SoEZ-USB2.0(CY7C68013A) Slave-FIFO开发文档

Version (1.0)

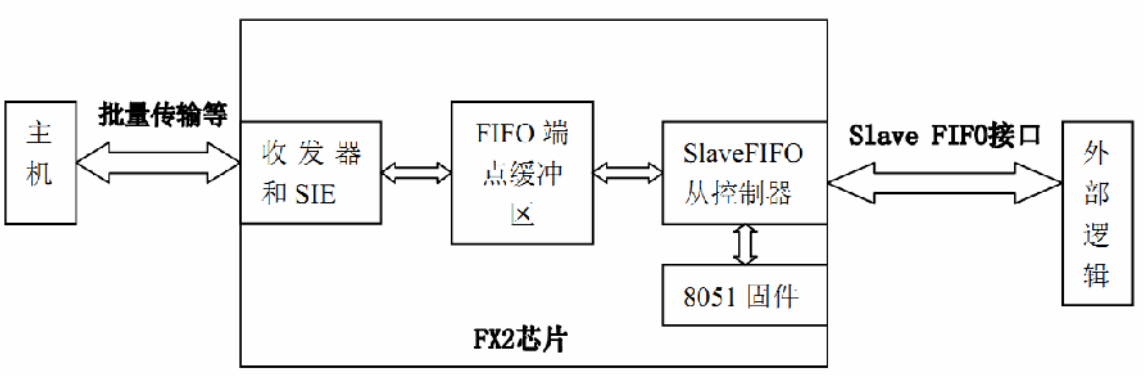
网址：http://soez.taobao.com
联系 QQ：120076915

目录

一、Slave FIFO 传输

1．1 概述

当有一个与 FX2LP 芯片相连的外部逻辑只需要利用 FX2LP 作为一个 USB 2.0 接口而实现与主机的高速通讯，而它本身又能够提供满足 Slave FIFO 要求的传输时序，可以做为 Slave FIFO 主控制器时，即可考虑用此传输方式。
Slave FIFO 传输的示意图如下：



Slave FIFO传输示意图

在这种方式下，FX2LP 内嵌的 8051 固件的功能只是配置 Slave FIFO 相关的寄存器以及控制 FX2LP 何时工作在 Slave FIFO 模式下。一旦 8051 固件将相关的寄存器配置完毕，且使自身 工作在 Slave FIFO 模式下后，外部逻辑（如 FPGA）即可按照 Slave FIFO 的传输时序，高 速与主机进行通讯，而在通讯过程中不需要 8051 固件的参与。

1．2 硬件连接(标准)

在 Slave FIFO 方式下，外部逻辑与 FX2LP 的连接信号图如下：

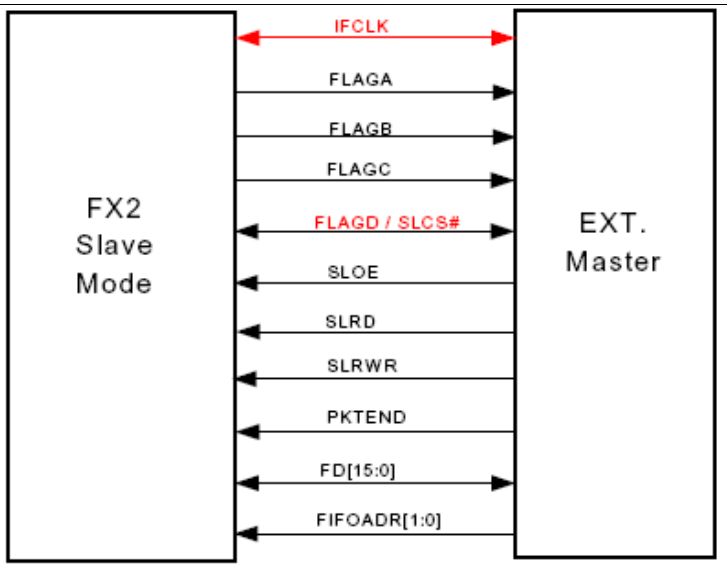


Figure 9-2. FX2 Slave Mode Full-Featured Interface Pins

IFCLK：FX2LP 输出的时钟，可做为通讯的同步时钟；

FLAGA，FLAGB，FLAGC，FLAGD：FX2LP 输出的 FIFO 状态信息，如满，空等；

SLCS：FIFO 的片选信号，外部逻辑控制，当 SLCS 输出高时，不可进行数据传输；

SLOE：FIFO 输出使能，外部逻辑控制，当 SLOE 无效时，数据线不输出有效数据；

SLRD：FIFO 读信号，外部逻辑控制，同步读时，FIFO 指针在 SLRD 有效时的每个 IFCLK 的上升沿递增，异步读时，FIFO 读指针在 SLRD 的每个有效—无效的跳变沿时递增；

SLWR：FIFO 写信号，外部逻辑控制，同步写时，在 SLWR 有效时的每个 IFCLK 的上 升沿时数据被写入，FIFO 指针递增，异步写时，在 SLWR 的每个有效—无效的跳变沿时数 据被写入，FIFO 写指针递增；

PKTEND：包结束信号，外部逻辑控制，在正常情况下，外部逻辑向 FX2LP 的 FIFO 中写数，当写入 FIFO 端点的字节数等于 FX2LP 固件设定的包大小时，数据将自动被打成一包进行传输，但有时外部逻辑可能需要传输一个字节数小于 FX2LP 固件设定的包大小的包，这时， 它只需在写入一定数目的字节后，声明此信号，此时 FX2LP 硬件不管外部逻辑写入了多少字 节，都自动将之打成一包进行传输；

