数字 IC 设计知识

作者：HateHanzo

联系方式：HateHanzo@163.com

修改记录

版本号	修改文件	描述	作者	时间
v1.0		初稿	HateHanzo	20201023
		增加了寄存器写“1”清 0 控制逻辑	HateHanzo	20210227

缩略语与约定符号

缩略语

英文简写	英文全称	中文描述
AHB	Advanced High-performance Bus	先进高性能总线
SRAM	Static Random-Access Memory	静态随机存取存储器
FSM	Finite State Machine	有限状态机

约定符号

1、对于某信号名 A，若无特殊声明，A_d1 表示该信号延时一拍，A_d2 表示延时两拍，以此类推。如 ahb_addr_d1 表示信号 ahb_addr 延时一拍。

目　　录

1 数字 IC 设计基础知识

1.1 setup time 和 hold time

以一道练习题为例来讲解。

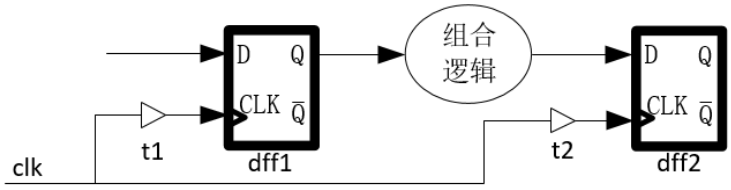

图 1 setup time 和 hold time 练习题图

1、两个 DFF 要求的 setup 和 hold 时间为 Ts 和 Th，DFF 的 cell delay 为 Td
2、中间组合逻辑的延时为 Tc
3、时钟的 uncertainty 为 Tu
4、clk 到 DFF1 和 DFF2 的延时分别为 t1 和 t2

问题：1、为使电路能正常工作，对中间组合逻辑电路的延时 Tc 有什么要求？
2、这个电路最高可以工作在什么频率下？写出推导计算公式。

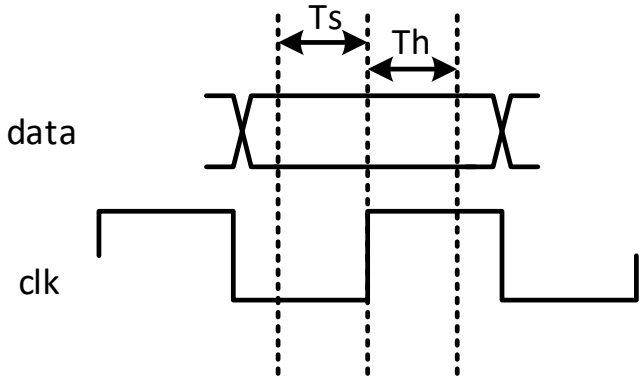

图 2 setup time 和 hold time 解释图

图 1 是建立时间和保持时间的示意图，可以看到触发器对时序的要求有一个时间窗口，如果真实的采样时钟边沿超前，则不利于建立时间，但有利于保持时间；如果滞后，不利于保持时间，但有利于建立时间。对建立时间的分析应该看 dff2。时序约束中有个概念叫时钟不确定性(uncertainty)，这是为了给与时钟更加严格的约束，换句话说，Tu 的存在应该是考虑让时序变差的情况。

建立时间余量：

$$Tclk + t2 - (t1 + Td + Tc) - Tu > Ts$$

保持时间余量：

$$t1 + Td + Tc - Tu > Th$$

1.2 同步时钟与异步时钟

所谓的同步时钟也就是 EDA 工具可以对其进行时序分析从而确保建立时间和保持时间得到满足，而对于异步时钟 EDA 工具无法对其进行时序分析，所以对于跨时钟域的单 bit 或多 bit 信号都有专门的电路设计去保证，在给约束的时候会将其设置为 false_path。

即使两个时钟的频率相同，也不一定是同步时钟，因为有可能来自不同的时钟源。

两个有分频关系的同源时钟(比如在设计中一个为 40MHz，一个为 20MHz)，作为同步时钟域处理，那么下 SDA 由后端保证时序(调相位、设 multicycle 等)；如果将其作为异步时钟处理，那么需要在设计上进行保证，给约束时将其设置为 false_path，不做 STA 分析。

