VLSI设计CQIU

2017年4月17日星期一

目 录

[1. 模板标题1 3](#_Toc1684283382)

[1.1 模板标题2 3](#_Toc1267391699)

[1.1.1 模板标题3 3](#_Toc397310112)

[1.1.1.1 模板标题4 3](#_Toc476051819)

## 模板标题1

见下文

### 模板标题2

见下文

#### 模板标题3

见下文

##### 模板标题4

见下文

###### 模板标题5

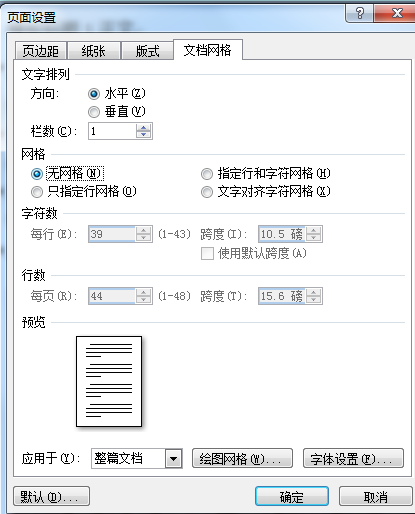
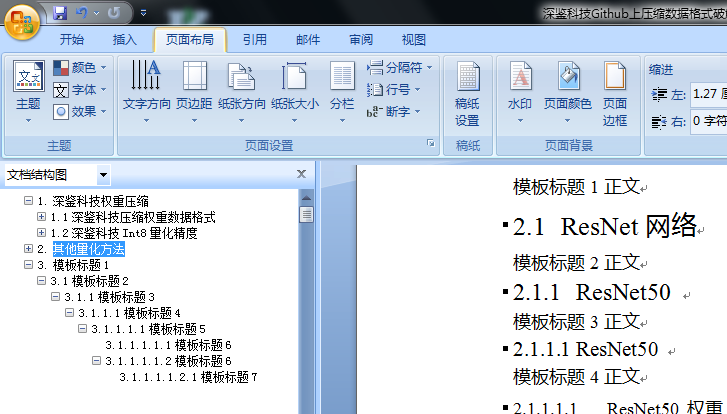
见下文

模板标题6

见下文

模板标题7

见下文



对于WPS来说，可以从”文件”-->”页面设置”-->”文档网络”，然后选择“无网络”

## 引言

什么是VLSI（Very Large Scale Integration，超大规模集成电路） ；

什么是FPGA（Field-Programmable Logic,Array）；

什么是ASIC（Application Specific Integrated Circuit）专用集成电路；

什么是CPLD（Complex Programmable Logic Devices）复杂的可编程逻辑元器件；

什么是SPLD（Simple Programmable Logic Devices）简单的可编程逻辑元器件；

什么是PDFP（Programmable Digital Signal Processor）通用的可编程数字信号处理器；

什么是IP（Intellectual Property）知识产权。

数字IC前端设计与FPGA开发，原则上属于两个职位，但是他们都属于RTL级别的开发，有共同点，也有区别。

区别在于：

第一：FPGA层次直接调用IP core能搞定的事情，但是对于数字IC设计来说，他们更喜欢刨根问底，了解细节，比如异步FIFO是如何做跨时钟域同步的等等，这些特殊之处，正是与FPGA开发显著的区别。

第二：有些数字IC设计公司，在验证阶段，不一定会用FPGA做原型验证，why？因为IC里面用到的标准库与FPGA能提供的可能不一致，毕竟一个是面向代工厂，一个是面向FPGA器件本身。正因为不一致，有时候为了原型验证而去改代码，反而得不偿失。

第三：数字IC设计公司目前开发主流语言还是Verilog、VHDL，还没用到HLS，why？功耗、面积成本的综合考虑，我宁愿开发时间稍微多花一点儿，我也不能用HLS输出的代码就去流片，这个风险太高了，基于这个原因，其实做IC设计的人，骨子里是抵触HLS设计的，心里有这种感觉。

基于这些考虑，本科，研究生微电子专业的我，工作后从事了7年FPGA开发，对自己之前的工作有了一些反思，希望能把自己之前学过的进行阶段整理，对VLSI的设计有更加精准的把握。

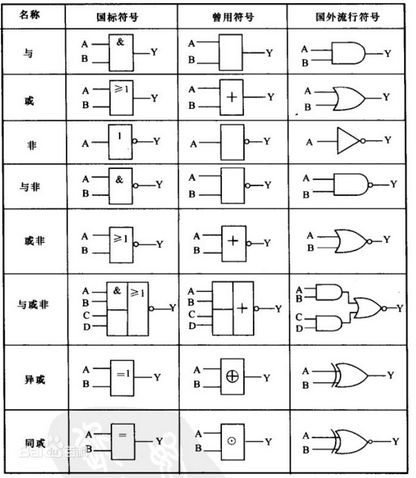
## 数字编码

数字编码中包含很多种编码方式，以及进制转换的方式。

### 布尔代数

#### 符号表格

布尔代数的符号表格



#### 基本定理

##### 代入定理

代入定理，在任意一个包含A的逻辑等式中，以另外一个逻辑式，替换A的位置，则等式依然成立。

##### 反演定理--最重要的定理

反演定理，对于任何逻辑式Y，将其所有的

与换成或，

或换成与，

0换1,

1换0,

原变量换成反变量，则得到的结果就是/Y.

原则：必须遵守先括号，然后乘，最后加的运算次序;

不属于单个变量上的反号，不变。

特别记忆：徳.摩根定理，就是反演定理的特例。

与非，等于非或



或非，等于非与



###### 例题，利用反演定理编程与非，或者与或形式。

比如将逻辑函数，



简成与非，与非的形式。

我们一般都按照最小项的方式写最简函数，如果给定的例题是，比如让你用与非门搭建电路，要如何将逻辑式进行替换，可以考虑用反演定理。

变形1：

变形2：利用反演定理



这样就变成了与非的形式。

##### 对偶定理

对偶定理：若两个逻辑式相等，则对偶式也相等。

其实是根据反演定理推的

比如Y=A(B+C)

我们知道它的反演定理，/Y = A+(B\*C)

那么就是对偶了。

##### 常用公式

###### 消因子法

类似形式：



这种A X一个系数1

加上A的非 X 系数2

等于再原来基础上+系数1X系数2,等式不变。

化简举例：



#### 逻辑函数表示方法

真值表，逻辑函数式（简称逻辑式，或者叫，函数式），逻辑图，卡诺图。

### 常见编码转换

#### 二-十进制之间转换

二进制-->十进制，比如(1011.01)2,这种，直接乘以对应想权重即可。



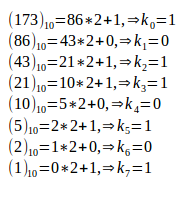
十进制-->二进制，这个是重点

利用如下公式



这个公式表明，若将 除以2，则得到的余数就是 ,商就是

举例说明：

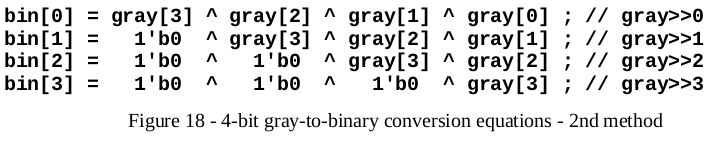


然后这些余数，从最后开始排列，就是10101101，这个数就是173的二进制表示方法。

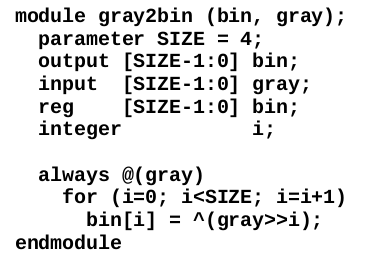
#### 二进制与Gray码之间转换

##### Gray码到二进制

参考论文《Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs》

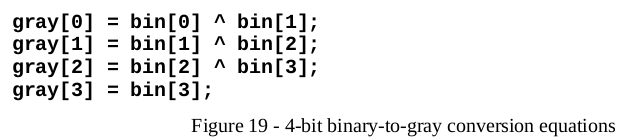


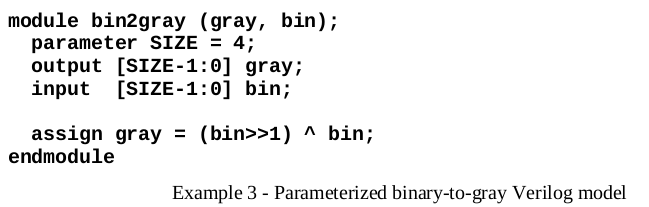
通用代码



##### 二进制到Gray码转换

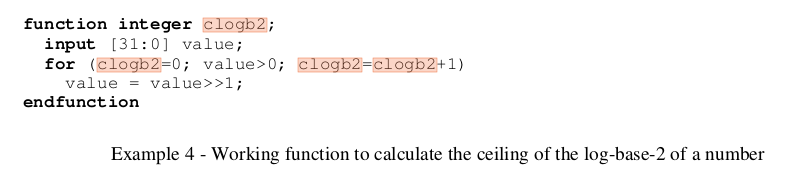
参考论文《Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs》





#### 给定数据求位宽 ceiling of log-base 2

Graph:



Code:

function integer clogb2;

input [31:0] value;

for (clogb2=0; value>0; clogb2=clogb2+1)

value = value>>1;

endfunction

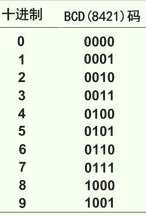
### 常见码制

#### BCD码

在四位二进制数码表示1位10进制数0-9时，有多种不同的表示方法，这些代码叫二-十进制码，简称BCD(Binary Coded Decimal)代码。

##### 编码8421码

有8421码，就是我们说的二进制表示



##### 余三码

余三码，这个码，比8421码都多3，该编码特点：

特点1：两个余三码相加，所得的和比原来10进制数多6,若两个数的和是10，则能自动产生进位信息。

特点2：0与9,

1与8,

2与7,

3与6,

4与5,他们之间互为反码，这对于求取10的补码来说很方便。

余3码，不是恒权码。

##### 编码2421码

它是恒权码

特点：0与9,

1与8,

2与7,

3与6,

4与5,他们之间互为反码，这对于求取10的补码来说很方便。

##### 编码5211码

它是恒权码，如果用8421码接成计数器后，4个触发器输出脉冲对于计数器脉冲的分频比依次为5：2：1：1，利用这种关系，某些情况下很有用。

##### 余三循环码

它是变权码，特点是相邻代码之间仅有1位状态不同。

## 门电路

CMOS与非门中，上面管子是PMOS管，下面是NMOS管，一般的，P并联，N串联的管子，是与非门。

否则是或非门。

### 基本门

这里的基本门指的是Verilog语言中的基本门电路。

#### 多输入门

##### 基本结构

这类门电路结构如图所示，多个输入，一个输出



##### 基本种类

与门 and

与非门 nand

或门 or

或非门 nor

异或门 xor

同或门 xnor

##### 语法格式

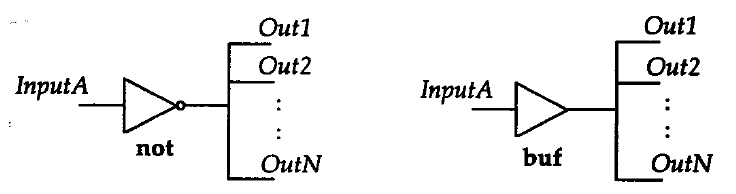
Verilog语法，类似这种，第一个端口是输出，其他均为输入



#### 多输出门

##### 基本结构

这类结构的门，单个输入，多个输出。



##### 基本种类

缓冲 buf

非门 not

##### 语法格式

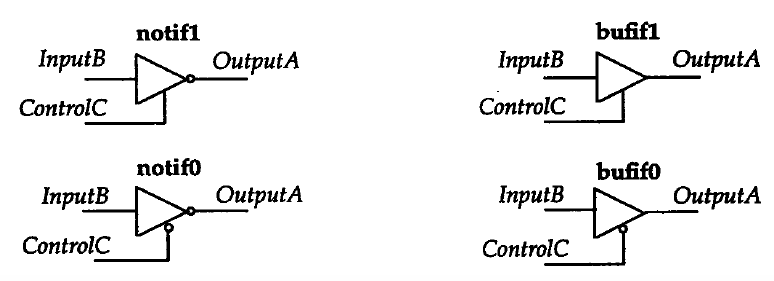
verilog使用基本形式：



#### 三态门

##### 基本结构

三态门结构：主要用来对三态驱动器建模，内部有一个输出端，一个数据输入端，和一个控制输入端。



##### 基本种类

bufif0

bufif1

notif0

notif1

##### 语法格式

基本语法结构：

这种形式;

比如bufif0,如果ControlC=1,则输出OutputA=Z，否则OutputA=InputB

比如bufif1,如果ControlC=0,则输出OutputA=Z，否则OutputA=InputB

比如notif0,如果ControlC=1,则输出OutputA=Z，否则OutputA=~InputB

比如notif1如果ControlC=0,则输出OutputA=Z，否则OutputA=~InputB

#### 上拉/下拉电阻

##### 基本结构

没有输入，只有固定输出，上拉电阻，输出置1,下拉输出置0

##### 基本种类

pullup

pulldown

##### 语法格式

基本语法格式

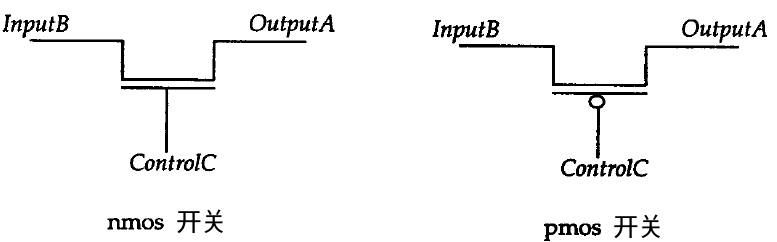


比如 pullup PUP(Pwr);将输出信号Pwr置为高电平。

### MOS开关

#### 单向开关建模

##### 基本结构



##### 基本种类

cmos

pmos

nmos

rcmos

rpmos

rnmos

其中，这里面的r代表电阻的意思,rnmos比nmos在输入引线和输出引线之间，存在了高阻抗，因此数据从输入传输到输出时，对于rpmos和rmos来说，存在数据信号强度衰减。

##### 基本语法



举例说明：



#### 双向开关建模

##### 基本种类

tran rtran tranif0 rtranif0 tranif1 rtranif1,这些开关是双向的，数据可以双向流动，当数据在开关中传播没有延迟时，后面4个开关可以设置合适的控制信号关闭。但是tran，rtran不能关闭.