FD[15:0]：数据线；

FIFOADR[1:0]：选择四个 FIFO 端点的地址线，外部逻辑控制。

1．3 Slave FIFO 的几种传输方式

1．3．1 同步 Slave FIFO 写

同步 Slave FIFO 写的标准连接图如下：

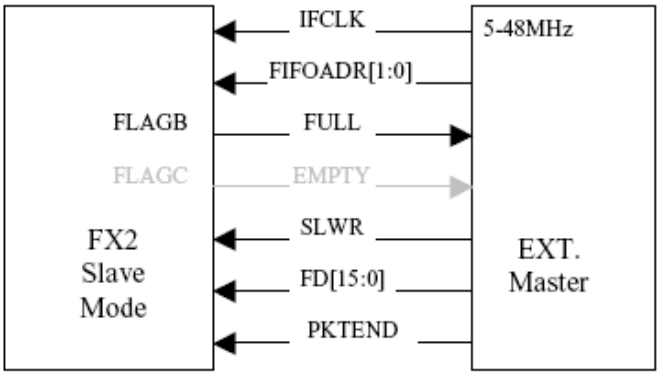


Figure 9-10. Interface Pins Example: Synchronous FIFO Writes

同步 Slave FIFO 写的标准时序如下：
IDLE：当写事件发生时，进状态 1；
状态 1：使 FIFOADR[1:0]指向 IN FIFO，进状态 2；
状态 2：如 FIFO 满，在本状态等待，否则进状态 3；
状态 3：驱动数据到数据线上，使 SLWR 有效，持续一个 IFCLK 周期，进状态 4；
状态 4：如需传输更多的数，进状态 2，否则进状态 IDLE。
状态跳转示意图如下：

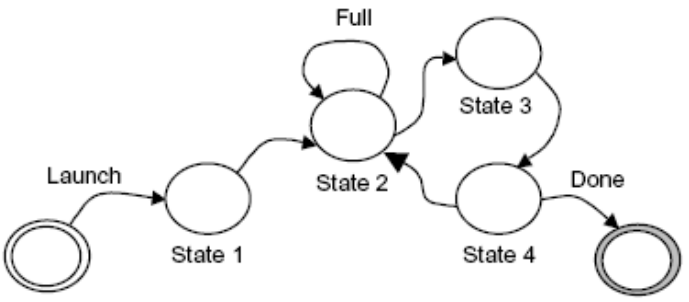


Figure 9-11. State Machine Example: Synchronous FIFO Writes

几种情况的时序图示意如下（FULL，EMPTY，SLWR，PKTEND 均假定低有效）：

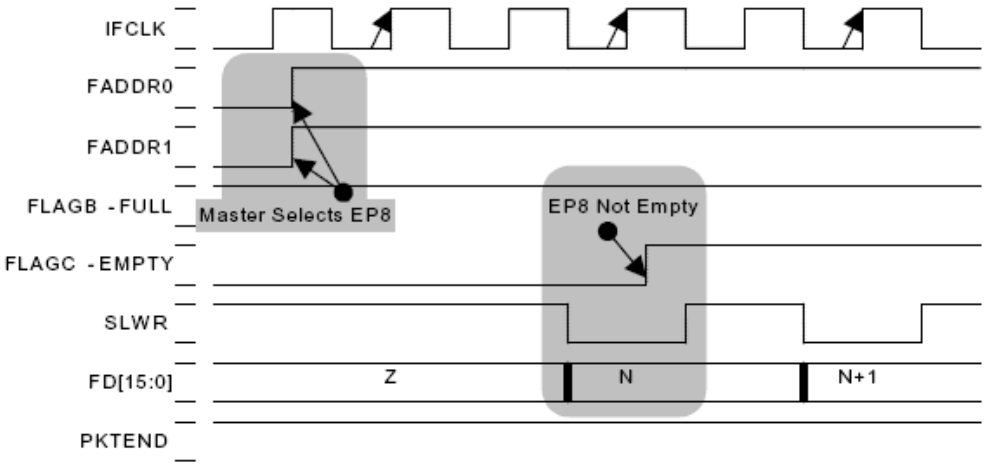


Figure 9-12. Timing Example: Synchronous FIFO Writes, Waveform 1

图示 FIFO 中本来没有数据，外部逻辑写入第一个数据时的情况。

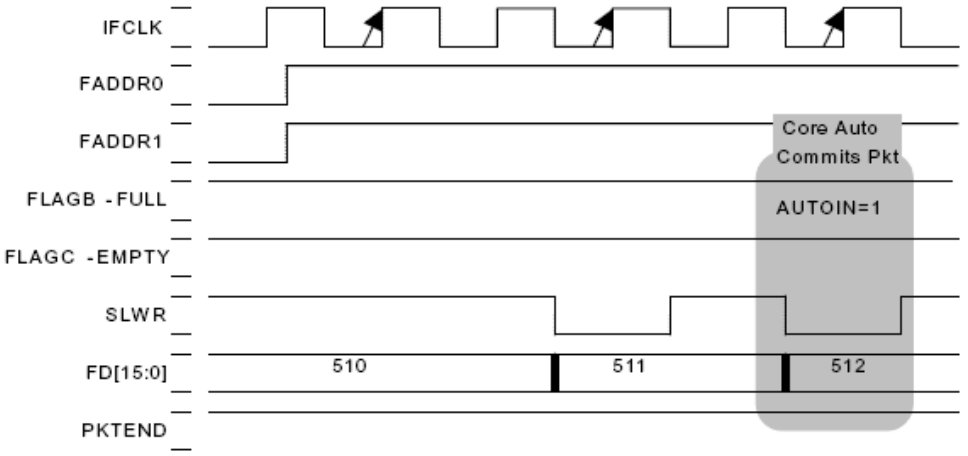


Figure 9-13. Timing Example: Synchronous FIFO Writes, Waveform 2

图示假定 FX2LP 设定包大小为 512 字节，外部逻辑向 FIFO 端点中写入的数据达 512 字节时的情况。此时 FX2LP 硬件自动将已写入的 512 字节打成一包准备进行传输，这个动作就和 在普通传输中，FX2LP 固件向 FIFO 端点中写入 512 字节后，把 512 这个数写入 EPxBC 中一 样，只不过这个过程是由硬件自动完成的。在这里可以看出“FX2LP 固件不参与数据传输过程” 的含义了。外部逻辑只须按上面的时序图所示的时序向 FIFO 端点中一个一个字节（或字） 地写数，写到一定数量，FX2LP 硬件自动将数据打包传输，这一切均不需固件的参与，由此实 现高速数据传输。