2 数字 IC 设计基础电路

2.1 2 级同步电路

clka 时钟域下的信号传递到 clkb 时钟域下会产生亚稳态问题，亚稳态产生的原因就在于 setup time 和 hold time 不满足，在同一个时钟域下，EDA 工具可以在数据路径或时钟路径插入 buffer 从而使其满足时序要求，但对于跨时钟域信号 EDA 工具无法进行分析，也就是在静态时序分析里说的 false path[1]。对于跨时钟域信号传输，2 级同步器是基本要求，如果是关键信号或者高速设计可能会采用 3 级同步器。

使用两级同步器对信号进行同步，如果是由慢时钟域向快时钟域同步，那么直接使用 2 级同步器进行同步即可，但要特别注意的是此时用于同步的信号必须是寄存器输出信号而不能是组合逻辑输出。

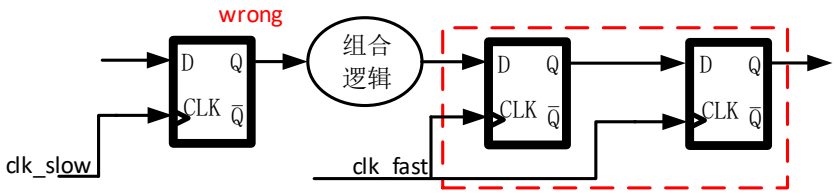

图 3 错误的跨时钟域同步电路

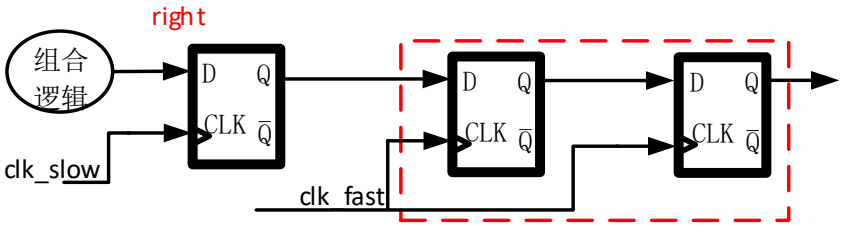

图 4 正确的跨时钟域同步电路

2.2 单比特脉冲信号由快到慢同步电路

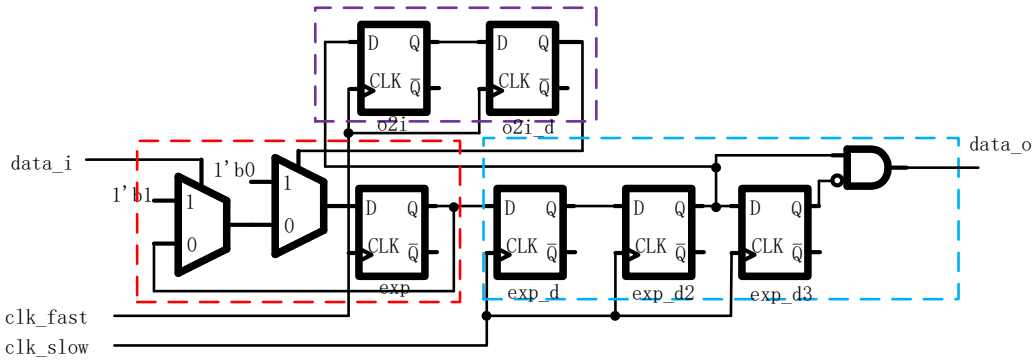

图 5 单比特脉冲信号由快到慢同步电路

单比特信号由快时钟域向慢时钟域传递，首先需要在快时钟域下将脉冲信号展平为电平信号(红框部分)，此处电平信号的意思是在慢时钟域来看，这个信号是很宽的一个信号，在慢时钟域下一定能抓到。在慢时钟域下将电平信号抓 2 拍后的信号在慢时钟下就是可以使用的了，此处将其和打一拍后的信号做组合逻辑产生慢时钟域的脉冲信号(蓝框部分)，与此同时同步到慢时钟域下的信号需要反馈回快时钟域，所以再次使用 2 级同步电路(紫框部分)将该信号反馈回慢时钟域将电平信号拉低，从而完成慢时钟域信号与快时钟域信号的握手。