### 时延建模

可以使用内部时延，定义从任何输入到输出信号的传输延迟。

语法结构

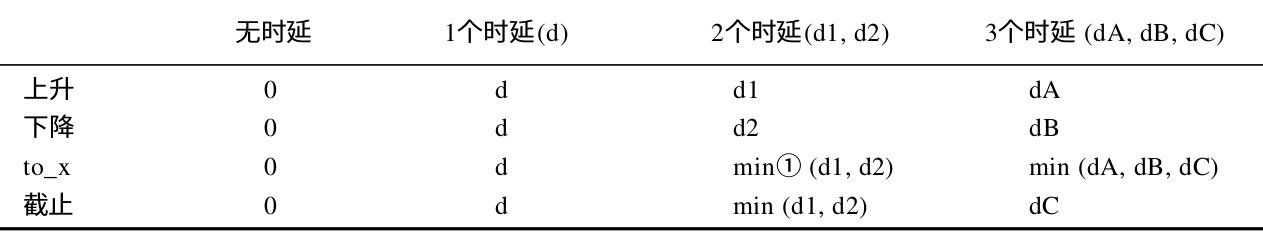


比如：



表明输出信号out经过6个延迟单位得到。

内部延迟可以定义3个



### 常见组合逻辑门电路

#### 编码器

编码器介绍

#### 译码器

译码器介绍

#### 数据选择器

##### 二选一数据选择器

从2选1数据选择器入手，介绍多选1数据选择器以及占用的资源情况介绍。

##### 多选一数据选择器

扩展到多选一的数据选择器情况。

#### 加法器

##### 一位加法器

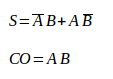
###### 一位半加器

定义

如果不考虑右来自低位的进位

直接将两个1位二进制数相加，这就叫半加器，实现半加运算的电路叫做半加器。

逻辑式（逻辑函数式）



资源

种类 个数

2输入异或门 1

2输入与门 1

###### 一位全加器

定义

如果考虑右来自低位的进位信号，那么全加器是靠略3个数相加得到和，还有进位信息。

逻辑式



一位全加器的逻辑电路分析，

分析S，当3个值时，就是3个值之间进行异或即可，和半加器的异或思路一致。

分析CO，当，A，B，以及CI之间，只要有2个是1,就有进位信号产生。

资源

种类 个数

1输入非门 3

2输入与门 3

3输入与门 4

3输入或非门 2

###### 拓展思考

可否能够通过硬砍掉逻辑函数式的方式，创造新的加法运算，实际上是一种错误的运算方法，但是电路结构异常简单，这样使得硬件加速效果更好。

##### 多位加法器

###### 串行进位加法器（行波进位加法器）

将低位全加器的输出CO，连接到高位的输入CI，组成串行进位加法器，这种加法器叫行波进位加法器。

###### 超前进位加法器

公式推导过程

加法器中，如果事先知道每个进位Ci，那么非常容易求出Si

因此有如下逻辑式Si，就是当前位置的Ai，Bi和Ci之间做异或即可。



下面的问题集中在求取Ci的问题上。

我们知道，对于一位全加器来说，有如下公式，



本地Ci经过运算后可得到下级Co，考虑到迭代关系，可认为Co是Ci+1，因此可以方便整理成如下形式，



我们定义AiBi叫进位生成函数Gi

定义Ai+Bi叫进位传送函数Pi，

经过变形有如下公式



其中



可以进行迭代展开。





如果展开到第0号位置，考虑，这个种形式，

可以看出，公式中只有一项带有C进位信息。

第一，如果展开到第0位，那么有这样的项





第二，然后还有这种项，



如果展开到Go,有如下形式



其中

类似这样的项，有i+1个，对于每个项，有从G0开始的，从G1开始的，到从Gi-1开始的。

因此可得到这样结果：

其中

第三，最后一项，固定的Gi

因此超前进位链的Ci+1为如下公式：



##### 加法器的资源评估

#### 乘法器

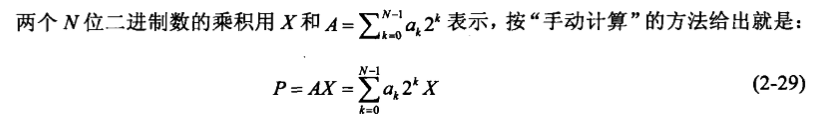
##### 常见乘法器结构

结构描述如下，乘法器的思路就是移位再相加，

###### 串串乘法器

对于输入位宽是NxN的乘法器，算法的时间复杂度是NxN，直接用与门和加法器来搭建即可。

###### 串并乘法器

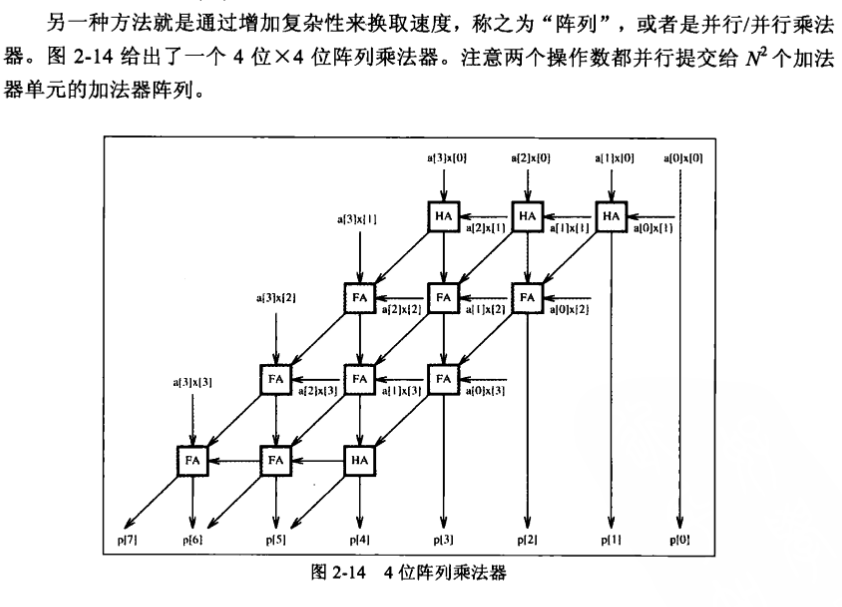


思路是，一个并行X，不变，一个A，把它按照位进行拆分。这个就是移位相加的方式。

时间复杂度是N，需要N个周期完成乘法。

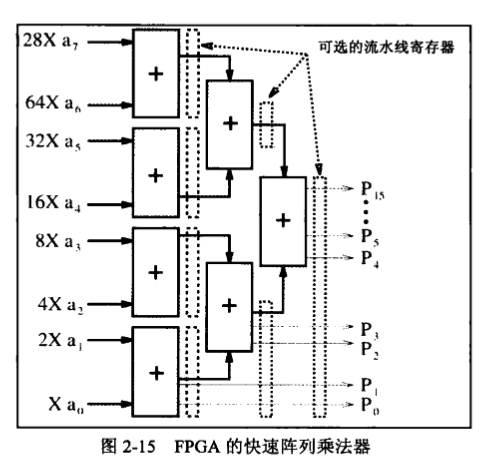
###### 并并乘法器

也叫矩阵乘法器，一拍搞定输出。



这个结构需要保证进位计算的时间和累加和的时间相同，才能最大效率发挥他的价值。FPGA不能满足这个条件，另辟他法。

###### 二叉树乘法器



两个两个组合的方式，流水线级数是log2(N)，这个就是之前介绍的那种流水线加法器的思路。

###### Booth乘法器（ASIC适用）

###### Wallace乘法器（ASIC适用）

用在FPGA领域不多。

###### 乘法器的资源评估方法

迭代公式， 对于位宽是2Nx2N的乘法器来说，假如我们采用二叉树乘法器的方案，需要做4个 NxN的乘法，以及3个 （2N）

2N x 2N的，需要4个NxN的乘法和 3次加法。

这三次加法中有两次3N+3N，加法，一次4N+4N加法

推导过程： Na Nb

X Nc Nd

-------------------------------

Na Nb

Na Nd

Nb Nc

Na Nc

---------------------------------

N3 N2 N1 N0

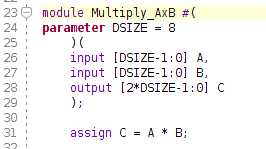
粗略评估：

1. N=1,时，两个1输入的乘法器，占用一个与非门
2. =>2x2 乘法器，需要 4x1=4个与非门，Xilinx实测需要2个Lut单元
3. =>4x4 乘法器，需要4×4=16个门电路，Xilinx实测 18个Lut单元
4. =>8x8 乘法器，需要4×16=64个门电路，Xilinx实测71个Lut单元
5. =>16x16乘法器，需要4x64=256个门电路，xilinx实测需要298个Lut
6. =>32x32乘法器，需要4x256=1024个门电路，xilinx实测需要1213个Lut
7. =>64x64乘法器，需要4x1024=4096个门电路，xilinx实测需要5048个Lut
8. =>128x128乘法器，需要4x4096=16384个门电路，xilinx实测需要x个Lut

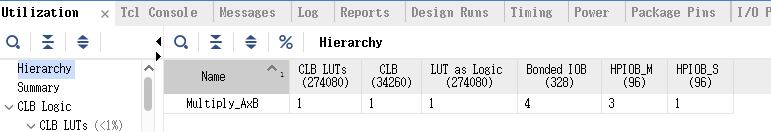
总结：NxN的乘法器，大概需要NxN个Lut资源。

Xilinx 实测结果

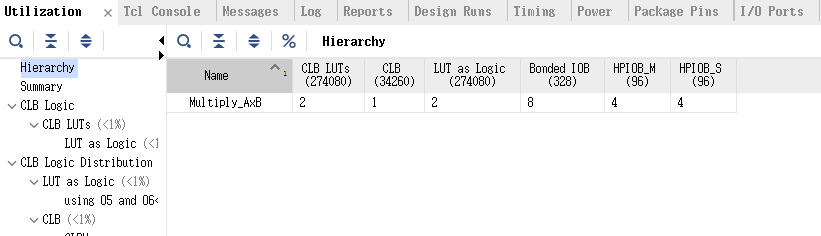
源码： 8位无符号数乘法器



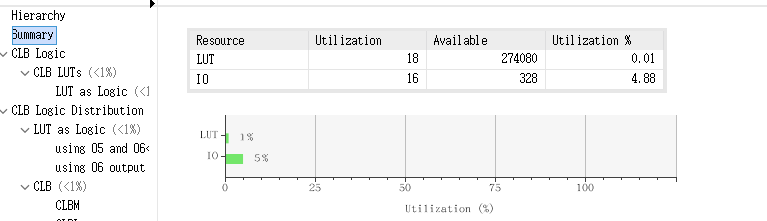
1）1x1



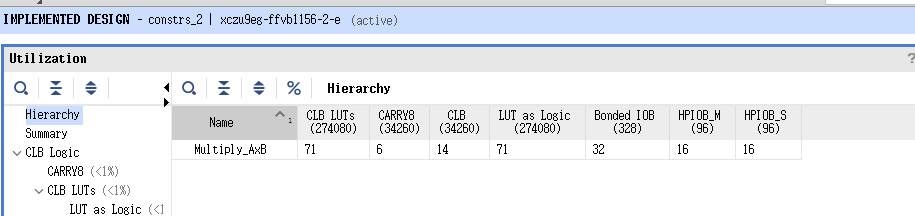
2) 2x2



3)4x4



4)8x8

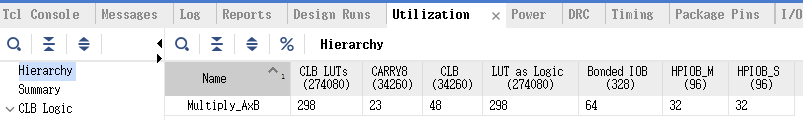


5)16x16

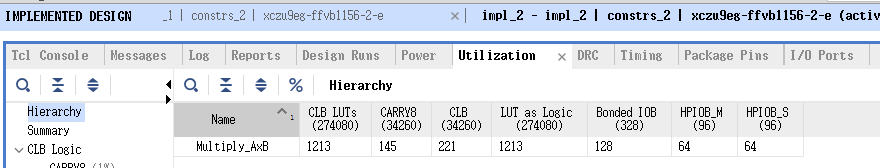
Vivado 工具默认情况下，位宽大于等于16时，采用dsp来进行电路映射，为了避免这种情况，我们将dsp电路映射功能关闭。



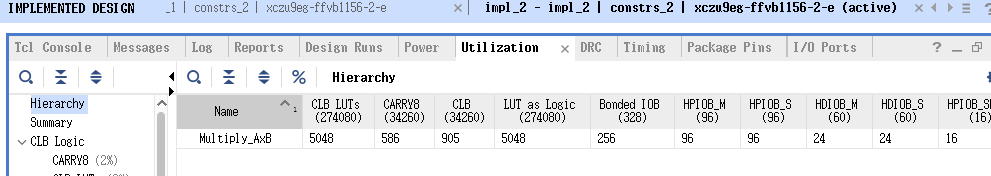
综合后资源分析：



6)32x32

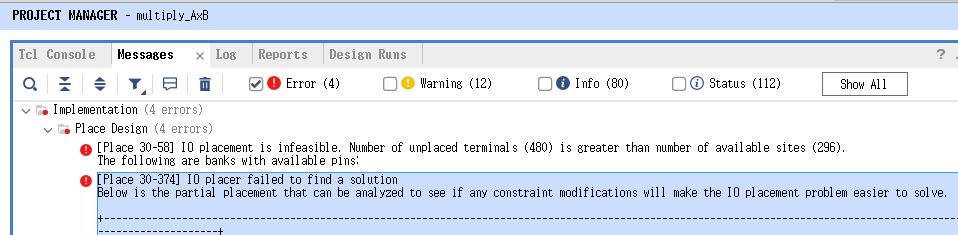


7)64x64



8)128x128

位宽太大的乘法器，无法布线，失败。



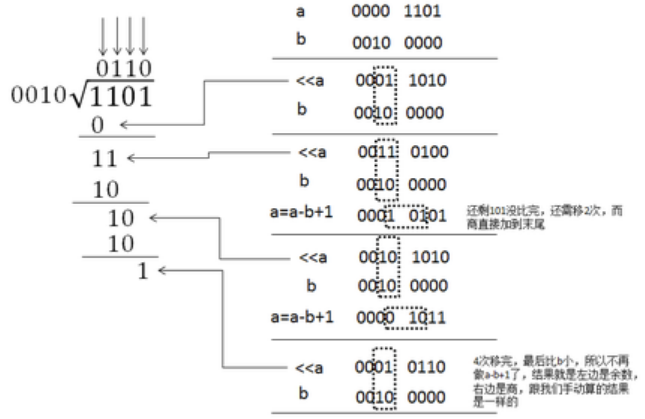
#### 除法器

##### 整数除法器设计利用移位相减，最快省资源的办法

算法推倒

假设4bit的两数相除 a/b，商和余数最多只有4位 （假设1101/0010也就是13除以2得6余1）

我们先自己做二进制除法，则首先看a的MSB，若比除数小则看前两位，大则减除数，然后看余数，以此类推直到最后看到LSB；而上述算法道理一样，a左移进前四位目的就在于从a本身的MSB开始看起，移4次则是看到LSB为止，期间若比除数大，则减去除数，注意减完以后正是此时所剩的余数。而商呢则加到了这个数的末尾，因为只要比除数大，商就是1，而商0则是直接左移了，因为会自动补0。这里比较巧因为商可以随此时的a继续左移，然后新的商会继续加到末尾。经过比对会发现移4位后左右两边分别就是余数和商。



具体思路总结：

当被除数N，除数都是N位时，另N=4

S1：被除数a--dividend , 高位补0

除数b-divisor , 先左移动4位

S2：判断a>b? 如果为真，则a’=a-b+1，否则a’=a

S3：调整新的a=a’ << 1左移动1位

执行到步骤S2,S3，反复执行N次，然后跳到S4步骤，做最后判断

S4：判断a>b?如果为真，则a=a-b+1，否则a’=a,注意都不必左移位了。

最后看a[2N-1:N]保存了余数

a[N-1:0]保存了商

对于N位除法，需要消耗的时钟周期=N+2次。

#### 数值比较器

数值比较器

#### 组合电路中的竞争冒险现象

##### 竞争产生

把门电路中两个输入信号同时向相反的逻辑电平跳变，的现象叫竞争。

##### 竞争冒险

由于竞争而在电路中输出端可能产生的尖峰脉冲现象叫 竞争-冒险。

##### 检查竞争-冒险的方法

只要输出端的逻辑在一定条件下可以转换成



只要存在这种情况，就可能存在竞争冒险。

##### 消除竞争-冒险的方法

方法1, 接入滤波电容

方法2, 引入选通脉冲

方法3, 增加冗余项。

### 常见时序逻辑门电路

#### D触发器，寄存器

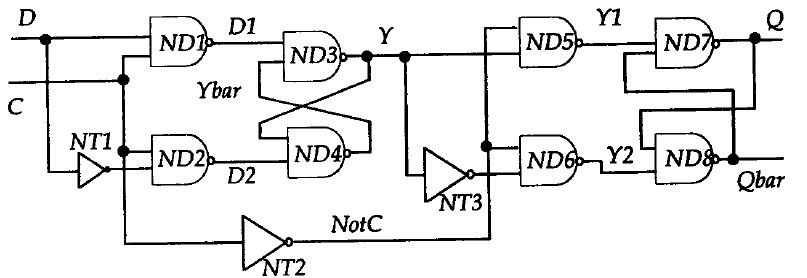
见下文

##### 主从触发器，D触发器

###### 电路结构

该电路结构，摘自verilog中文版，第47页

电路结构如图所示



就是2个基本RS触发器进行嵌套得到的。这个是D触发器的结构图，D触发器是组成数字电路中最重要的寄存器，也基本上，后来的DC综合工具，都是根据D触发器来实现的综合电路。

这个电路结构，能为后续时序分析中的建立时间和保持时间，奠定了电路基础。

有人问，为何时序电路中有建立时间和保持时间的概念，具体是怎么回事，学东西要学它的本质，刨根问底的精神必须有。

###### 占用资源

名称 个数

2输入与非门 8

1输入非门 3

#### 移位寄存器

见下文

#### 计数器

见下文

#### 顺序脉冲发生器

见下文

#### 序列信号发生器

见下文

## 常用IP

IC设计中常用一些优秀点典型的IP模块来实现基本的功能，这样能加快开发进度和效率，这就是IP复用的思想，这里主要以FPGA角度看常见的IP单元。

### 基本元件IP

见下文

#### 存储器Memory

##### RAM随机存储器

RAM，全称：Random Access Memory，随机存取存储器，所谓随机存取，是指，当存储器中的数据被读取或者写入的时候，所需要的时间与这段信息所在的位置或所写入的位置无关，其实他们是等延迟的。

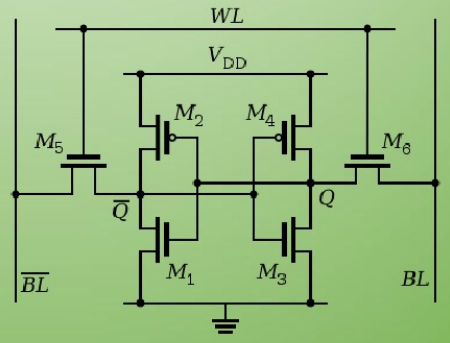
与之相反的存储器就是顺序存储器，典型的顺序存储器就是磁带。当然这里，主讲是RAM存储器。

###### SRAM，静态随机存储器

Mos管组成的6管单元存储器结构

这个图是6管单元组成的1Bit存储器，其中SRAM的值Q,/Q，是存储在了M1，M2，M3，M4这4个MOS管中，这4个管构成了一个交叉耦合反相器（注意反馈握手的思想方法）。另外两个M5，M6管，其实就是控制开关WL，这里称为字线(Word Line)，其实就是地址。

还有一个BL，/BL(Bit Line)，就是要传入的1 bit数据了。

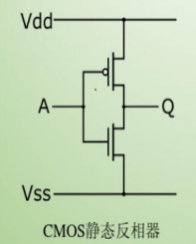


6管单元

思考，为何6管单元能存住数据呢？因为这是一个反馈的机制，我们知道，栅极电荷过一段时间就会跑掉的，如果把栅极和源或者漏电极接到一块，就有比较强的电荷。就变得稳定了。