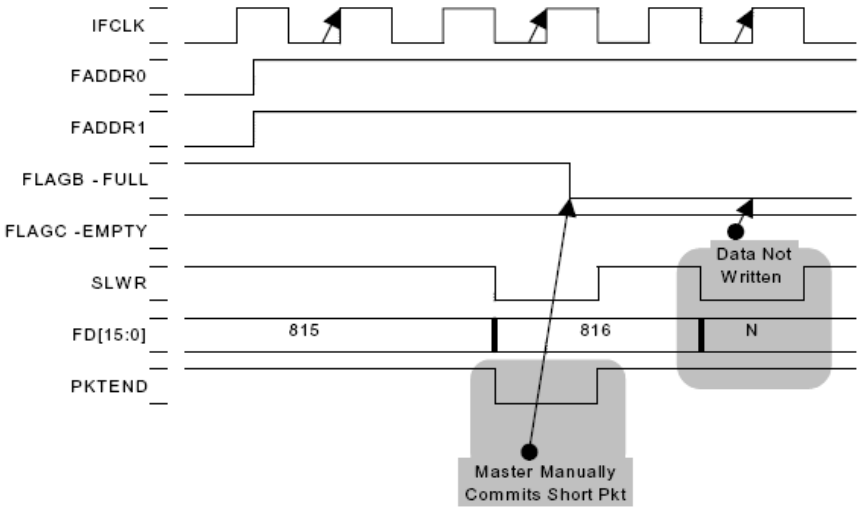


Figure 9-14. Timing Example: Synchronous FIFO Writes, Waveform 3, PKTEND Pin Illustrated