2.3 同步 FIFO

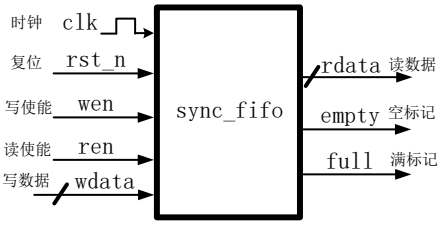

图 6 同步 FIFO 接口框图

1、当 wen && (!full) 为同步 fifo 的一次有效写，当 ren && (!empty)为同步 fifo 的一次有效读

2、若同步 fifo 有写指针 waddr 和读指针 raddr，其位宽为 N，但实际用于寻址的读写指针位宽为 N-1。一次有效的写到来后，将 wdata 写入 fifo 的存储，同时 waddr 自加 1，其余情况 fifo 存储和 waddr 保持；一次有效的读到来后，将 fifo 存储里的数据打到 rdata，同时 raddr 自加 1，其余情况 rdata 和 raddr 保持。

3、当读地址的最高位(MSB)和写地址的最高位(MSB)相异且其余位相同时，满信号 full 产生，当读地址和写地址完全相同时，空信号 empty 产生。

2.4 异步 FIFO

单 bit 数据进行同步的方法(打两拍、握手)并不适用于多 bit 数据跨时钟域传输，主要原因在于多 bit 数据输出时，每个触发器时钟延时可能不一致，信号传输的延时也可能不一致，异步时钟之间相位信息不确定，时序分析工具无法得到准确的建立保持时序，无法保证数据都在同一个周期内采样，从而导致设计结果错误。对于多 bit 数据的跨时钟域传输，通常采用异步 FIFO 进行数据同步。

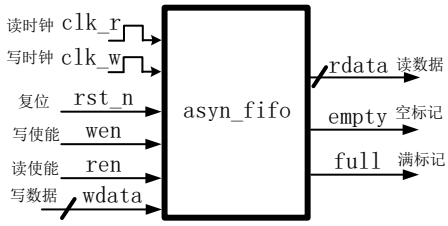

图 7 异步 FIFO 接口框图

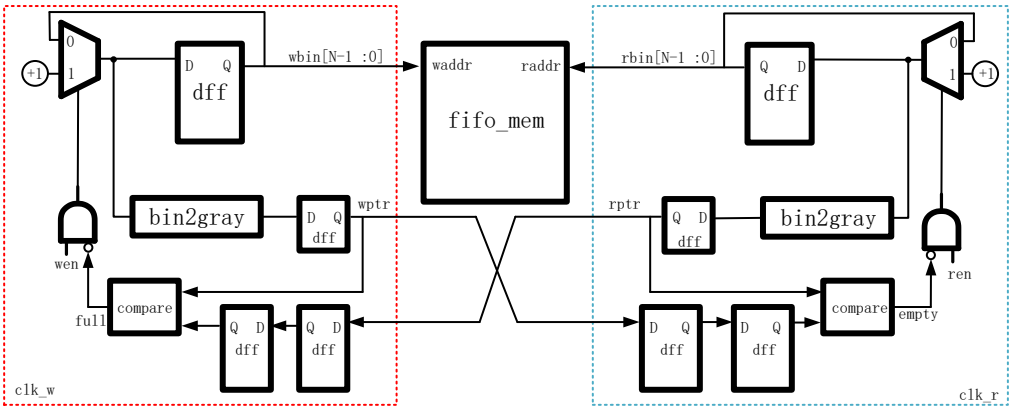

图 8 异步 FIFO 电路结构图

1、和同步 FIFO 一样，异步 FIFO 也是当 wen && (!full)时为一次有效的写，当 ren && (!empty)时为的一次有效的读。

2、异步 FIFO 写指针 wbin 和读指针 rbin，其位宽为 N，但实际用于寻址的读写指针位宽为 N-1。一次有效的写到来后，将 wdata 写入 fifo 的存储，同时 wbin 自加 1，其余情况 fifo 存储和 wbin 保持；一次有效的读到来后，将 fifo 存储里的数据打到 rdata，同时 rbin 自加 1，其余情况 rdata 和 rbin 保持。

3、与同步 FIFO 不同的是，异步 FIFO 将读写指针的格雷码进行比较从而产生空满信号，通过组合逻辑将二进制读写指针转换为格雷码指针，在本时钟域下寄存后被跨时钟域的两级同步器采样进行比较产生空满信号。在读时钟域，当寄存后的读格雷码指针 rptr 与同步过来的写格雷码指针 wptr_d2 完全相等时，产生 empty 信号；在写时钟域，当寄存后的写格雷码指针 wptr 与同步过来的读格雷码指针 rptr_d2 的最高位和次高位都不同，其余位相同时，产生 full 信号。

