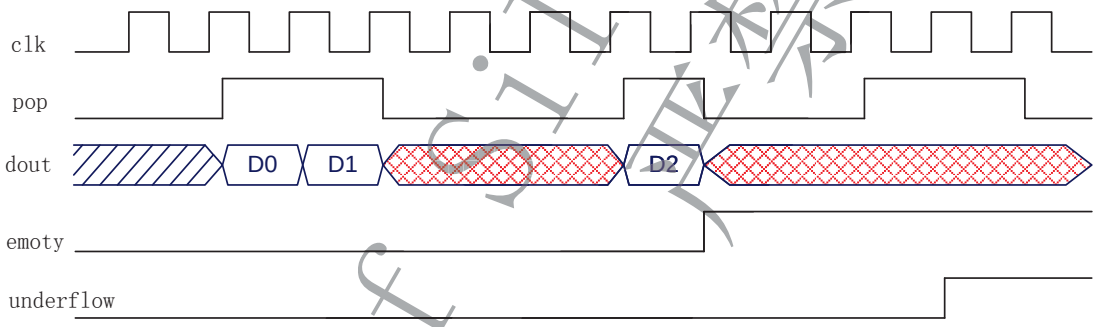
要明白怎么使用 FIFO，在明白其功能与工作原理后，再看看其 IO 上的接口时序。
PUSH 端的时序如下图：



"push"/"din"同时有效，把数据写入 FIFO；在写入一定数量后（假设期间没有 POP 操作），FIFO 的"full" flag 会拉高，表示 FIFO 已经满了（MEM_cell 已经没有空间装新的数据了）；如果在 full==1'b1 时，再来一个 push == 1'b1，则 FIFO 的 overflow flag 会拉高，表示 FIFO 已经产生了溢出错误，已经导致了数据丢失，甚至冲掉了以前的数据（具体行为，根据具体 FIFO 的设计而定）。

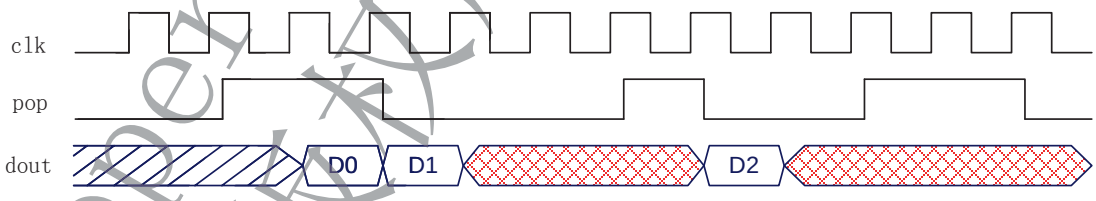
正常情况下，控制 FIFO 的 user control logic 是不能让 FIFO 发生 overflow 的；如果发生 overflow，一般需要 reset FIFO（rstn 拉低），使 FIFO 从 overflow 中恢复。

POP 端的时序如下图：

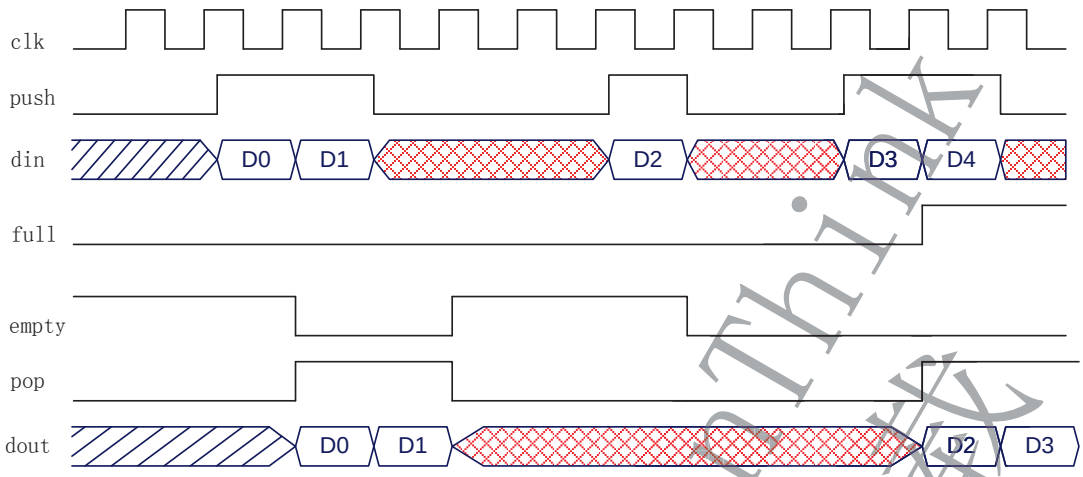


"pop"有效时，把一个 dout 上的数据读走；当 POP 一定数量的数据后（假设期间没有 PUSH 操作），FIFO 的"empty" flag 会拉高，表示 FIFO 已经空了（MEM_cell 里面所以的数据已经被取走）；如果在 empty==1'b1 时，再来一个 pop == 1'b1（如上图最后 2 次 POP 操作），则 FIFO 的 underflow flag 会拉高，表示 FIFO 已经产生了下溢错误，POP 得到是错误数据，甚至导致 FIFO 内部控制逻辑混乱，影响后续的 PUSH/POP 操作（具体行为，根据具体 FIFO 的设计而定）。

根据 FIFO 设计的不同，"dout"也可能在"pop"的下一个时钟有效，见下图：



FIFO 在实际使用时，PUSH/POP 操作是可以随机发生的，相互之间并无具体需求。user control logic 只需要保证 FIFO 不会发生 overflow/underflow 即可。示例波形如下：



5. FIFO 设计

在理解 FIFO 的功能，接口时序后，是不是想自己设计个 FIFO 呢？如果有这个冲动，就自己动手吧，一定可以的。 ^_^

在设计时，可以考虑使用 verilog 的 parameter 语法，让你设计的 FIFO 的 MEM_cell depth, Data width 是可以 parameter 配置的，这样就是一个灵活的 FIFO IP，也符合实际使用需求。

6. FIFO 的兄弟 STACK

FIFO 还有一个兄弟：STACK（堆栈）。其功能与 FIFO 类是，只是 last in, first out（后进先出）。其接口与 FIFO 类似，好奇的自己研究吧。

顺带打个广告：

a) 《数字 IC/FPGA 设计入门_合集》：https://ke.qq.com/course/3133628?tuin=64ce5e2a

b) 《PPGA 设计入门》：https://ke.qq.com/course/3067626?tuin=64ce5e2a

c) 《On-Chip-Bus 精讲》：https://ke.qq.com/course/2900266?tuin=64ce5e2a

d) 《数字 IP 设计实例_A》：https://ke.qq.com/course/3132227?tuin=64ce5e2a

e) 《数字 IP 设计实例_B》：https://ke.qq.com/course/3200590?tuin=64ce5e2a

f) 《数字 IP_FPGA 设计实战》：https://ke.qq.com/course/3292002?tuin=64ce5e2a

---

群主介绍（QQ 技术交流群：877205676）：

sky：2006 年电子科大毕业；前 Verisilicon Senior Staff Engineer；数字电路前端设计从业 14 年；主要做视频 IP 设计（H.264/H.265 编解码器设计，JPEG 编解码器设计），CNN 加速器 IP 设计。参与 7 颗 ASIC/SOC 芯片设计（量产 3 颗）。目前申请 3 篇国家发明专利。

公司主页：http://www.siliconthink.cn