图示的是 FIFO 端点被写满时的情况。

1．3．2 同步 Slave FIFO 读：

同步 Slave FIFO 读的标准连接图如下：

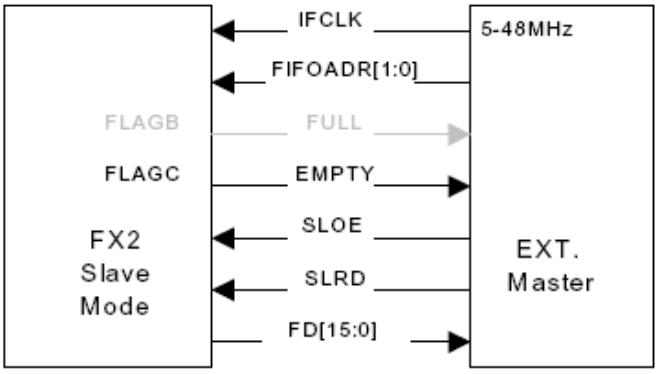


Figure 9-15. Interface Pins Example: Synchronous FIFO Reads

同步 Slave FIFO 读的标准时序如下：

IDLE：当读事件发生时，进状态 1；

状态 1：使 FIFOADR[1:0]指向 OUT FIFO，进状态 2；

状态 2：使 SLOE 有效，如 FIFO 空，在本状态等待，否则进状态 3；

状态 3：从数据线上读数，使 SLRD 有效，持续一个 IFCLK 周期，以递增 FIFO 读指针，进状态 4；

状态 4：如需传输更多的数，进状态 2，否则进状态 IDLE。

状态跳转示意图如下：

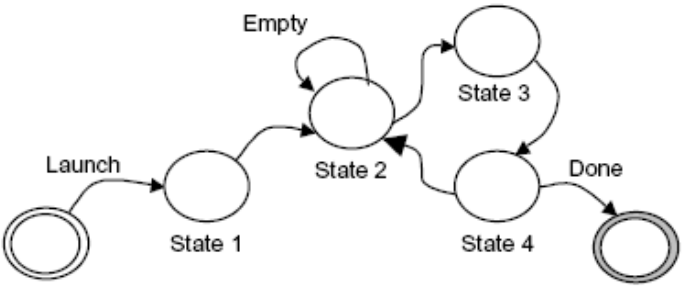


Figure 9-16. State Machine Example: Synchronous FIFO Reads

几种情况的时序图示意如下（FULL，EMPTY，SLRD，SLOE 均假定低有效）：

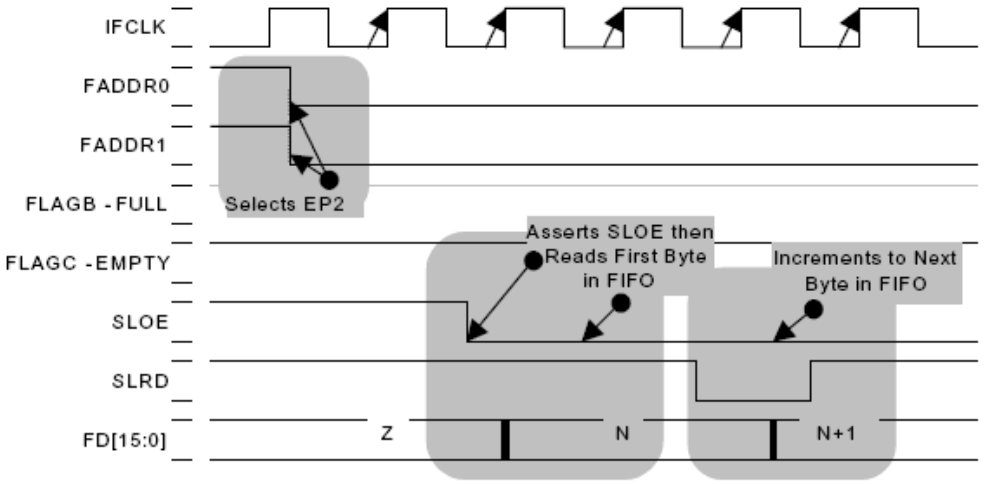


Figure 9-17. Timing Example: Synchronous FIFO Reads, Waveform 1

图示正常情况时的时序。

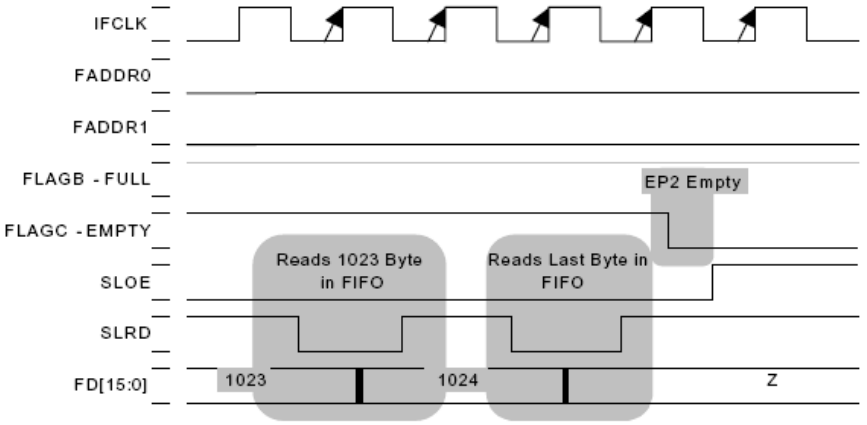


Figure 9-18. Timing Example: Synchronous FIFO Reads, Waveform 2, EMPTY Flag Illustrated

图示 FIFO 被读空时的情况。

1．3．3 异步 Slave FIFO 写：

异步 Slave FIFO 写的标准连接图如下：

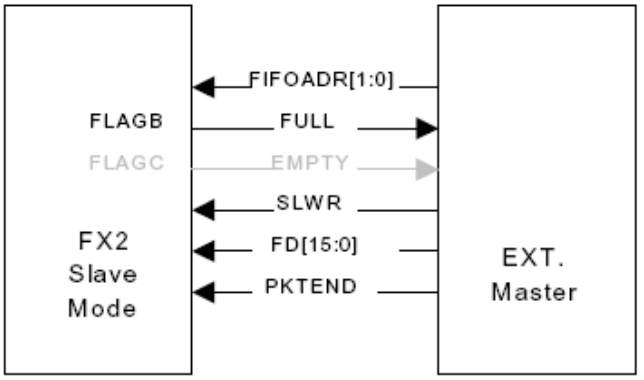


Figure 9-19. Interface Pins Example: Asynchronous FIFO Writes

异步 Slave FIFO 写的标准时序如下：
IDLE：当写事件发生时，进状态 1；
状态 1：使 FIFOADR[1:0]指向 IN FIFO，进状态 2；
状态 2：如 FIFO 满，在本状态等待，否则进状态 3；
状态 3：驱动数据到数据线上，使 SLWR 有效，再无效，以使 FIFO 写指针递增，进状 态 4；
状态 4：如需传输更多的数，进状态 2，否则进状态 IDLE。
状态跳转示意图如下：