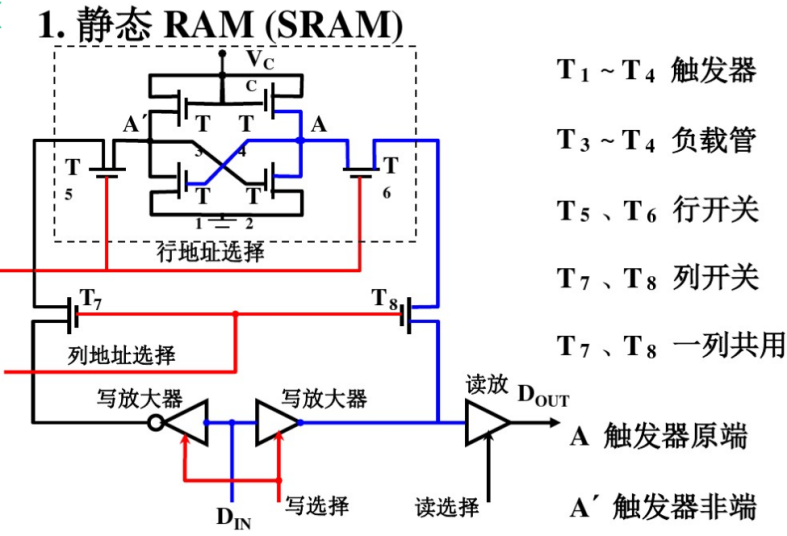
反相器如何做的

是一种电路器件，它的输出是输入的逻辑非，由两个互补的金属氧化物半导体场效应管（MOSFET）组成，源极连接在高电平，是P沟道场效应管，上管，连接在低电平的是N沟道场效应管。输入连接到场效应管的栅极上。输入为0时，P沟道开通，也就是上面开通，则输出就是1。



反相器

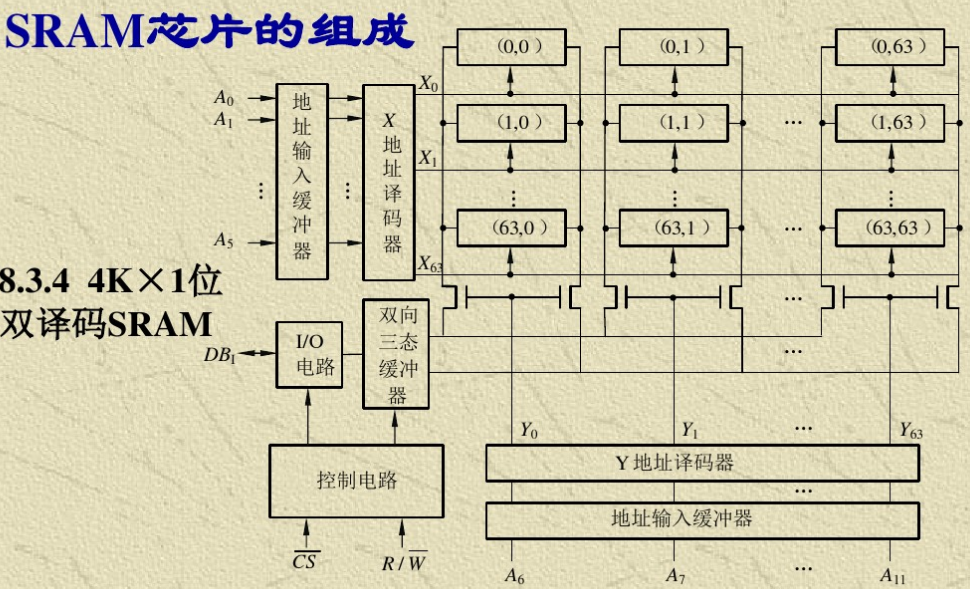
带有行列地址控制的单BIT SRAM



再增加2个管，做成了8管单元，就能控制行列选择了，输入Din接了一个放大器（应该就是反相器），放大器的输出驱动力强，可以改变之前电平的存储方式。而且不会对Din造成影响。

这里和之前的相比，多了地址选择器，然后多了写选择，和读选择。

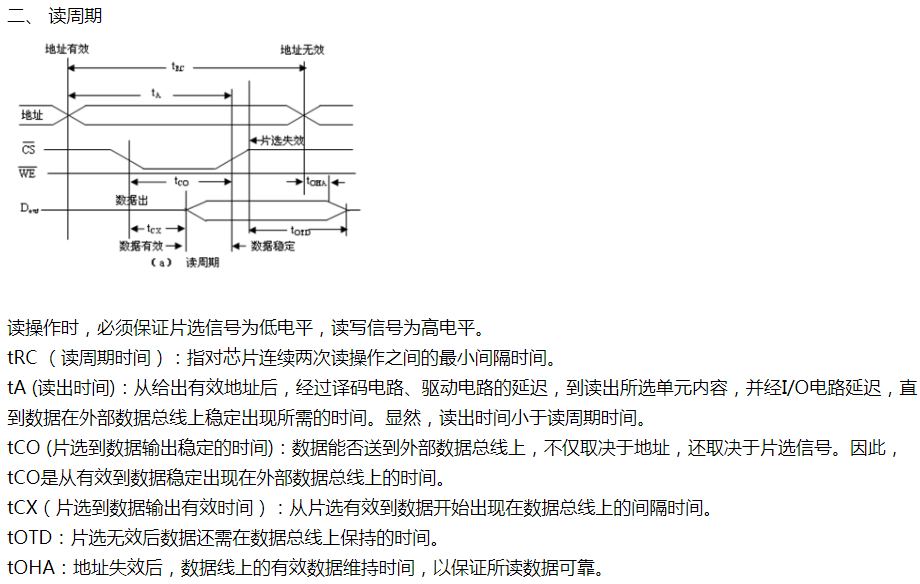
单Bit的4KSRAM

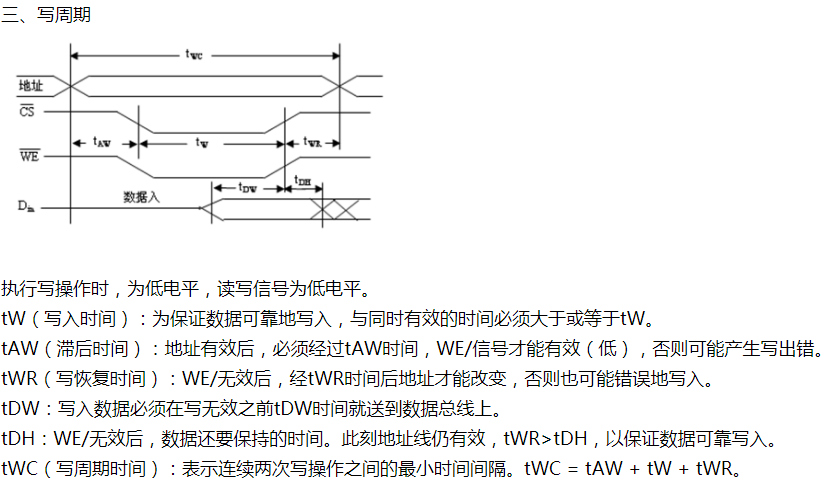


输入的地址从A0到A11，地址经过译码器后，选中唯一的一个Bit位，然后再对它进行读写即可。这就是单Bit RAM变成存储阵列后的样子，多Bit的话，也是一样的。

SRAM的读写时序

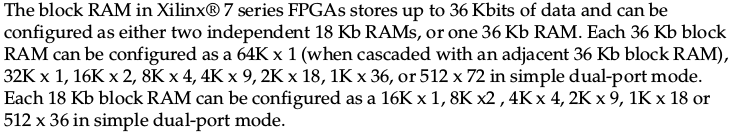
以Intel 2114为例子





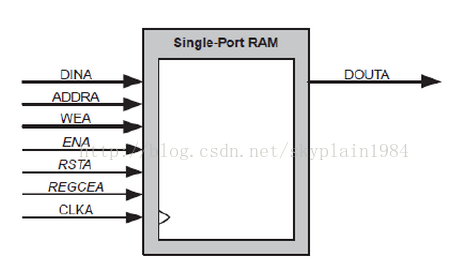
##### Xilinx的RAM存储器

参考文档：ug473\_7Series\_Memory\_Resources.pdf



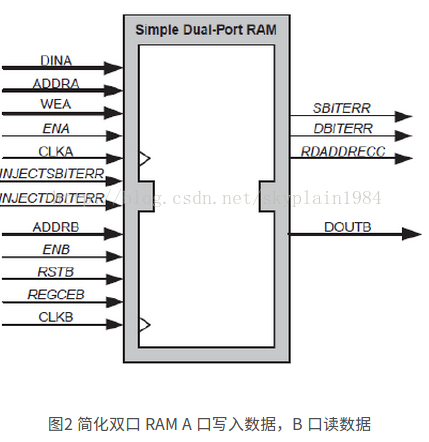
###### 单口RAM

一个口的地址，同一时刻，要么写，要么读，DDR chip也是这种单口RAM。



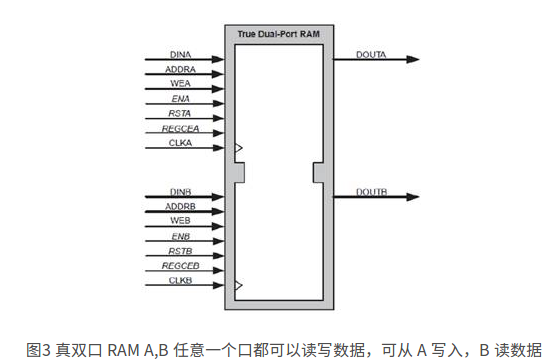
###### 简单双口SDPRAM

只能从A口写，从B口读



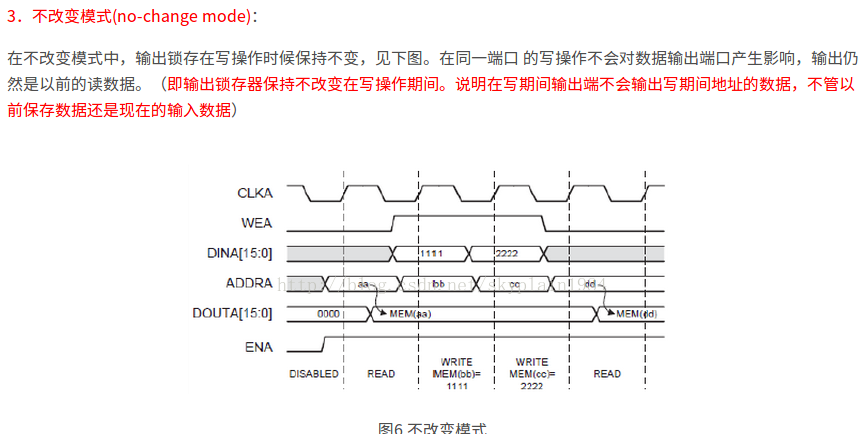
###### 真双口SDPRAM

A，B任意一个端口都可以读或者写数据

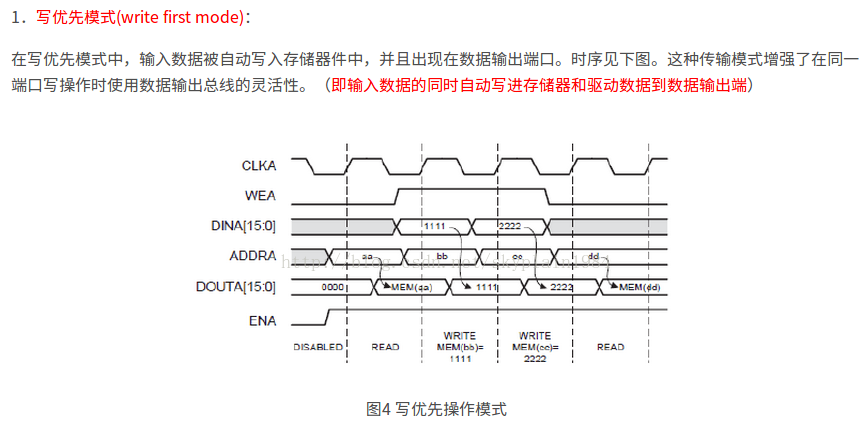


###### 读/写/不变三种模式

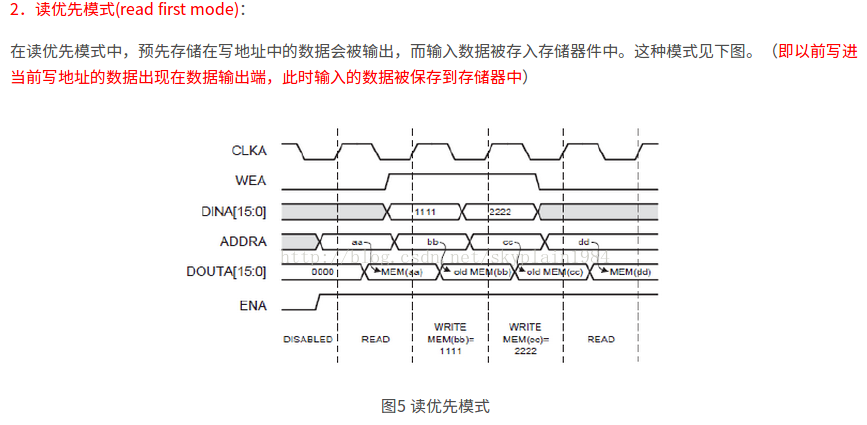
不改变模式



写优先



读优先

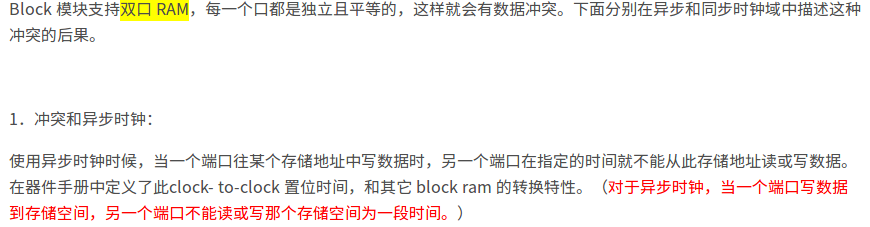


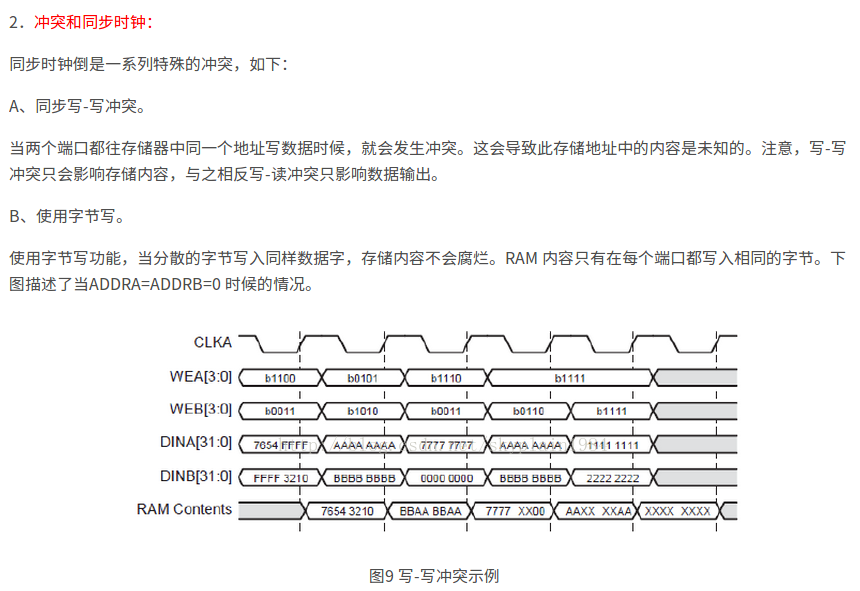
###### 冲突

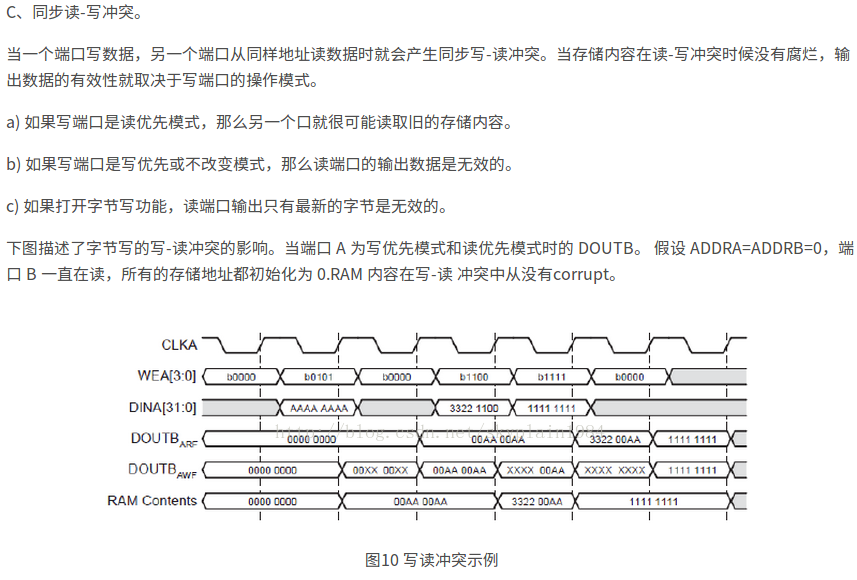
异步时钟下的冲突处理，此处很关键，RAM的冲突处理机制，一定要谨慎。

异步时钟的时候，当一个端口往某个存储器地址中写数据时，另一个端口在指定的时间内，

不能写，也不能读数据。 这里面有 clock-to-clock 置位时间。







##### DRAM动态随机存储器

见下文

###### 如何设计

#### PLL

见下文

#### DSP

介绍一下DSP的结构，如何做的乘法器等等。

#### 浮点数IP

浮点数=符号位+指数位+尾数位

### 标准IP

这里的IP主要指能实现一定稍微复杂点的功能的IP。

#### DDR控制器

见下文

#### PCIE控制器

见下文

#### 1000M以太网控制器

见下文

#### 10000M以太网控制器

见下文

## 典型电路设计

这里讲一些经典电路，比如分频器、复位电路，除法器等等，复习一下大学的数字电路设计中，卡诺图，状态图化简等等知识。

### 存储器类相关电路设计

注意问题：带有时钟的RAM电路，电路结构啥样的，双口RAM电路结构，

#### FIFO

##### 同步FIFO

同步FIFO设计简单，下面重点介绍异步FIFO

##### 异步FIFO

###### 异步FIFO的设计思路

采用双口RAM+Gray码的方式进行设计，特别注意，

1. FIFO的深度必须是2的n次密，因为只有这个数，才能保证Gray的循环都能保证每次都是只有1位变化。
2. 异步FIFO的空和满判断条件，都是精确到来或者提前到来，晚些撤销。也就是说，FIFO的空满判断更加保守，一定程度上损失了一点点性能，这个性能基本可以忽略不计。

特别注意，当写慢读快时，可能会存在empty提前到来，然后不读，一会儿后又非空，此时可能还得读净数据的情况。

1. Gray码的转换方法。

二进制数rbinnext 的 bit0和bit1进行异或，得到结果作为新的bit0

Bit1和bit2进行异或，得到结果作为新的bit1

Bit2和bit3进行异或，得到结果作为新的bi2

... ...