4、以上 3 点是所有的异步 fifo 的通用接口，事实上在此基础上还可以利用内的信号在产生一些别的辅助信号，譬如 fifo 已满，但电路仍然产生写信号，此时可以通过寄存器输出一个 overflow(“1”有效)接口信号表示异常产生，同理，当 fifo 为空，但电路就产生了读信号，此时可以通过寄存器输出一个 underflow(“1”有效)接口信号表示异常的产生。

可以添加 wclr 和 rclr 两个输入接口信号，wclr 是写时钟域的信号，rclr 是读时钟域的信号。在写时钟域，wclr 有效时将 wbin、wptr 和 overflow 清“0”；在读时钟域，rclr 有效时将 rbin、rptr 和 underflow 清“0”。

在写时钟域，将同步过来的 rptr_d2 转换为二进制码 rptr_bin，则 wleft = wbin –rptr_bin 表示的是 fifo 写入了多少个数。

2.5 去抖滤波电路

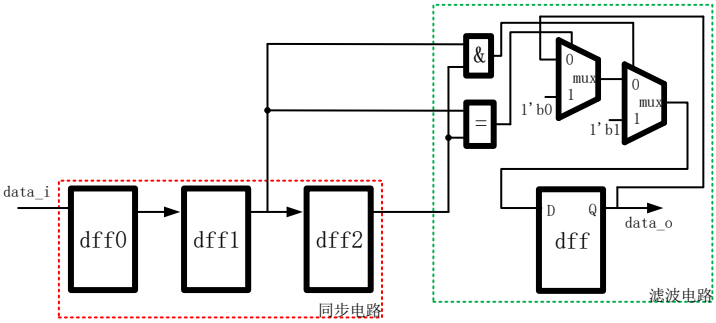

图 9 去抖滤波电路图

一种常用的去抖动滤波电路，通常用在芯片的 PAD 输入信号，或者模拟电路给数字电路的接口信号上，图 9 电路对应的 RTL 代码如下，电路分为两部分，同步电路和滤波电路。该电路可以滤除宽度小于 1 个 clk 的毛刺信号。可以依据实际芯片的时钟增减同步电路的级数与滤除宽度的大小。比如该滤波电路常用 I2C 控制器设计中，若时钟频率较快，如 4.8MHz，9.6MHz，通常会采用同步加滤波电路的组合，但假如时钟频率较慢，比如只有 128KHz，这时候毛刺信号通常是 ns 级的，而时钟频率是毫秒级别的，时钟是很难抓到毛刺信号的，所以这时候通常只需要将输入信号用两级同步电路处理即可。

同步电路：

```
always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
    data_deb <= 3’b111;(该复位值依据 data_i 的初始值而改变)
  else
    data_deb <= #1 {data_deb[1:0],data_i};
end
```

滤波电路：

```
always@(posedge clk or negedge rst_n)
begin
  if(!rst_n)
    data_o <= 1’b1;(该复位值依据 data_i 的初始值而改变)
  else if(data_deb[2:1]==2’b11)
    data_o <= #1 1’b1;
  else if(data_deb[2:1]==2’b00)
    data_o <= #1 1’b0;
  else ;
end
```

该电路设计思路也容易理解，首先将输入信号 data_i 同步 3 个 clk,在后两个 dff 都为“1”的时候，把 1’b1 用寄存器输出，在后两个 dff 都为“0”的时候，把 1’b0 用寄存器输出，这样就把 1 个 clk 脉宽以下的毛刺给滤除掉了。

2.6 寄存器写“1”清 0

在 SOC 的设计中，往往有这样的需求：对寄存器的某个 bit 写“1”，就会将该位清“0”

例子：如下所示，当 i2c_master_end 或 i2c_slave_iow 脉冲产生时，寄存器 regs_0x00 的第 29bit 会置“1”，利用软件对该 bit 写“1”可以将其清“0”。

```
wire regs_0x00 = {x,y,i2c_pend,29’d0}
always@(posedge hclk or negedge rst_n)
  if(!rst_n)
    i2c_pend <= #1 1’b0;
  else if(iow00 && ahb_wdata[29])
    i2c_pend <= #1 1’b0;
  else if(i2c_master_end || i2c_slave_iow)
    i2c_pend <== #1 1’b1;
```

参考文献

[1] http://bbs.eetop.cn/thread-605419-1-1.html

[2] http://bbs.eetop.cn/thread-605729-1-1.html

[3]

附 录