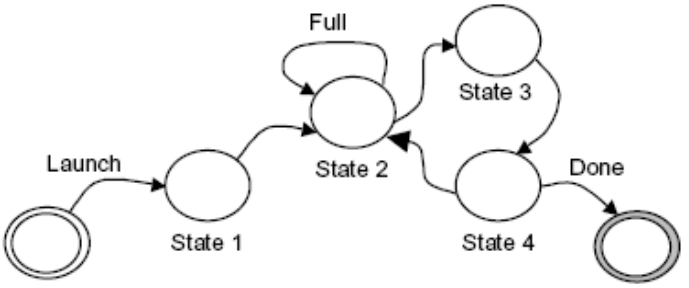


Figure 9-20. State Machine Example: Asynchronous FIFO Writes

几种情况的时序图示意如下（FULL，EMPTY，SLWR，PKTEND 均假定低有效）：

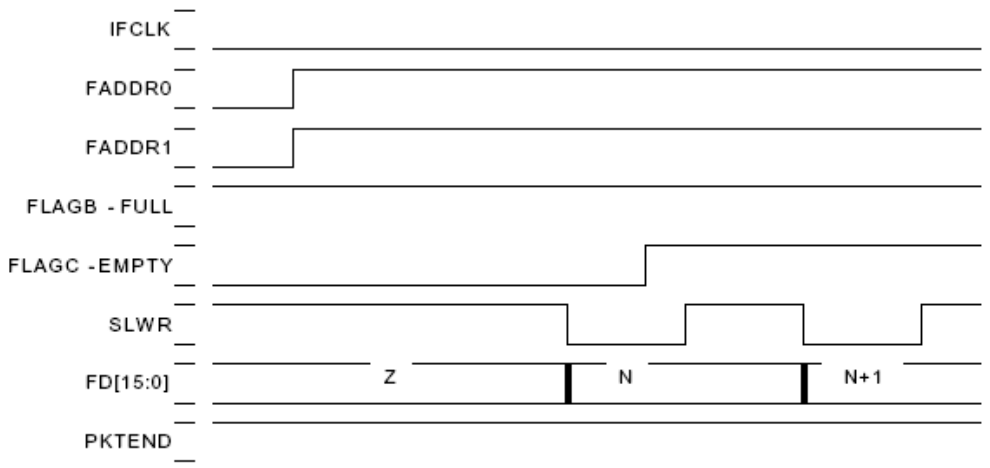


Figure 9-21. Timing Example: Asynchronous FIFO Writes

图示 FIFO 中本来没有数据，外部逻辑写入第一个数据时的情况。

1．3．4 异步 Slave FIFO 读：

异步 Slave FIFO 读的标准连接图如下：

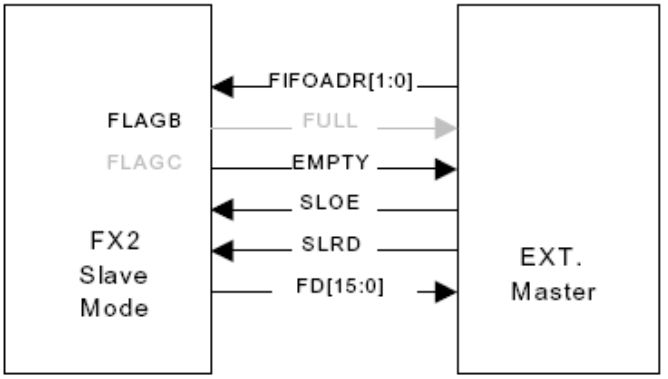


Figure 9-22. Interface Pins Example: Asynchronous FIFO Reads

异步 Slave FIFO 读的标准时序如下：
IDLE：当读事件发生时，进状态 1；
状态 1：使 FIFOADR[1:0]指向 OUT FIFO，进状态 2；
状态 2：如 FIFO 空，在本状态等待，否则进状态 3；
状态 3：使 SLOE 有效，使 SLRD 有效，从数据线上读数，再使 SLRD 无效，，以递增 FIFO 读指针，再使 SLOE 无效，进状态 4；
状态 4：如需传输更多的数，进状态 2，否则进状态 IDLE。
状态跳转示意图如下：

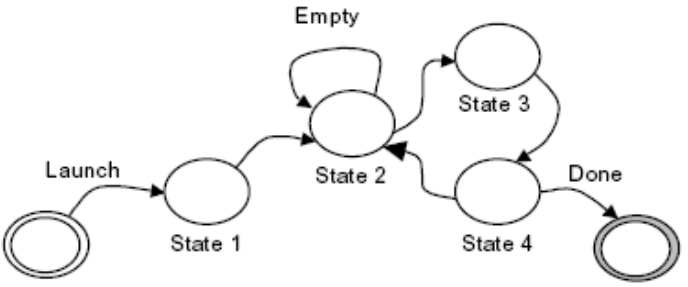


Figure 9-23. State Machine Example: Asynchronous FIFO Reads

几种情况的时序图示意如下（FULL，EMPTY，SLRD，SLOE 均假定低有效）：

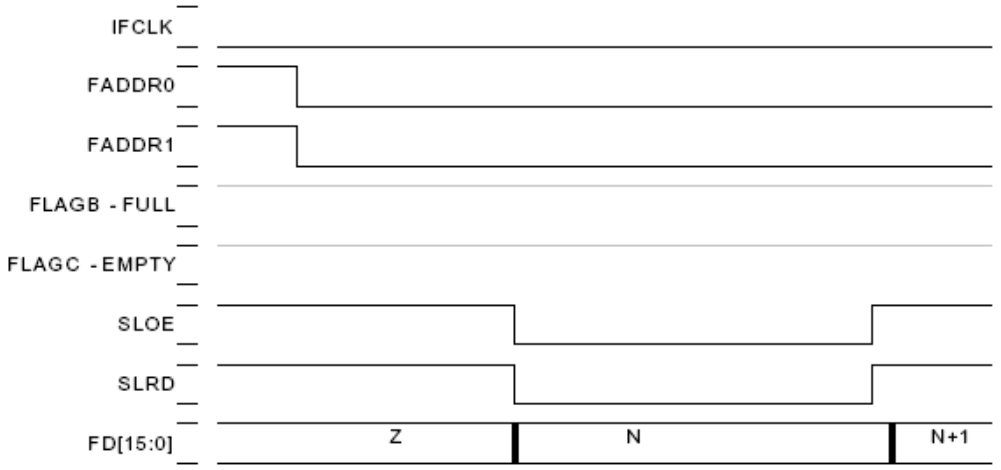


Figure 9-24. Timing Example: Asynchronous FIFO Reads

图示正常情况时的时序。

二、寄存器设置

slave fifo 模式下常用寄存器

IFCONFIG	EPxFIFOPFH/L
PINFLAGAB	PORTACFG
PINFLAGCK	INPKTEND
FIFORESET	EPxFLAGIE
FIFOPINPOLAR	EPxFLAGIRQ
EPxCFG	EPxFIFOBCH:L
EPxFIFOCFG	EPxFLAGS
EPxAUTOINLENH:L	EPxBUF

2．1 IFCONFIG：接口配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
IFCLKSRC	**3048MHZ**	**IFCLKOE**	**IFCLKPOL**	**ASYNC**	**GSTATE**	**IFCFG1**	**IFCFG0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	0	0	0	0	0	0

IFCLKSRC：FIFO 时钟内部/外部时钟源选择，0 外部时钟源，1 内部时钟源。
3048MHZ：如选择内部时钟，30MHz/48MHz 频率选择，0 IFCLK 时钟 30M，1 =48M。
IFCLKOE：IFCLK 时钟输出使能，0 关闭，1 打开。
IFCLKPOL：IFCLK 输出反转使能，0 不反转，1 反转。
ASYNC：Slave FIFO 同步/异步工作方式选择，0 同步，1 异步。
GSTATE：选择是否将 GSTATE[2:0]在 PORTE[2:0]输出，0 关闭，1 使能。
IFCFG1:0：FX2LP I/O 端口模式选择，也既是上面所说的 FX2LP 与外部逻辑传输方式的选择。