Bit n-2和bitn-1进行异或，得到结果作为新的bitn-1

Bit n-1和bitn(就是0了)进行异或，得到结果作为新的bitn-1

这样bitn-1 还是本身。因此得到下式。



###### 参考论文

论文1：SimulationAndSynthesisTechniquesForAsynchronousFIFODesign.pdf

介绍异步FIFO的设计方法，并提供了源代码。

论文2：SimulationAndSynthesisTechniquesForAsynchronousFIFODesignWithAsynchronousPointerComparisons.pdf

介绍比较器中的异步FIFO设计方法。

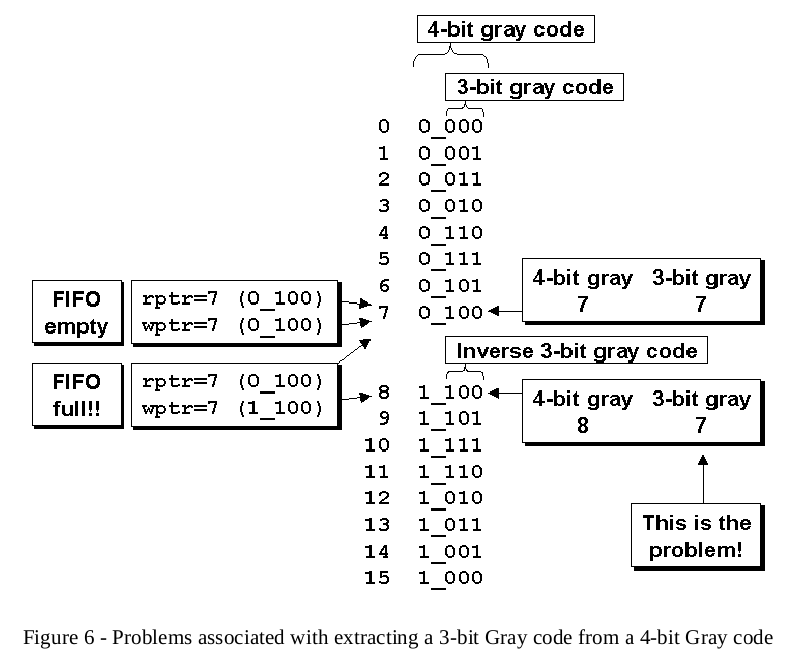
具体设计方法要详细研究论文即可

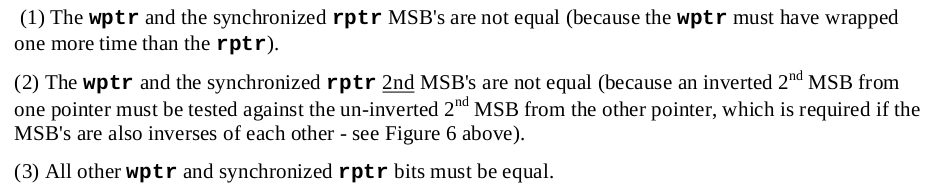
###### 论文疑点以及解决方法

论文1

问题1

问题描述：

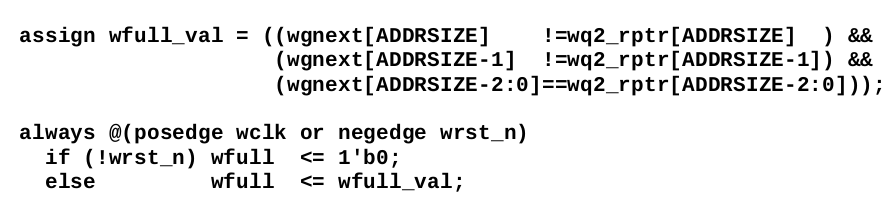




第2句话：(2) The wptr and synchronized rptr 2nd MSB’s are not equal。这段话如何理解。

问题1解答

解答：



我们知道代码中，wq2\_rptr也是传递的gray码，判断空和满时，两个gray码直接进行相等判断，这句话意思是，当满的情况，不仅要判断ADDRSIZE位置，还要判断ADDRSIZE-1位置，都要保证是异或关系才行，为什么ADDRSIZE-1这个位置的bit也要不同呢？

可以看上图Feature6 图中，满的情况，可以是

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 描述 | 读指针  （10进制） | GRAY码  （BIT4）  RMSB | GRAY码  （BIT3）  RMSB-1 | 写指针  （10进制） | GRAY码  （BIT4）  WMSB | GRAY码  （BIT3）  WMSB-1 |
| 数值 | 0 | 0 | 0 | 8 | 1 | 1 |
| 数值 | 1 | 0 | 0 | 9 | 1 | 1 |
| 数值 | 。。。 | 。。。 | 。。。 | 。。。 | 。。。 | 。。。 |
| 数值 | 6 | 0 | 1 | 14 | 1 | 0 |
| 数值 | 15 | 1 | 0 | 7 | 0 | 1 |

采用穷举法，可以看出，读指针的最高位以及次高位与写指针的最高位以及次高位都会处于不同状态。

比较对象： RMSB 与 WMSB进行比较

RMSB-1 与 WMSB-1进行比较。

论文2

SimulationAndSynthesisTechniquesForAsynchronousFIFODesignWithAsynchronousPointerComparisons.pdf

论文2中的fifo空满判断与论文1相比，更接近实际情况，但是个人认为编写风格不友好，不推荐。

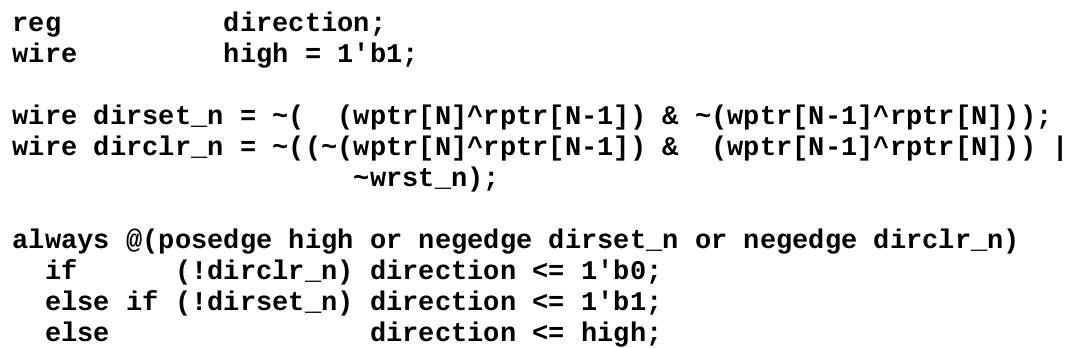
论文2中的fifo方法不推荐

论文中截取的一段话



虽然这篇论文获得了最佳论文，但是这篇论文中的编码风格与STA静态时序分析还有DFT不兼容，如果用户想友好使用STA和DFT，建议参考论文1.

为啥不兼容呢？



可以看到always块中，居然将high，dirset\_n等信号都作为敏感列表了，这种方式不推荐，因此不建议学习这篇论文。

#### 如何利用好单口RAM把它变成双口RAM来使用？

思路是，利用时钟的上升沿和下降沿，比如上升边沿到来时读，下降沿到来时，写。

## 时序分析篇

### FPGA时序分析基本知识

XDC时序约束文件，是xilinx公司的vivado工具使用的时序约束文件，本质是Tcl脚本，和他们之前的ISE中udf不同，忘了udf吧。

基础语法来源于业界统一的约束规范SDC（最早由Synopsys公司提出，故名Synopsys Design Constraints）。

等价于XDC和SDC类似即可。

### 时序分析基本概念

先讲时序分析基本概念，然后讲几种约束方法。

#### 触发沿Launch和所存沿

触发沿Launch Edge，源端寄存器触发数据边沿。

锁存沿Latch edge，目的端寄存器锁存数据的边沿。

注意，锁存沿Latch edge-Launch Edge为一个时钟周期T

#### 建立时间Tsu

建立时间(Setup Time)：在时钟有效沿到来前，数据必须保持稳定的最小时间(图3-1中的Tsu)，通常<1ns

#### 保持时间Th

保持时间(Hold Time)：在时钟有效沿到来后，数据必须保持稳定的最小时间(图3-1中的Th)，通常<1ns

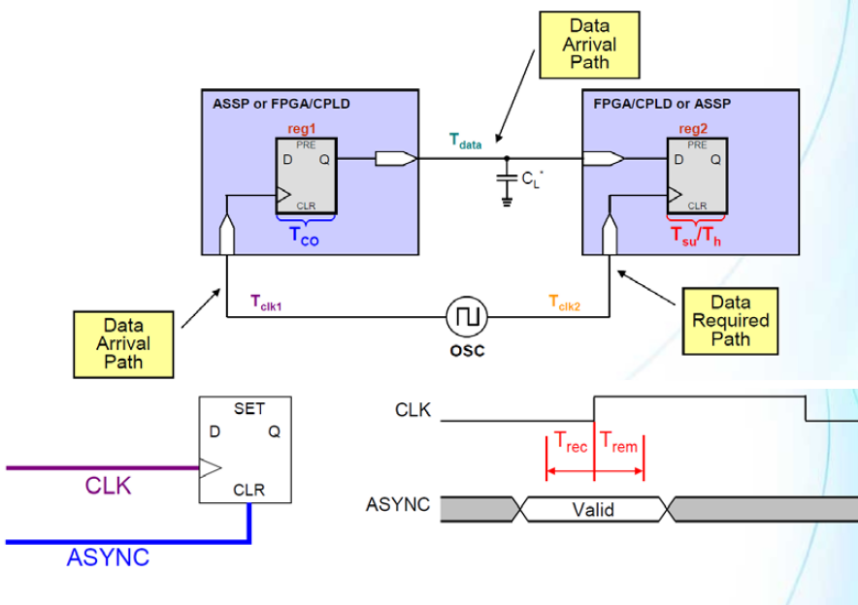


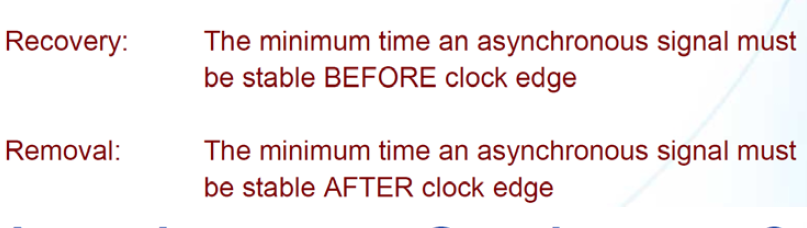
建立时间和保持时间

#### 异步信号恢复时间Trec和移除时间Trem

恢复时间Trecovery，相当于建立时间，

移除时间Tremoval，相当于保持时间。





Altera公司这么叫的，其他的没怎么听说过。

#### 走线延迟Tclk1

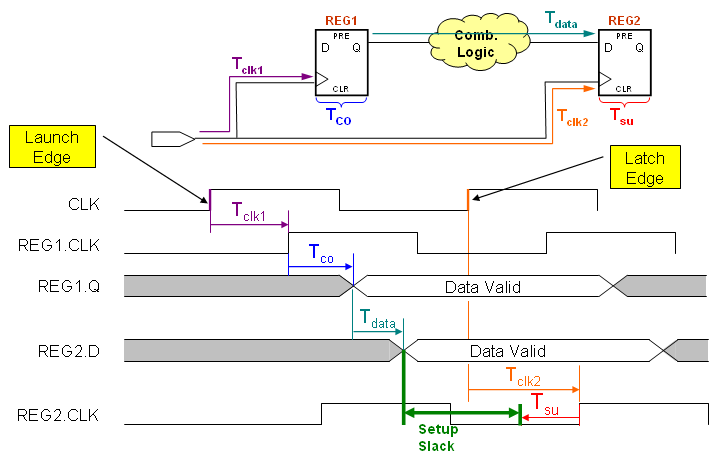
Route Delay Time 指：布局布线后走线所带来的延迟，比如下图中Tclk1.

#### 组合逻辑延迟Tdata

Combination Logical Delay：指电路中组合逻辑的存在所带来的延迟。Tdata。

#### 触发器的输出延迟Tco

指的是触发器的输出数据与有效时钟沿之间的延迟值。



时序分析基本概念图，分析建立时间

#### 数据到达时间TData Arrival Time

指的是数据到达目的寄存器D端的时间

**TData Arrival Time = Tclk1+ Tco+ Tdata**

#### 时钟到达时间TClock Arrival Time

指的是时钟目的寄存器时钟输入端的时间。

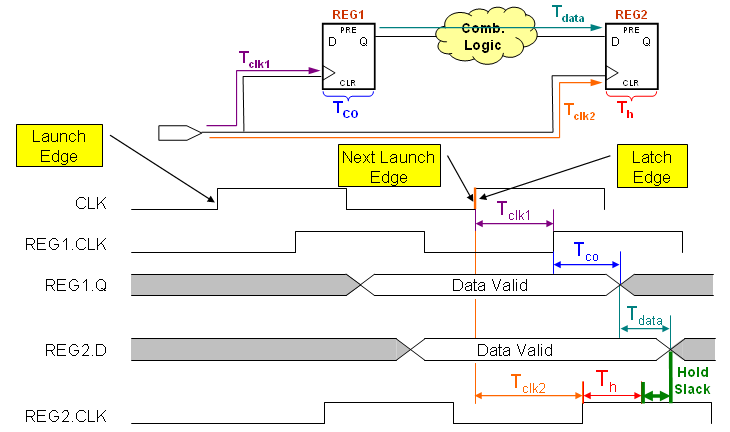
**TClock Arrival Time = 锁存沿时刻（latch Time）+Tclk2**

#### 数据要求时间TData required time

指的是Latch时钟沿到来后，将寄存器的建立时间，保持时间考虑进去，要求数据到达目的地的最小时间。

数据要求的建立时间Td\_su = **TClock Arrival Time - Tsu**

数据要求的保持时间Td\_h = **TClock Arrival Time + Th**



时序分析基本概念图，分析保持时间

#### 时钟偏斜Tskew

Tskew = **Tclk2- Tclk1**

时钟源的偏差，

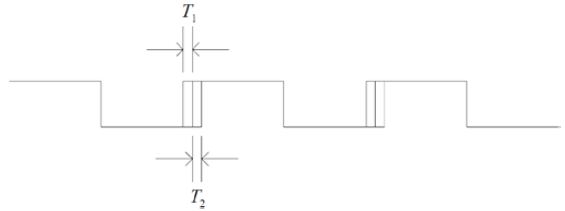
情况1：同一个PLL所输出的不同时钟信号之间的偏斜。

情况2：时钟分配网络的偏斜。



#### 时钟抖动Tjitter

指的是时钟边沿的输出位置和理想情况存在一定的偏差。



时钟信号边沿变化的不确定时间称之为时钟抖动。严格说，建立时间应该是Tsu + T1，而保持时间应该是Th + T2。

#### 时序裕量Tslack

指的是实际所需的时间和设计所需时间的差值，表示设计是否满足时序的一个称谓，>0时就是通过了。

##### 建立时间裕量

建立时间的裕量 **Tsu slack =（数据要求的到达时间）-（数据到达时间）**

**Td\_su** - **TData Arrival Time**

**= TClock Arrival Time - Tsu - TData Arrival Time**

**= 锁存沿时刻（latch Time）+Tclk2- Tsu - TData Arrival Time**

**= Tclk + Tclk2- Tsu - TData Arrival Time**

**Tsu slack = Tclk + Tclk2- Tsu - Tclk1 - Tco - Tdata**

##### 建立时间裕量与生活中的联系

让你8点开会，结果你7:55到了，那么裕量就是5分钟。

8点就是数据要求到达时间

7:55就是数据到达时间。

##### 保持时间裕量与生活中联系

让你8点半散会，结果你8：35走了，

8:30就是“数据要求的保持时间Td\_h”**=锁存沿时刻（latch Time）+Tclk2+ Th**

8:35就是，新的数据到达时间，**TData Arrival Time = Tclk1+ Tco+ Tdata**

保持时间裕量：**Th slack = Tclk1+ Tco+ Tdata-（Tclk2+ Th）= Tco+ Tdata-Th- Tskew >= 0**

##### 小总结

最小时钟周期**Tclk**  = **Tco+Tdata -Tskew +Tsu**

保持时间裕量**Th slack = Tco +Tdata -Tskew -Th >=0**

**特别注意，这里的Tco ， Tdata ， Tsu还有Th，都是非负数哦。**

这2个公式非常类似，区别在于最小时钟周期+T**su，**保持时间裕量-**Th**

从公式可以看出，如果保持时间裕量不够，那么一定是Tskew过大导致的结果，因此保持时间要是出现问题，就只能调整**Tskew了，**调整它把，让他变小，保持时间就好过了。

**Tskew** = **Tclk2- Tclk1 。这个意思是目的端时钟走线延迟-远端时钟的走线延迟。**

如果调整，尽量让他们等长把。

那么也就是全局时钟的概念。

##### Tskew如何设置

要使目的寄存器正确的采样到源端寄存器过来的数据，则Tskew ~0是最好的，因为Tskew>0时，保持时间的裕量变小；Tskew<0，建立时间的裕量变小。

##### 如何保证保持时间裕量

为保证到芯片的各个角落的延时尽量相等，时钟分配树要走到芯片的中间，再向芯片的四周分布。所以时钟源端到触发器的路径长，延迟大，但是没关系，只要保证到达各个寄存器的输入端等长就好了。保证SKew很小，就能保证可以上很高的主频了。

全局时钟网络的驱动能力特别强，主要是防止芯片内部的信号对时钟信号产生影响，这样就能保证时钟信号引入的抖动非常小。

#### 时钟周期与最高频率

当建立时间裕量为0时，能得到最小时钟周期，也就是最高频率

**Tclk + Tclk2- Tsu - Tclk1 - Tco - Tdata >= 0**

**Tclk >= Tsu + Tclk1 + Tco +Tdata - Tclk2 = Tco + Tdata + Tsu - Tskew**



### 时序约束

分为三大类

#### 时钟约束

##### 时序分析策略

我们希望器件在各种极限情况下，都能满足建立时间和保持时间要求，因此考虑如下两种极端模型，以Altera芯片时序模型为例，65nm。

（1）Slow Corner模型通过假设最大的环境温度（operating temperature）和VCCmin来模拟一条信号路径可能的最慢的情况。

（2）Fast Corner模型通过假设最小的环境温度（operating temperature）和VCCmin来模拟一条信号路径可能的最快的情况。这两个模型的意义在于，我们可以通过slow corner模型来保证建立时间的时序，通过fast corner来保证保持时间的时序（对于源同步来说必须使用）。由于一般情况下设计以建立时间违规为主，所以Timequest默认使用slow corner。

温度升高，信号传递时间变慢。

##### 全局时钟约束

根据建立时间和保持时间的裕量关系：

**Tsu slack = Tclk + Tclk2- Tsu - Tclk1 - Tco - Tdata**

**Th slack = Tco +Tdata -Tskew -Th >=0**

**为了满足建立时间和保持时间，Tskew为0最好。**

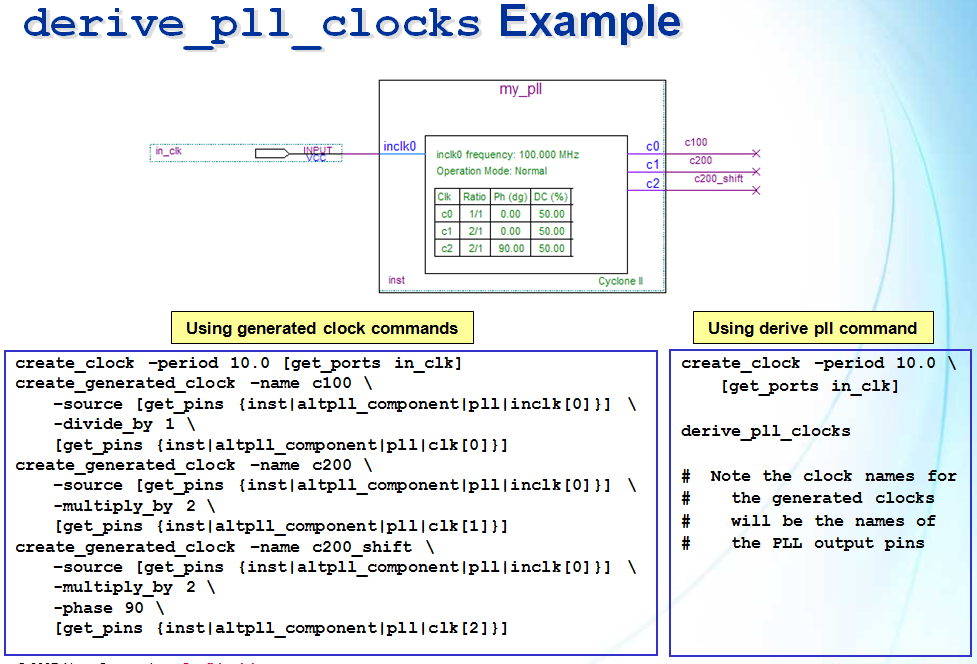
**分析原因：**

要使目的寄存器正确的采样到源端寄存器过来的数据，则Tskew ~0是最好的，因为Tskew>0时，保持时间的L；Tskew<0，建立时间的裕量变小。

为了保证到芯片的各个角落的延时尽量相等，时钟分配树首先是走到芯片的中间，再向芯片的四周分布，所以从时钟的源端到所驱动的触发器走过的路径比较长，延时比较大，但是到各个时序元件(触发器)时钟输入端等长，保证时钟偏斜(SKew)很小；全局时钟网络具有很强的驱动能力，而且在芯片设计的时候对时钟网络做了保护，尽量防止芯片内部的信号对时钟信号质量有影响，这样可以保证时钟信号引入的抖动非常小。

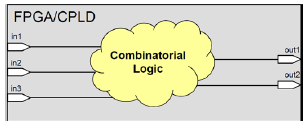
##### PLL的相关时钟约束（Altera）

###### derive\_pll\_clocks



#### I/O约束

##### set\_max\_delay/set\_min\_delay约束



Set Max Delay和Set Min Delay用来指定两点间的最大延迟和最小延迟。这个是寄存器内部的延迟。

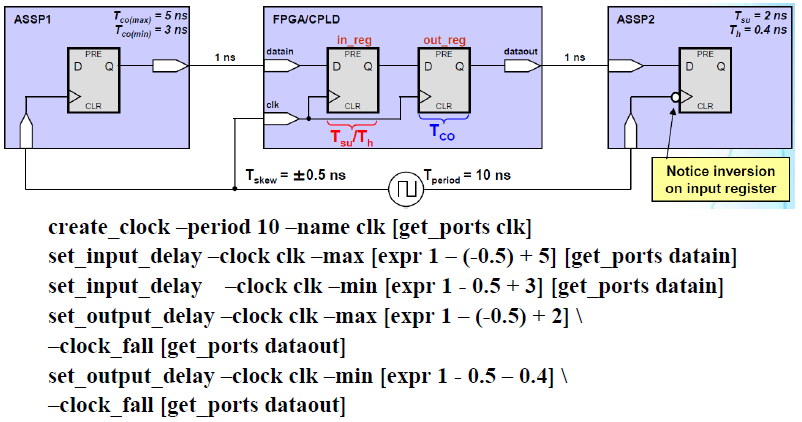
##### set\_input\_delay约束

具体参见Input Delay的动画。

##### set\_output\_delay约束

具体参见Output Delay的动画。

##### I/O约束例题1

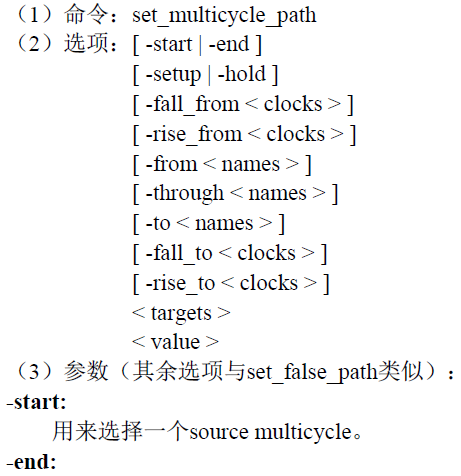


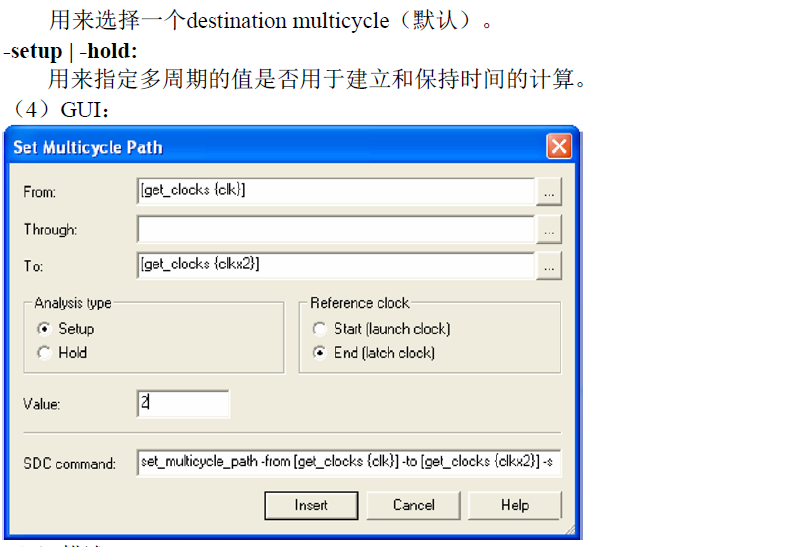
#### 时序例外约束

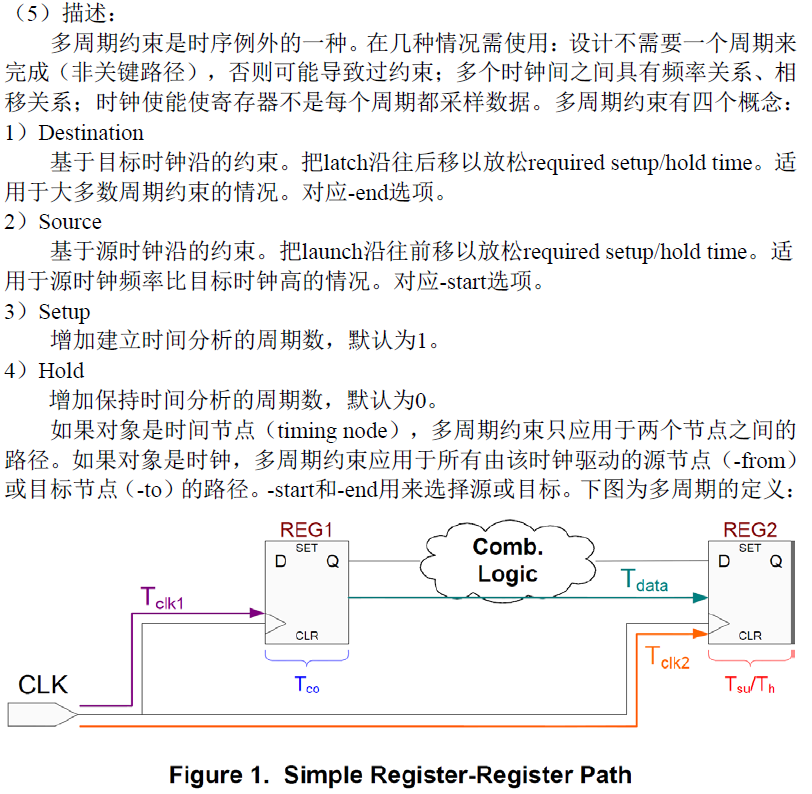
##### set\_multicycle\_path约束

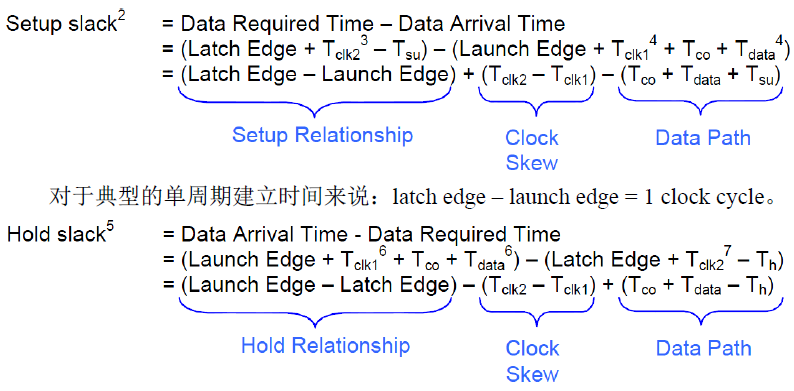
多时钟周期路径的约束，实际上可以理解为单时钟周期约束的扩展，这里资料参见《通向FPGA之路---七天玩转Altera时序篇V1.0》

###### 命令参数理解









此处注意，这个地方，应该是+号。对于典型的单周期保持时间来说：launch edge – latch edge = 0 clock cycle。注，setup和hold两个slack检查所使用的launch和latch不是同一个，参考定义： 对于建立时间检查，每一个目标寄存器的latch edge都寻找一个之前的离他最近的源寄存器的launch edge。即：setup check = current latch edge – closest previous launch edge 对于保持时间检查，Timequest有两种hold checks。第一次保持时间检查用于确定当前发射沿的数据不会被前一个锁存沿捕获。第二个保持时间检查用于确定下一个发射沿的数据不会被当前锁存沿捕获。

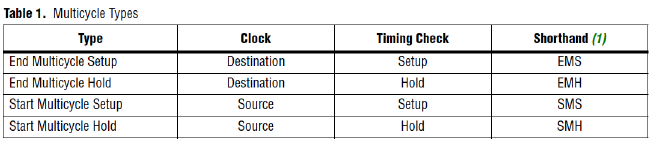
即：hold check 1 = current launch edge – previous latch edge

hold check 2 = next launch edge –current latch edge

到此，我们可以总结如下四种Multicycle Types。

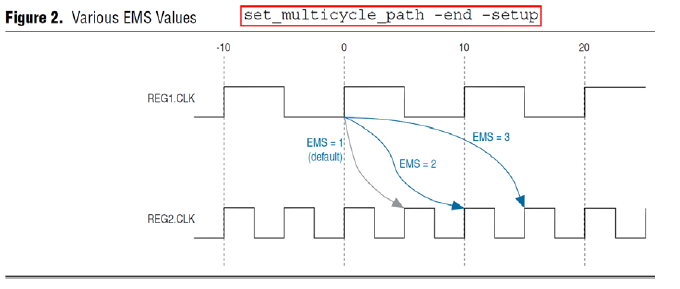
这个问题在多周期路径中，保持时间比较难搞了，要保证每个沿都要满足保持时间。分成4种情况。

时钟有快有慢，需要考虑建立时间和保持时间，以及源时钟，目的时钟两种情况，因此共有4个情况。

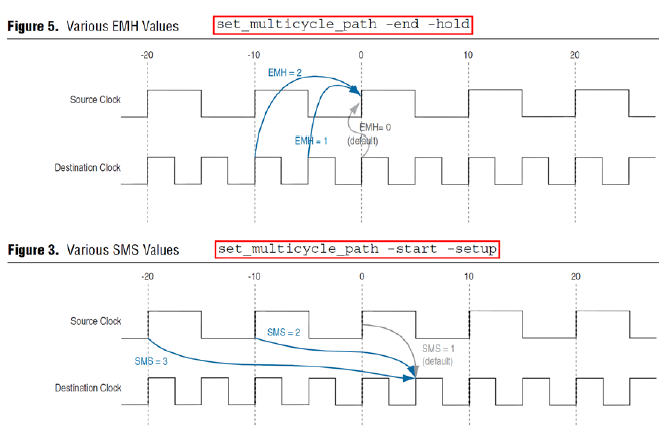


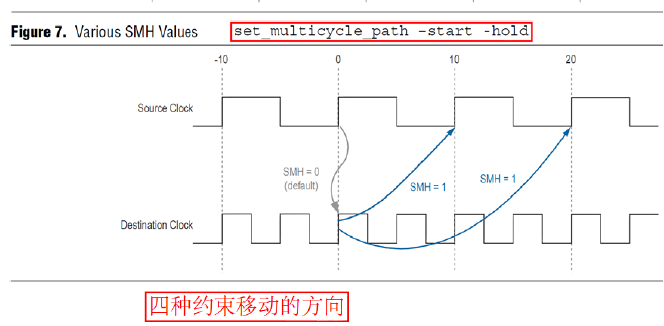
注意命名方式，E表示End，M表示Multicycle，S表示Setup

S表示Start，M表示Multicycle，H表示Hold



这个图说的是EMS，表示要看End clock的个数，S表示建立时间。





###### 小总结

对于基本的时序模型来说，建立时钟数是1，保持是0.