IFCFG1	IFCFG0	Configuration
0	0	普通端口模式
0	1	保留
1	0	GPIF（GPIF Interface）
1	1	Slave FIFO 模式

2．2 PINFLAGS AB/CD：FLAGx 引脚配置寄存器

✧ PINFLAGSAB，FIFO FLAGA 和 FLAGB 配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
FLAGB3	**FLAGB2**	**FLAGB1**	**FLAGB0**	**FLAGA3**	**FLAGA2**	**FLAGA1**	**FLAGA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ PINFLAGSCD，FIFO FLAGC 和 FLAGD 配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
FLAGD3	**FLAGD2**	**FLAGD1**	**FLAGD0**	**FLAGC3**	**FLAGC2**	**FLAGC1**	**FLAGC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	0	0	0	0	0

FLAGA，FLAGB，FLAGC，FLAGD 反映 FIFO 状态选择。每个脚有编址/固定两种模 式：如设为编址模式，则它们都反映 FIFOADR[1:0]脚当前所指端点的状态，其中，FLAGA 反映“可编程极限”，FLAGB 反映“满”标志，FLAGC 反映“空”标志，FLAGD 不存在； 如设为固定模式，它们均可任意设置成反映任意端点的任意标志，而不受限于 FIFOADR[1:0] 脚当前所指端点的状态。

Slave fifo 模式中，用引脚 FLAGA~FLAGD 来定义用端点 FIFO 的状态，并可灵活编程来实现 FLAGx 设置，见表 2.2

FLAGx3	FLAGx2	FLAGx1	FLAGx0	引脚功能
0	0	0	0	FLAGA=PF, FLAGB=FF, FLAGC=EF,FLAGD=EP2PF (除FLAGD之外，FLAGA ~FLAGC对应的端点FIFO由引脚FIFOADR[0,1]来确定，详见表8.4)
0	0	0	1	保留
0	0	1	0	
0	0	1	1	
0	1	0	0	EP2PF
0	1	0	1	EP4PF
0	1	1	0	EP6PF
0	1	1	1	EP8PF
1	0	0	0	EP2EF
1	0	0	1	EP4EF
1	0	1	0	EP6EF
1	0	1	1	EP8EF
1	1	0	0	EP2FF
1	1	0	1	EP4FF
1	1	1	0	EP6FF
1	1	1	1	EP8FF

说明：

1. PF 表示 FIFO 编程状态，EF 表示 FIFO 已空，FF 表示 FIFO 已满
2. 0000 为索引模式，其它为固定模式

表 3.3 端点选择

FIFOADR1 pin	FIFOADR0 pin	选择端点
0	0	EP2
0	1	EP4
1	0	EP6
1	1	EP8

2．3 FIFORESET:端点缓冲区复位寄存器

b7	b6	b5	b4	b3	b2	b1	b0
NAKALL	**0**	**0**	**0**	**EP3**	**EP2**	**EP1**	**EP0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

将 FIFO 复位到初始状态。具体过程是，写 0x80 到此寄存器，NAK 所有主机请求；写 0x02，0x04，0x06，0x08 分别复位各个端点；写 0x00，结束复位过程。

一般，在每一次开始进行 slave FIFO 或 GPIF 传输之前，先复位端点，再清空端点，然后即可进行数据传输。

NAKALL－0 关闭 NAK 功能， 1 用 NAK 响应主控器请求，例如在复位端点 FIFO 时， 为了保证复位正常，防止主控器请求的干扰，先写入 0x80，然后复位端点，最后写入 0x00， 使能请求响应。

EP3~EP0,1 复位对应的端点缓冲区，其中 EP3~EP0 分别对应端点 EP8，EP6，EP4，EP2。

2．4 FIFOPINPOLAR：控制引脚极性设置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**PKTEND**	**SLOE**	**SLRD**	**SLWR**	**EF**	**FF**
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Slave FIFO 引脚极性设置:0 低有效，1 高有效。 提示：PF 极性没有提供寄存器设置，为高有效。

2．5 EPxCFG：端点 2，4，6，8 配置

✧ EP2CFG，端点 2 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	**DIR**	**TYPE1**	**TYPE0**	**SIZE**	**0**	**BUF1**	**BUF0**
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
1	0	1	0	0	0	1	0

✧ EP4CFG，端点 4 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	**DIR**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R/W	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

✧ EP6CFG，端点 6 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	**0**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

✧ EP8CFG，端点 8 配置

7	b6	b5	b4	b3	b2	b1	b0
VALID	**DIR**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R/W	R/W	R/W	R	R	R	R
1	1	1	0	0	0	0	0

VALID－0 端点无效，1 端点有效
DIR－端点方向， 0 =OUT 方向，1=IN 方向，默认端点 2，4 为 IN，端点 6,8 为 OUT
TYPE1，TYPE0－端点类型，见表 3.4

表 3.4 端点 2,4,6,8 类型

TYPE1	TYPE0	类型
0	0	无效
0	1	等时
1	0	批量（默认）
1	1	中断

SIZE－缓冲区大小（仅端点 2 和端点）， 0 =512 字节， 1 =1024 字节
BUF1，BUF0－端点缓冲区个数（仅端点 2 和端点 6），见表 3 .5

表 3.5 端点 2 和 6 缓冲区个数

BUF1	BUF0	类型
0	0	四缓冲
0	1	无效
1	0	双缓冲（默认）
1	1	三缓冲

2．6 EPxFIFOCFG：端点 FIFO 配置寄存器

✧ EP2FIFOCFG，EP4FIFOCFG，EP6FIFOCFG，EP8FIFOCFG

b7	b6	b5	b4	b3	b2	b1	b0
0	**INFM1**	**OEP1**	**AUTOOUT**	**AUTOIN**	**ZEROLENIN**	**0**	**WORDWIDE**
R	R/W	R/W	R/W	R/W	R/W	R	R/W
0	0	0	0	0	1	0	1

INFM1：FIFO 状态标志是否提前一个字节有效选择，IN 端点满减 1， 1 使能。

OEP1：FIFO 状态标志是否提前一个字节有效选择，OUT 端点空加 1， 1 使能。

AUTOOUT：在前面，我们说 Slave FIFO 方式下的数据传输过程不需要 FX2LP 固件的参与，实际上是不确切的，应该说，FX2LP 固件可以不参与数据传输过程，也可以参与。AUTOOUT 即可设置。如果设置 AUTOOUT 为 1，则就如上面所说的，FX2LP 固件只需要完成初始化工 作，真正的数据传输是不需要 FX2LP 固件的参与的，具体的说，当 FX2LP 从主机收到一包数据 时，外部逻辑即可看到 FIFO 端点缓冲区状态的改变，然后从中取数。如果设置 AUTOOUT 为 0，则数据传输过程就需要 FX2LP 参与了，此时当 FX2LP 从主机收到一包数据时，FIFO 端点缓冲区状态的改变并不会立刻在端口显现，而是固件先看到 FIFO 端点状态的改变，此时，FX2LP 固件可以做三件事情：

a．向 OUTPKTEND 中的 SKIP 位写 0，使 FIFO 端点状态的改变在端口显现，从而使 外部逻辑可以从 FIFO 端点中读取数据；

b．向 OUTPKTEND 中的 SKIP 位写 1，丢掉这包数据，这样就相当于主机从来就没有发送这一包数据，外部逻辑当然也不能从 FIFO 端点中读到这一包数据了；

c．从新编辑这一包数据，设置完全重写整个包的数据，再写 EPxBC 寄存器，把数据 传给外部逻辑。

在 FX2LP 复位之后，如果其 OUT 端点缓冲区内有一包数据未处理，这包数据并不会自动传给外部逻辑。所以，为保证 OUT 端点缓冲区内没有未处理数据，在 reset FX2LP 后，要清 空一下 OUT 端点缓冲区,具体做法就是向 SKIP 位写 1（OUT 端点缓冲区有几个缓冲区就写 几次）。

AUTOIN：Auto IN 和 Auto OUT 有一点不同，在 Auto OUT 里，包的大小只能是 512 或 1024，而在 Auto IN 里，包的大小可以任意设定，甚至可以是 0 字节，这可以通过 EPxAUTOINLENTH/L 设置。和 AUTOOUT 类似，当设置 AUTOIN ＝ 0 时，FX2LP 固件可以传输，丢弃，修改外部 逻辑传过来的数据，这通过向 INPTKEND 寄存器的 SKIP 写不同的值实现。

ZEROLENIN：是否允许传输 0 字节,1 使能， 0 非使能。

WORDWIDE：8 Bit，16 Bit 选择。当选择 8 Bit 模式时，Port B 将是 FD[7：0]；当选 择 16 Bit 模式时，Port D 将是 FD[15：8],1 则为 1 6 位， 0 则为 8 位。

2．7 EPxAUTOINLENH/L：端点 2 ,4,6,8AUTOIN 长度设 置（仅 IN 端点有效）

EP2AUTOINLENH,EP4AUTOINLENH,EP6AUTOINLENH,EP8AUTOINLENH，包长度高字节

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	PL10	PL9	PL8
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	1	0

EP2AUTOINLENL,EP4AUTOINLENL,EP6AUTOINLENL,EP8AUTOINLENL，包长度低字节

b7	b6	b5	b4	b3	b2	b1	b0
PL7	PL6	PL5	PL4	PL3	PL2	PL1	PL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

设置 AUTOIN 时自动传输的包大小（注意，不能大于 IN 端点的缓冲区的大小）。
说明：PL10 仅端点 2 和 6 有效

2．8 EPxFIFOPFH/L：FIFO 可编程 PF 状态长度设置

EP2FIFOPFH（非 ISO 端点）

b7	b6	b5	b4	b3	b2	b1	b0
DECIS	PKTSTAT	OUT:PFC12	OUT:PFC11	OUT:PFC10	0	PFC9	IN: PKTS[2] OUT:PFC8
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
1	0	0	0	1	0	0	0

EP4FIFOPFH（非 ISO 端点）

b7	b6	b5	b4	b3	b2	b1	b0
DECIS	PKTSTAT	0	OUT:PFC10	OUT:PFC9	0	0	PFC8
R/W	R/W	R	R/W	R/W	R	R	R/W
1	0	0	0	1	0	0	0

EP6FIFOPFH（非 ISO 端点）

b7	b6	b5	b4	b3	b2	b1	b0
DECIS	PKTSTAT	OUT:PFC12	OUT:PFC11	OUT:PFC10	0	PFC9	IN: PKTS[2] OUT:PFC8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	0	0	0

EP8FIFOPFH

b7	b6	b5	b4	b3	b2	b1	b0
DECIS	PKTSTAT	0	OUT:PFC10	OUT:PFC9	0	0	PFC8
R/W	R/W	R	R/W	R/W	R	R	R/W
0	0	0	0	1	0	0	0

DECIS－0 小于等于门限值 PF 有效， 1 大于等于门限值 PF 有效

PKSTAT－

1. OUT 端点 FIFO：门限值为 PFC12:0 设置，当 FIFO 长度小于等于门限值(DECIS=0),或者 FIFO 长度大于等于门限值(DECIS=1),则 PF 有效。
2. IN 端点 FIFO，且 PKTSTAT=1:门限值为 PFC9:0
3. IN 端点 FIFO，且 PKTSTAT=0:门限值由两部分组成：PKTS2:0(数据包)再加上 PFC9:0（当前数据长度）。