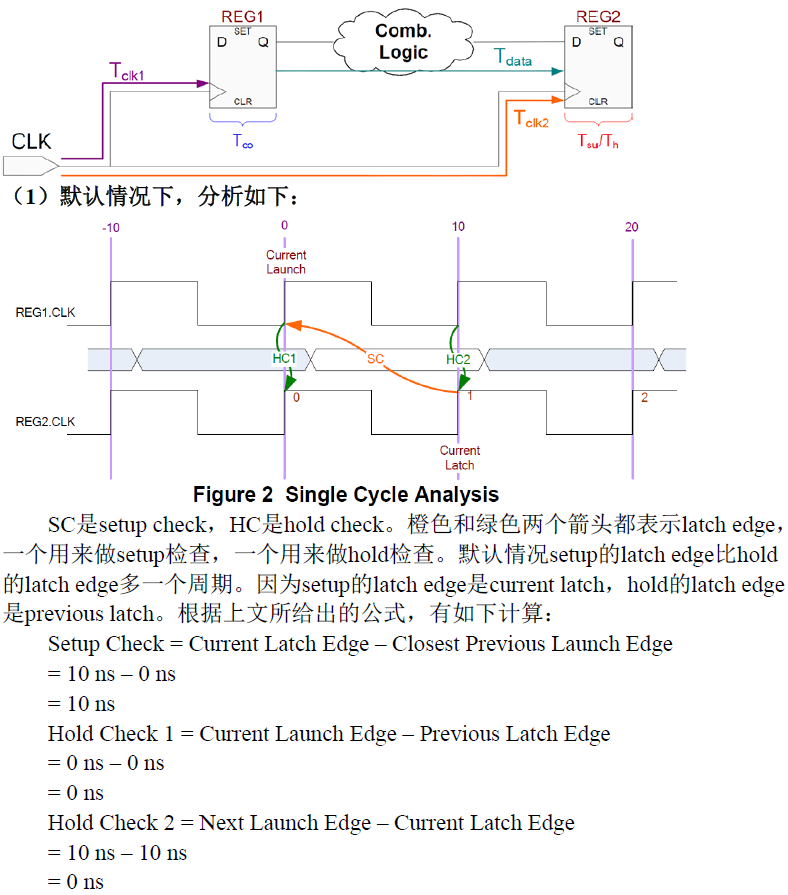
set\_multicycle\_path -setup 1 -from A -to B

set\_multicycle\_path -hold 0 -from A -to B

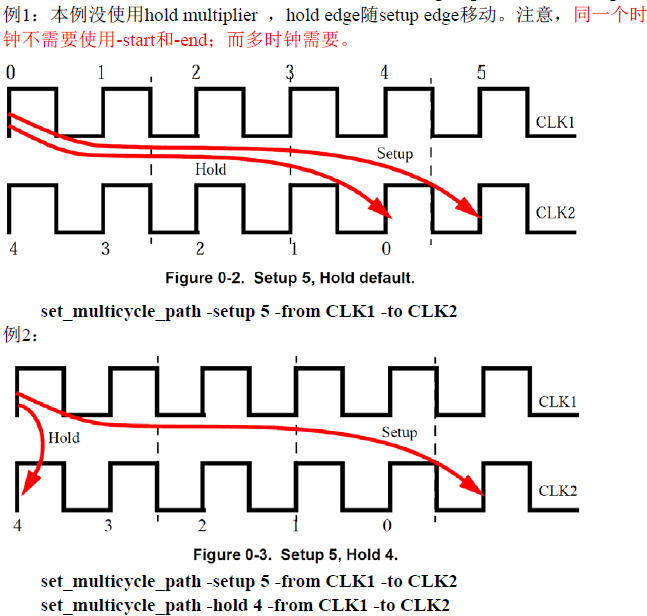
而且对于单时钟域来说，保持时间的时钟个数是建立时间的-1.

###### 单时钟域set\_multicycle\_path

考虑单时钟的情况。

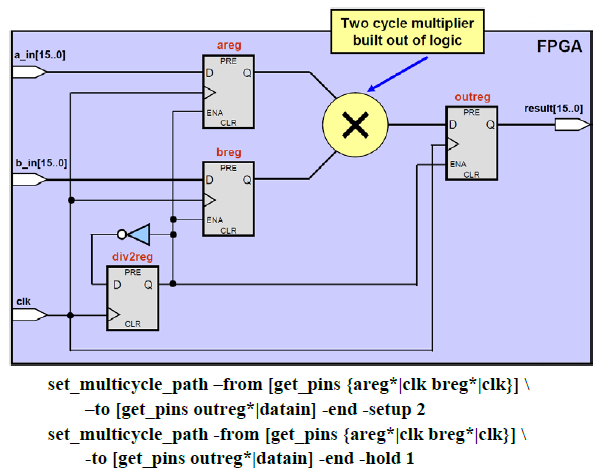


###### 单时钟域实例1



###### 单时钟域实例2

乘法器需要两个时钟来计算结果，同时本设计为单时钟的多周期约束。不需要-start和-end的指定。



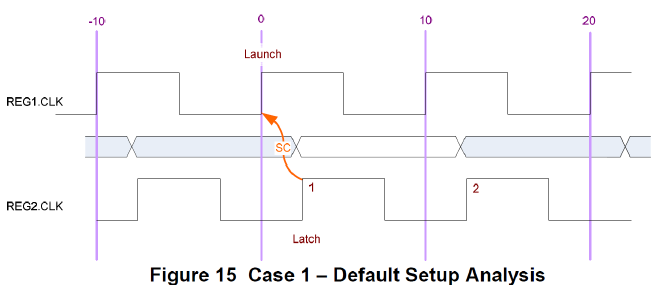
###### 多时钟域set\_multicycle\_path

多周期下多时钟域进行分析时，情况较多，分为5类根据Start和End频率不同按照如下方式进行划分。

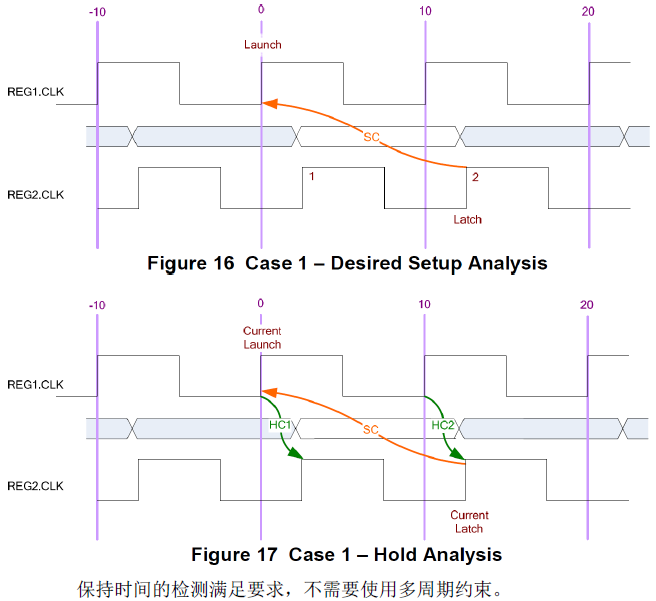
情况1

Freq(dst) = Freq(src) + Offset

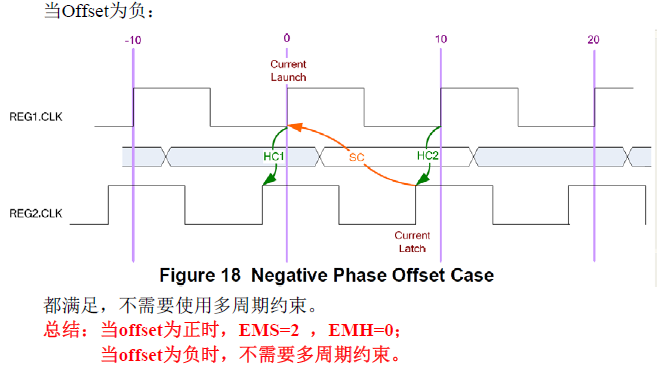
当Offset大于0时，



见下文

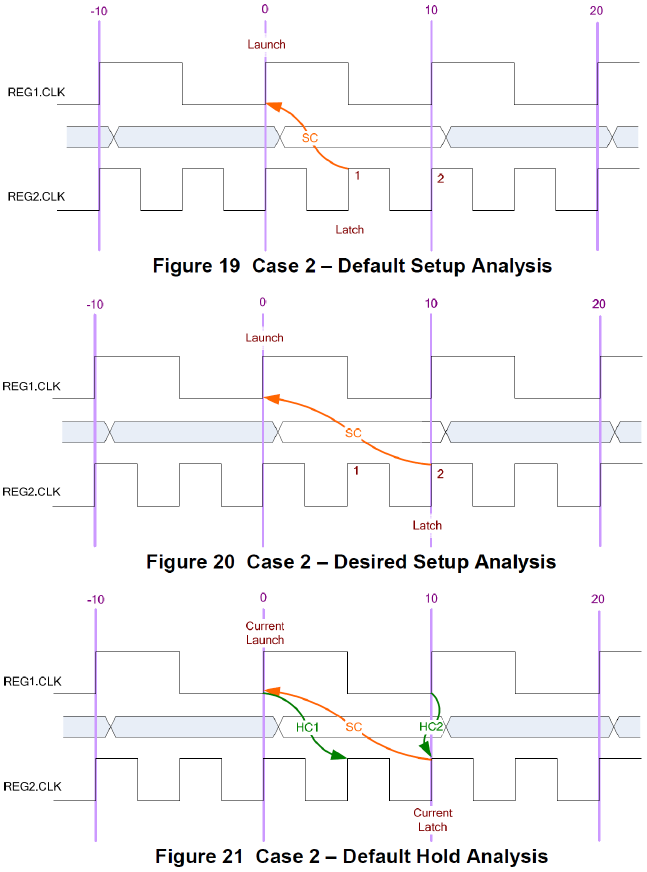


当Offset小于0时，目的慢，源快点



情况2

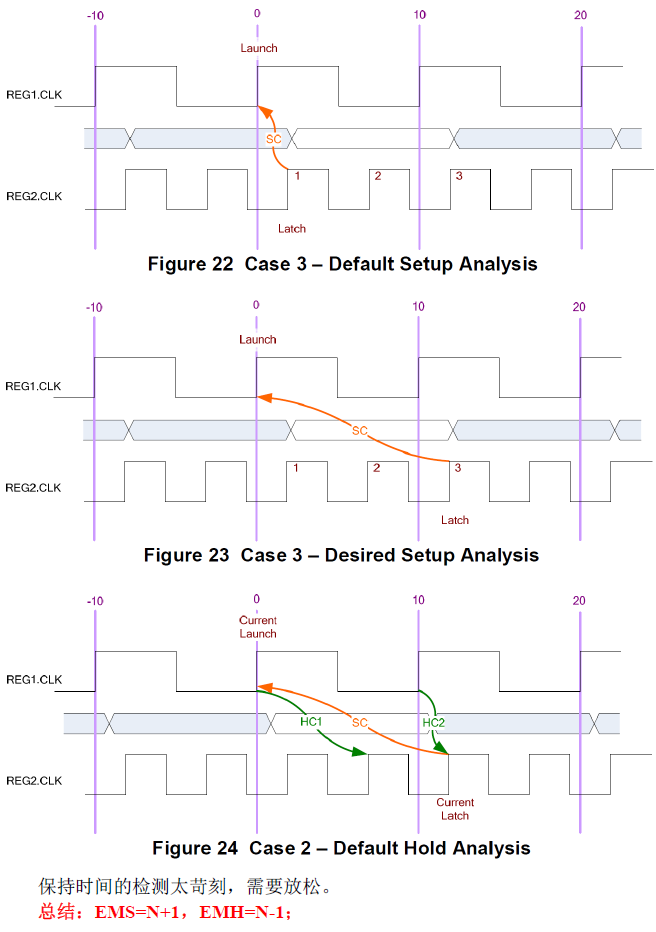
Freq(dst) = N\*Freq(src) 时，N>1，恰好为整数倍





情况3

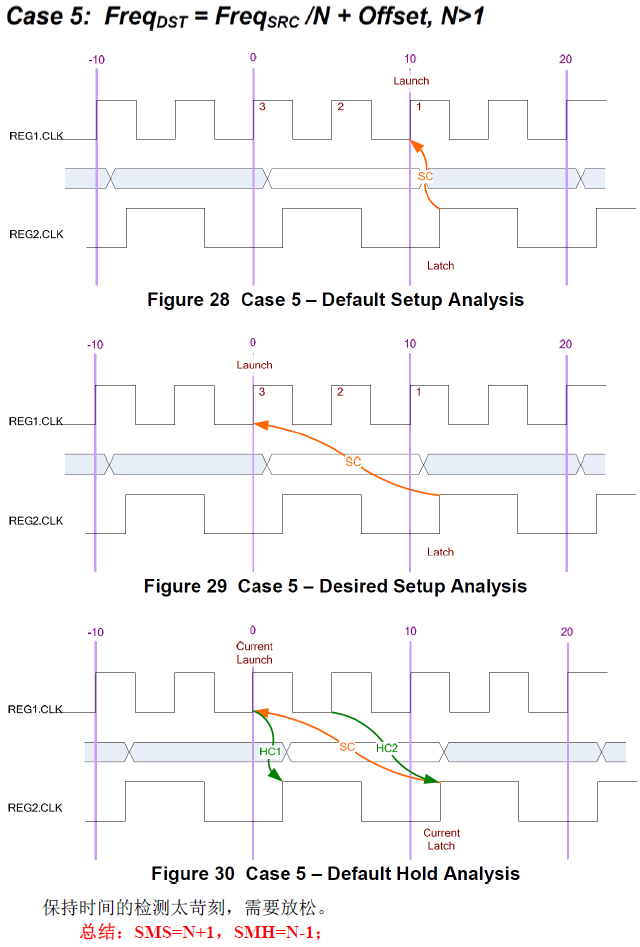
Freq(dst) = N\*Freq(src)+offset 时，N>1，恰好为整数倍