EP2FIFOPFL, EP4FIFOPFL, EP8FIFOPFL

b7	b6	b5	b4	b3	b2	b1	b0
PFC7	PFC6	PFC5	PFC4	PFC3	PFC2	PFC1	PFC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

解释：

对于 OUT 包，极限存储在 PFC[12:0]中，在整个 FIFO 缓冲区中的数据数目少于等于（DECIS＝0）或大于等于（DECIS＝1）这个极限时，PF 将有效。

对于 IN 包，当 PKTSTAT＝1 时，极限存储在两部分：PKTS[2：0]存储极限包数（已 经交给 SIE 但未传给主机的包数），PFC[9：0]存储极限字节数（正在编辑的包里的字节数）。 在整个 FIFO 缓冲区中的数据数目少于等于（DECIS＝0）或大于等于（DECIS＝1）这个极 限时，PF 将有效。

2．9 INPKTEND：结束 IN 传输

b7	b6	b5	b4	b3	b2	b1	b0
SKIP	0	0	0	EP3	EP2	EP1	EP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SKIP－当 ENH_PKT(REVCTL 寄存器 bit0)为 1 时，0 表示自动“分配”一个 IN 缓冲 区，1 表示将跳过一个 IN 缓冲区

EP3,EP2,EP1,EP0－代替 PKTEND 引脚功能，软件强行结束 IN 端点 8,6,4,2 IN 数据 传输，传输短包。

2．10 OUTPKTEND：强行结束 OUT 传输寄存器

b7	b6	b5	b4	b3	b2	b1	b0
SKIP	0	0	0	EP3	EP2	EP1	EP0
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

SKIP－当 ENH_PKT(REVCTL 寄存器 bit0)为 1 时，0 表示自动“分配”一个 OUT 缓 冲区，1

表示将跳过一个 OUT 缓冲区

EP3,EP2,EP1,EP0－代替 EPxBLH.7=1 引脚功能，软件强行结束 OUT 端点 8,6,4,2 数据传输。

2．11 EPxFIFOIE 和 EPxFIFOIRQ：端点 FIFO 中断（INT4）

使能和请求

EP2FIFOIE, EP4FIFOIE, EP6FIFOIE, EP8FIFOIE

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**EDGEPF**	**PF**	**EF**	**FF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EDGPF－PF 中断触发沿，0 上升沿触发， 1 下降沿触发
PF－ 1 使能端点 FIFO PF 中断， 0 非使能
EF－ 1 使能端点 FIFO EF 中断， 0 非使能
FF－ 1 使能端点 FIFO FF 中断， 0 非使能

EP2FIFOIRQ，EP4FIFOIRQ，EP6FIFOIRQ，EP8FIFOIRQ

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**PF**	**EF**	**FF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PF－0 无 PF 中断， 1 有 PF 中断
EF－0 无 EF 中断， 1 有 EF 中断
FF－0 无 FF 中断， 1 有 FF 中断

2．12PORTACFG：端口 A 配置

b7	b6	b5	b4	b3	b2	b1	b0
FLAGD	**SLCS**	**0**	**0**	**0**	**0**	**INT1**	**INT0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

置 1 使能端口 A 复用引脚，虽然 SLCS 出现在 PORTACFG.6 的位置上，当 IFCFG1:0=11 时，PORTA.7 复用为 SLCS，FLAGD 也出现在 PORTA.7 引脚上，当 PORTACFG.7 置位 时，PORTA.7 复用为 FLAGD 输出，当 PORTACFG.6 和 PORTACFG.7 均为 1，则 PORTA.7 复用为 FLAGD。所以 PORTACFG7:6=01 时，PORTA.7 复用为 SLCS。

2．13 EPxFIFOBCH EPxFIFOBCL：端点 FIFO 计数

EP2FIFOBCH, EP4FIFOBCH, EP6FIFOBCH, EP8FIFOBCH 计数高字节

b7	B6	b5	b4	b3	b2	b1	b0
0	0	0	BC12	BC11	BC10	BC9	BC8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

EP2FIFOBCL, EP4FIFOBCL, EP6FIFOBCL, EP8FIFOBCL计数低字节

b7	B6	b5	b4	b3	b2	b1	b0
BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

当前端点缓冲区中已有的数据数目。
说明：
端点 2 最大缓冲区计数 BC[12:0]，为 4096 字节。 端点 6 最大缓冲区计数 BC[11:0]，为 2048 字节。端点 4 和 8 最大缓冲区计数 BC[10:0]，为 1024 字节。

2．14 EP24\68 FIFOFLAG（SFR AB：SFR AC）和 EPxFIFOFLGS：端点 FIFO 状态标志寄存器

EP24FIFOFLGS

b7	b6	b5	b4	b3	b2	b1	b0
0	EP4PF	EP4EF	EP4FF	0	EP2PF	EP2EF	EP2FF
R	R	R	R	R	R	R	R
0	0	1	0	0	0	1	0

EP68FIFOFLGS

b7	b6	b5	b4	b3	b2	b1	b0
0	EP8PF	EP8EF	EP8FF	0	EP6PF	EP6EF	EP6FF
R	R	R	R	R	R	R	R
0	1	1	0	0	1	1	0

2．15 其它通用寄存器

CPUCS：
PORTCSTB：128 脚或 100 脚的 RD，WR 输出使能。

CLKSPD1，CLKSPD0：CPU 频率选择，00：12MHz（默认）；01：24MHz；10：48MHz；11：Reserved。
CLKINV：CLKOUT 反转选择。
CLKOE：CLKOUT 输出使能。
REVCTL（E608）：

正常情况下，简单地设置 DYN_OUT 和 ENH_PKT 位为 1 即可。