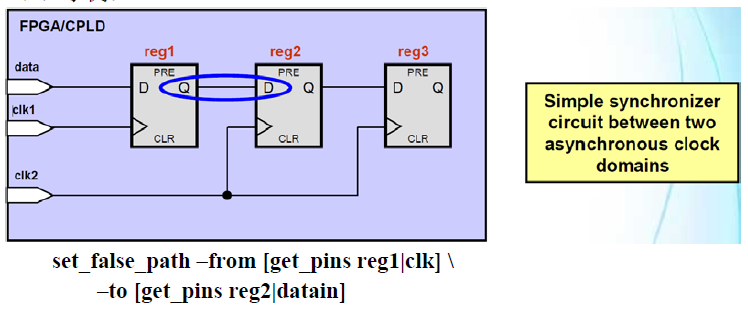
情况4



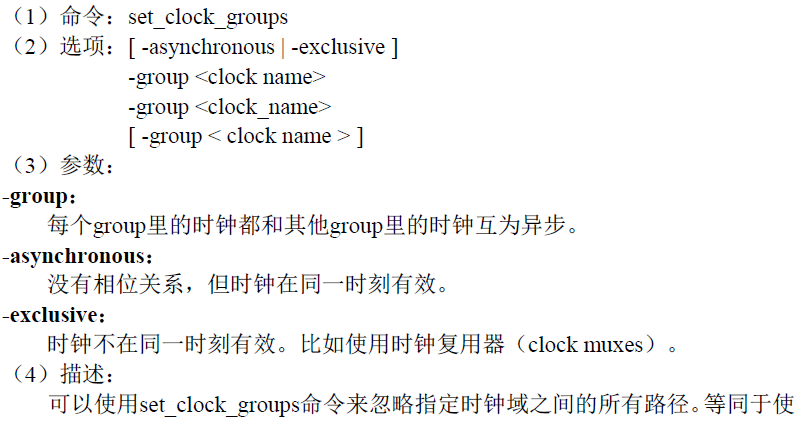
情况5



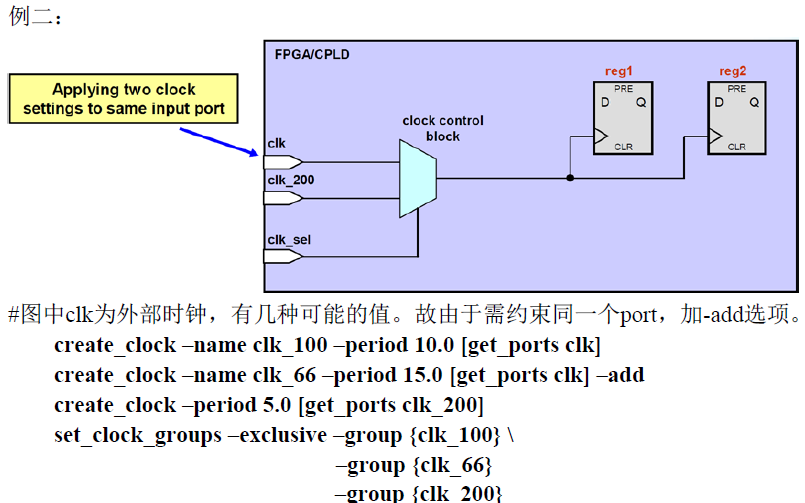
##### set\_false\_path约束



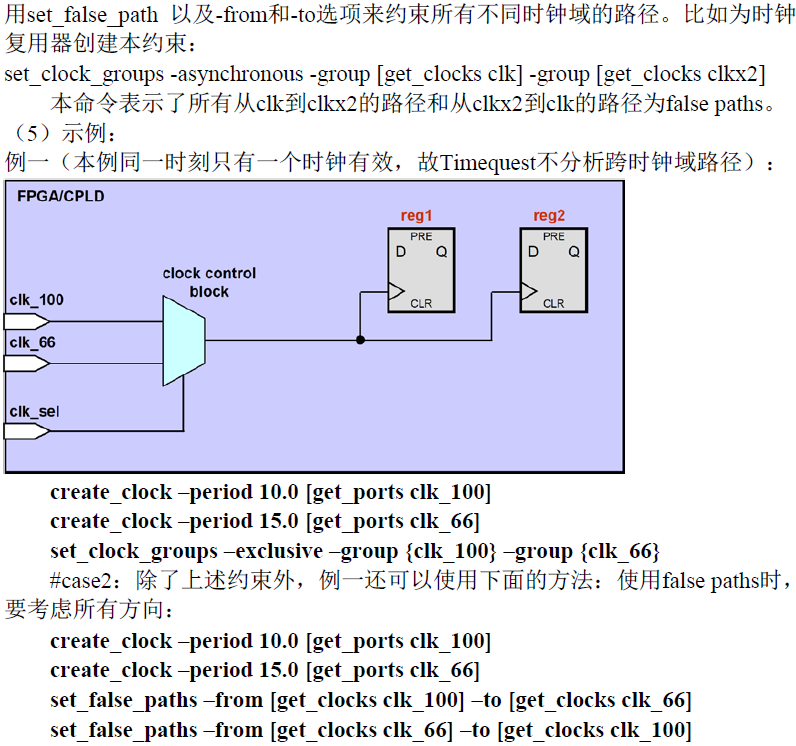
##### set\_clock\_groups约束



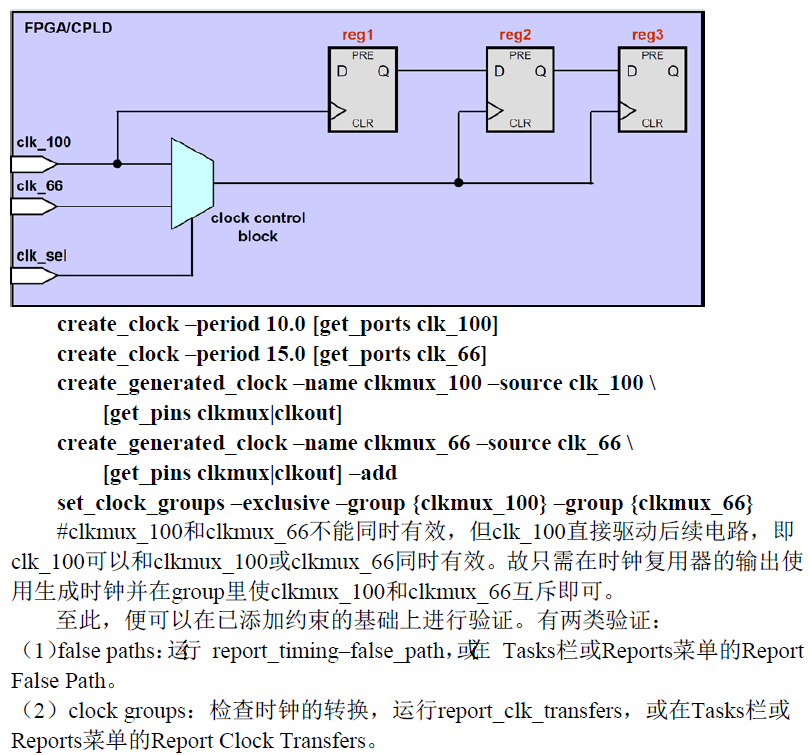
##### 时序例外例题1



##### 时序例外例题2



##### 时序例外例题3



### 时序约束之外常用法宝

#### FPGA方面

##### 增强时钟驱动能力

对时钟使用，比如Xilinx器件，使用ODDR，增强时钟驱动性能。或者包括时钟反向输出等等。 Altera的连接专用时钟管脚，再上PLL，提升时钟驱动能力。

##### I/O约束上FastIO或IOB

这块也叫Tsu和Tco的约束。

IO的约束方面，Altera芯片，上FAST IO，使得FPGA的延迟路径固定。Xilinx芯片，可以考虑上IOB，使得外部pin脚过来后，直接上IOB即可。

##### 固定路径Post-fit Netlist增量编译

引入Post-fit Netlist的过程是从一次成功的时序收敛结果开始，把特定的一组逻辑（设计分区）在FPGA上实现的布局位置和布线结果固定下来，保证这一布局布线结果可以在新的编译中重现，相应地，这一组逻辑的时序收敛结果也就得到了保证。这个部分保留上一次编译结果的过程就是增量编译，保留的网表类型和保留的程度都可以设置，从而获得相应的保留力度和优化效果。由于有了EDA工具的有力支持，虽然是精确到门级的细粒度约束，设计者只须进行一系列设置操作即可，不需要关心布局和布线的具体信息。由于精确到门级的约束内容过于繁多，在qsf文件中保存不下，得到保留的网表可以以Partial Netlist的形式输出到一个单独的文件qxp中，配合qsf文件中的粗略配置信息一起完成增量编译。

##### 固定区域LogicLock

LogicLock是在FPGA器件底层进行的布局约束。LogicLock的约束是粗粒度的，只规定设计顶层模块或子模块可以调整的布局位置和大小（LogicLock域）。

成功的LogicLock需要设计者对可能的时序收敛目标作出预计，考虑特定逻辑资源（引脚、存储器、DSP）与LogicLock Region的位置关系对时序的影响，并可以参考上一次时序成功收敛的结果。这一权衡和规划FPGA底层物理布局的过程就是FloorPlanning。LogicLock给了设计者对布局位置和范围更多的控制权，可以有效地向EDA工具传递设计者的设计意图，避免EDA工具由于缺乏布局优先级信息而盲目优化非关键路径。由于模块在每一次编译中的布局位置变化被限定在了最优的固定范围内，时序收敛结果的可重现性也就更高。由于其粗粒度特性，LogicLock的约束信息并不很多，可以在qsf文件中得到保留。需要注意的是，Logiclock是物理约束，design partition是逻辑约束。

##### 特定路径延迟约束MaxDelay/MinDelay

好的时序约束应该是“引导型”的，而不应该是“强制型”的。通过给出设计中关键路径的时序延迟范围，把具体而细微的工作留给EDA工具在该约束的限定范围内自由实现。这也是一个理想目标，需要设计者对每一条时序路径都做到心中有数，需要设计者分清哪些路径是可以通过核心频率和简单的时序例外约束就可以收敛的，哪些路径是必须制定MaxDelay和MinDelay的，一条也不能遗漏，并且还需要EDA工具“善解人意”的有力支持。设定路径延时约束就是间接地设定布局布线约束，但是比上述3、4、5的方法更灵活，而且不失其准确性。通过时序约束而不是显式的布局和网表约束来达到时序收敛才是时序约束的真谛。“好的时序是设计出来的，不是约束出来的”。好的约束必须以好的设计为前提。没有好的设计，在约束上下再大的功夫也是没有意义的。不过，通过正确的约束也可以检查设计的优劣，通过时序分析报告可以检查出设计上时序考虑不周的地方，从而加以修改。通过几次“分析—修改—分析”的迭代也可以达到完善设计的目标。应该说，设计是约束的根本，约束是设计的保证，二者是相辅相成的关系。

##### 设置Fan Out

用于内部Register到Register之间的约束，可以使用Max Fanout，使得输出端挂载的负载尽可能少，这样驱动能力更强，使得时序电路的建立时间更容易达到，从而得到更高的主频。

#### IC芯片方面

暂时无，没有总结

### 其他DDR的时序约束

考虑双沿都采样的情况，没具体分析过，暂时略掉，后续进阶再学习。

《七天玩转FPGA之时序篇里面有一些DDR时序介绍文章，可以参考》

### 异步时钟域电路设计

#### 参考论文

见下文《Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs》

##### 异步时钟域设计指导

###### 保证单module下只有一个时钟域



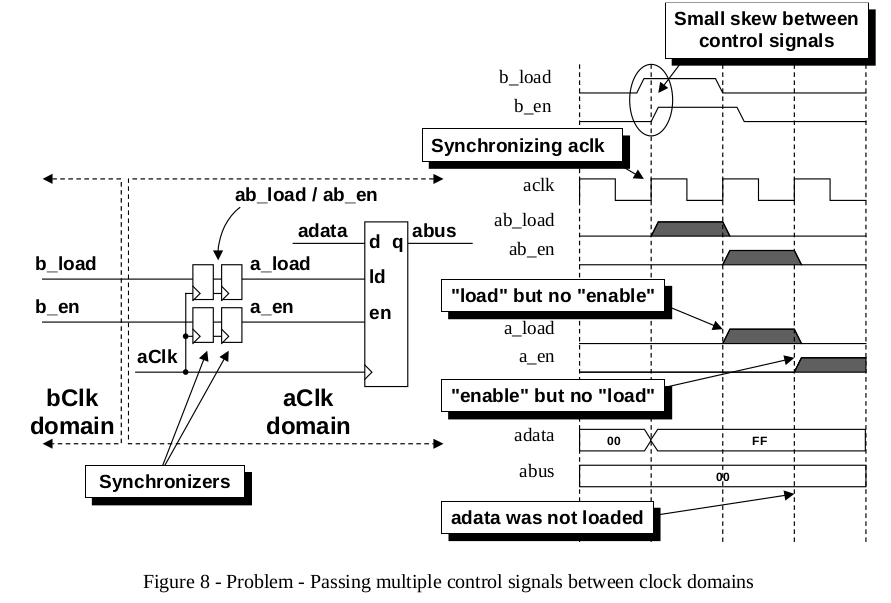
a.单个时钟域下面，只有一个模块。



单个时钟域下面，只有一个模块，这样做之后，时序分析变得更加容易。

B.按照模块区分后，更加容易编写时序分析的脚本

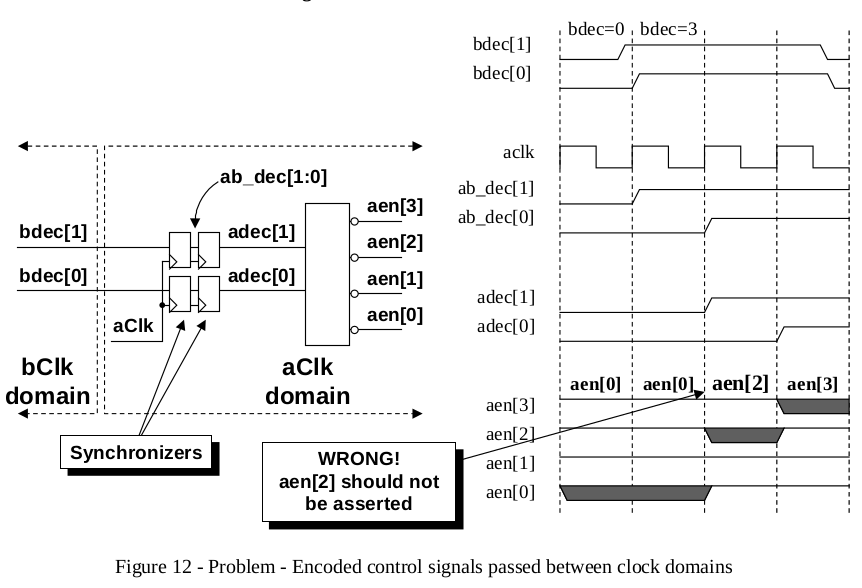
###### 多个控制信号进行跨时钟域传输时，先转成单信号



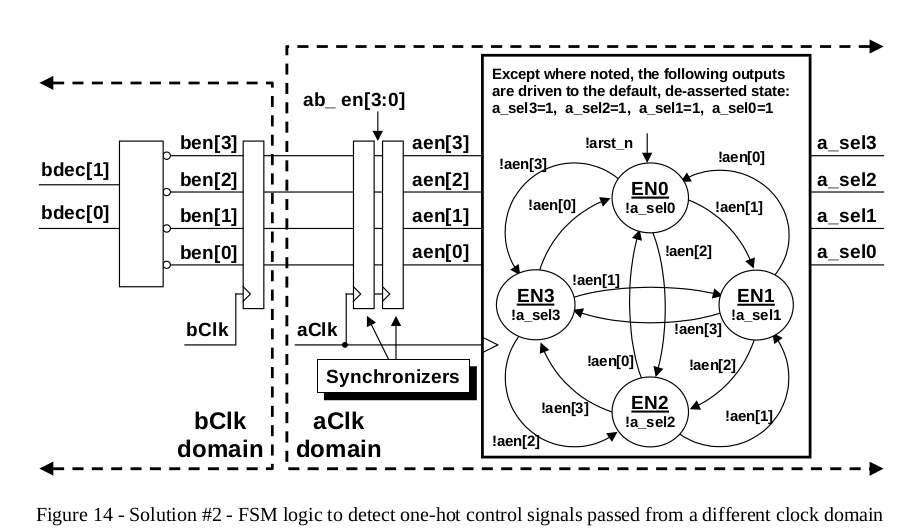
这种2个控制信号进行跨时钟域传输时，会存在不对齐的情况，必然导致出错，一个简单的方法是，在bClk时钟域下，对b\_load和b\_en进行逻辑操作，转成单bit信号，再打2拍就ok了。

###### 多个控制信号进行跨时钟域传输时，有编码的情况，采用状态机

跨时钟域踩错的情况下，导致了错误的aen[2]状态。

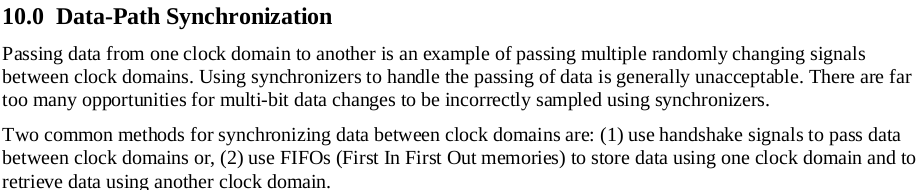


解决方法，采用状态机的办法，进行过滤，因为同一时刻仅有个pulse，当有2个信号为1时，必然是要把就的pulse1干掉，保留新的，就能保证得到正确结果。



###### 多个数据信号进行跨时钟域传输时，策略方法

对于多位宽的慢速信号，可以采用handshake方式，快速信号，就用异步fifo吧。



## 硬件加速篇

这里讲讲流水线，并行，展开，折叠，脉动电路，哈希算法等等

## Verilog语法篇

讲讲用HLS如何实现硬件加速。

### 语法基本介绍

#### 语法

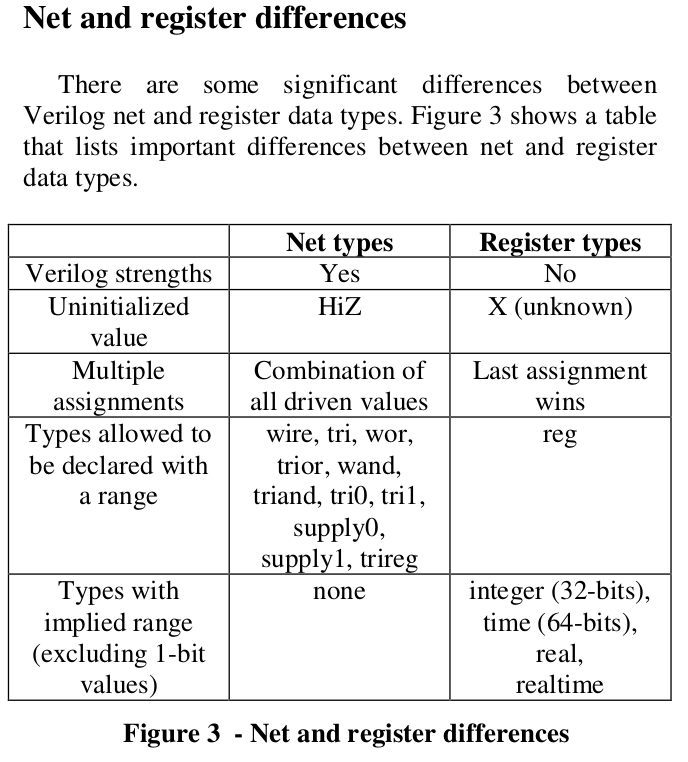
见下文

##### Net和Register区别

参考论文《A Proposal To Remove Those Ugly Register Data Types From Verilog》，建议从verilog中删除那些丑陋的寄存器数据类型。

注意，这里面，如果是reg类型，那么最后一条声明的值是最终的值。

如果是wire类型，那么最后的值是所有组合逻辑共同驱动的结果。



### Verilog代码编写指导

#### 如何正确使用复位

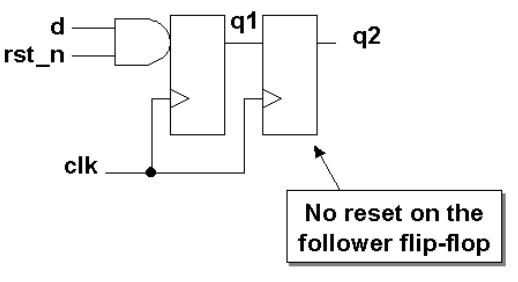
##### 论文1

参考论文《Synchronous Resets? Asynchronous Resets?

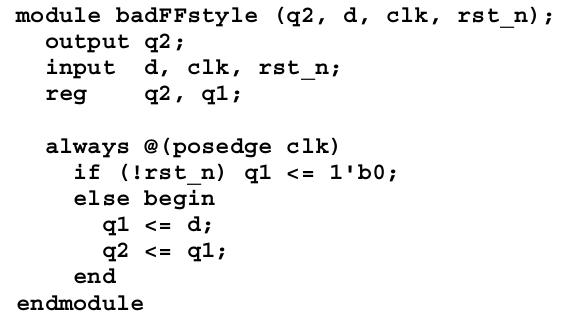
I am so confused! How will I ever know which to use?》

###### 内部寄存器复位写法

推荐写法：

错误写法：

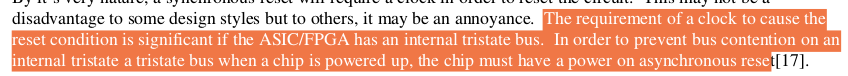


###### 同步复位寄存器的问题

如果复位信号是那种很长的组合逻辑产生的，那么可能会有问题。

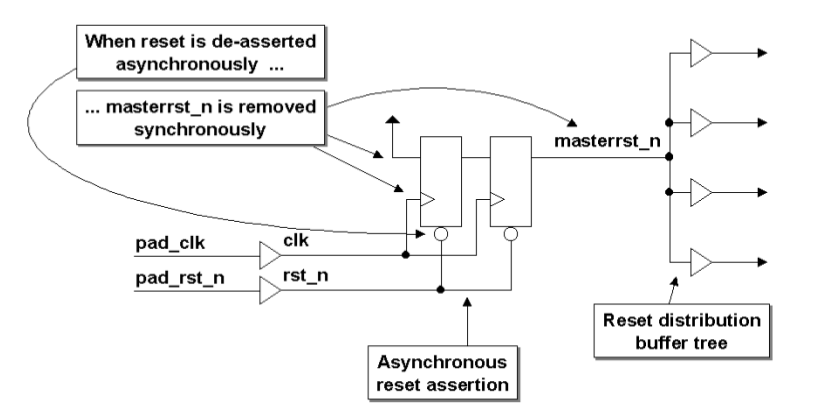


当有三态门的使用过程中，同步复位会有问题。

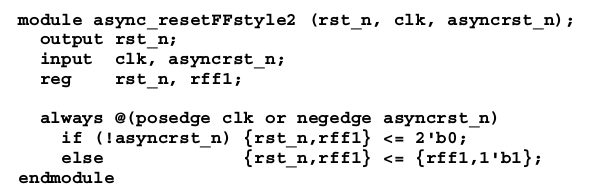


###### 异步复位，同步释放电路

图中复位信号电平有效，

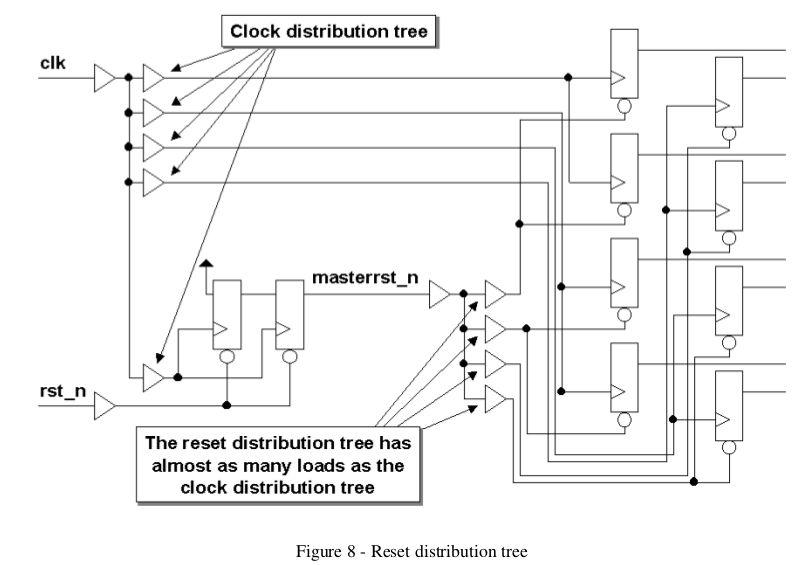


对应代码：



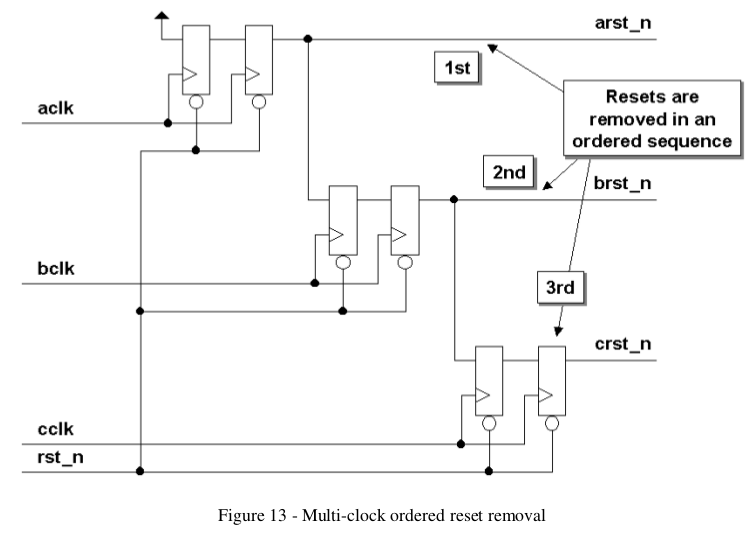
###### 复位树问题

复位树的时序要求没有时钟树那样严格，主要是skew要求不是太高，但是复位树要求复位信号必须再下一个时钟边沿到来前，释放复位信号。



###### 多时钟的异步复位问题

根据实际情况，图中的电路是一种选择，当然也有各自时钟域下各自复位的场景。



##### 论文2

#### 如何写出仿真-综合-流片等价功能的代码

参考论文《RTL Coding Styles That Yield Simulation and Synthesis Mismatches》

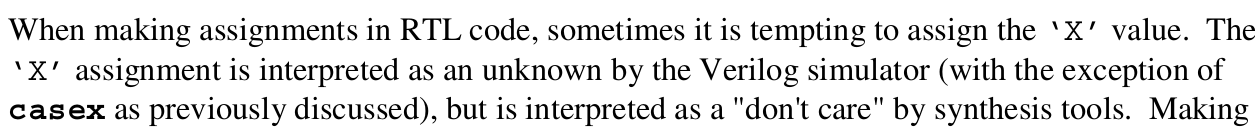
##### 不推荐的写法

###### Function谨慎使用，只生成组合逻辑

function内部的分支结果写全，避免推导出锁存器出来。

###### casex与casez不推荐使用

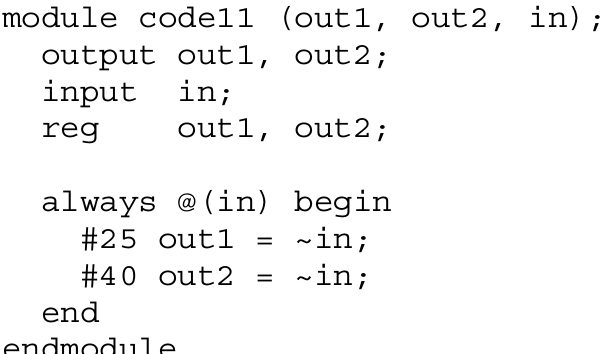
###### X谨慎使用



X的结果， 仿真器和综合工具推测出来的值是不一致的，谨慎使用，不过有时候为了fsm仿真之类的，写一个x能够快速定位到代码的问题。

###### 不要添加time delays指示信息

反面教材代码：添加这种延迟信息后，仿真器会认这些延迟信息，但是综合工具不会理睬这些信息，因此会导致综合结果与仿真不符的情况。



### Verilog仿真提速编码风格

#### 提速方法

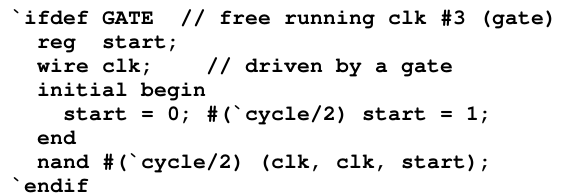
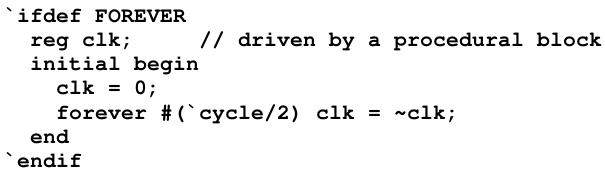
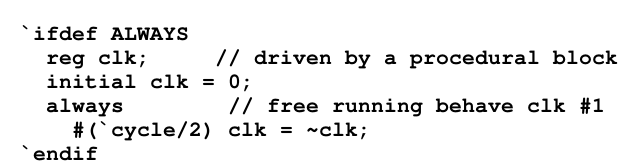
参考论文《Verilog Coding Styles for Improved Simulation Efficiency》

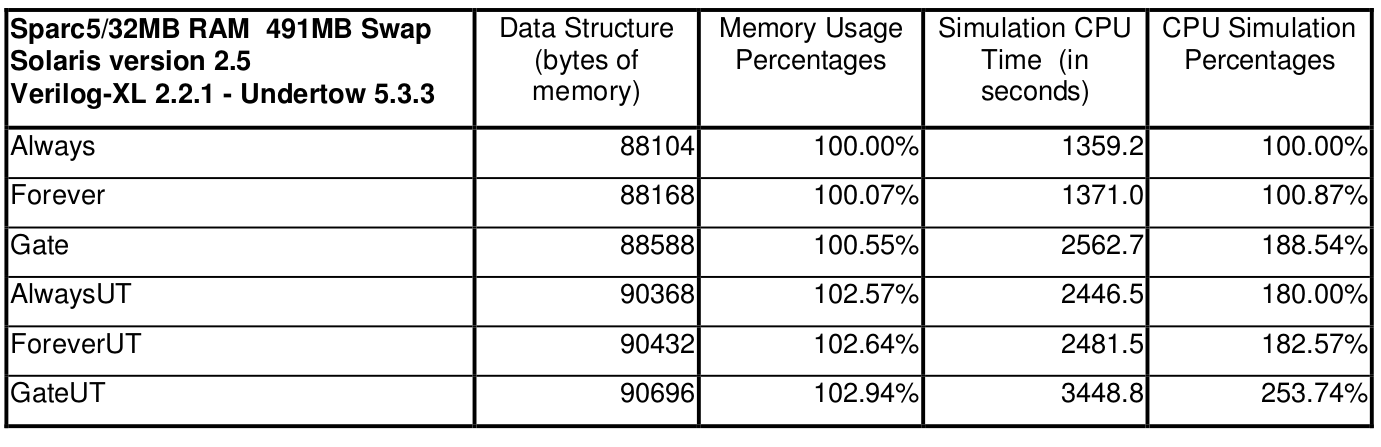
以下说的测试方法，都是在Verilog XL模拟器上面运行的，具有一定的指导意义，不是全部。

##### 虚拟时钟激励产生方法比较

采用always 的方法产生的时钟，仿真效率最高，推荐使用always的方法。

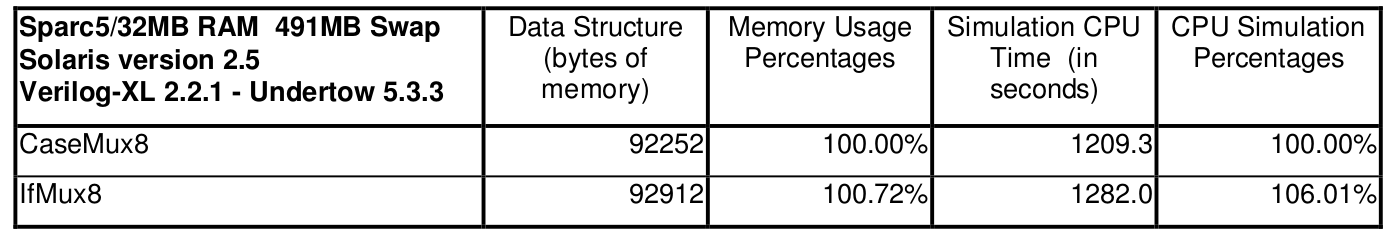
代码：





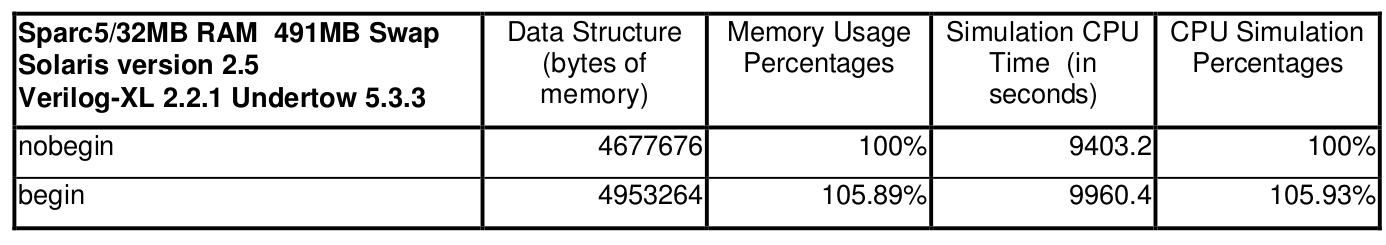
##### If-else和case比较

如果能用case实现的代码，仿真速度上，优于if-else方式。



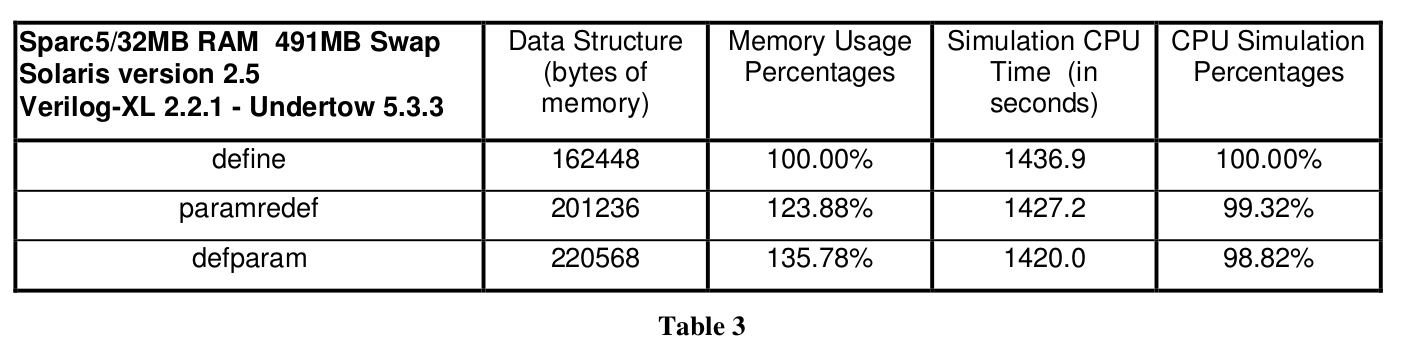
##### 添加无用的begin end代码

如果添加了无用的begin end代码，仿真速度上比不添加的方式要慢。



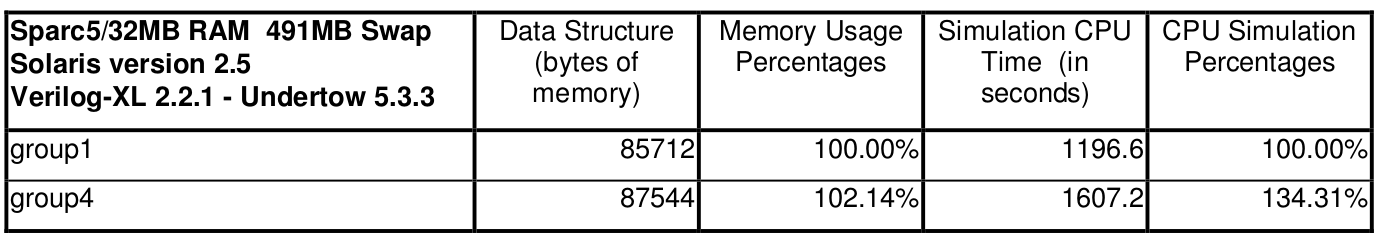
##### 添加define与parameter代码比较

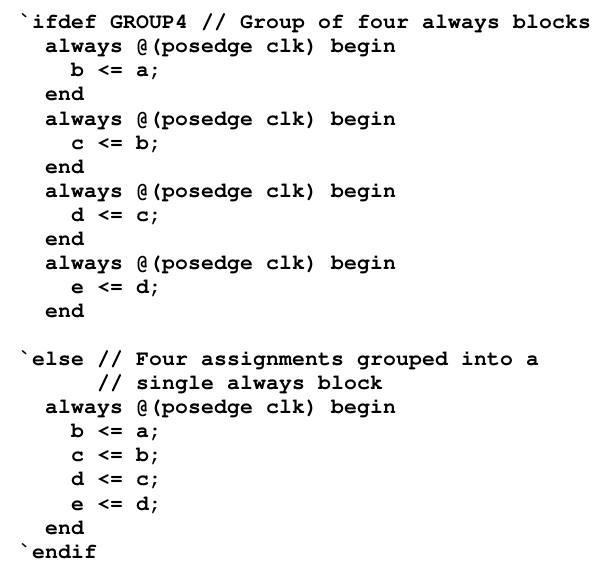
如果能用define做，那么仿真速度会快。



##### 同一个always块内多信号赋值

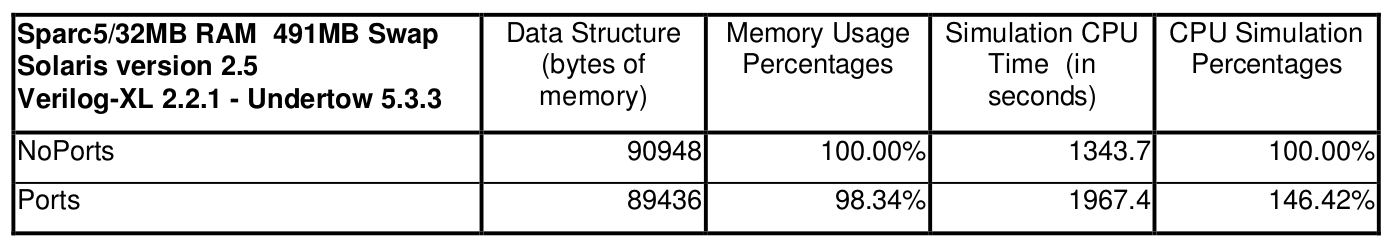
多信号赋值，比分散的always块要仿真快一些。





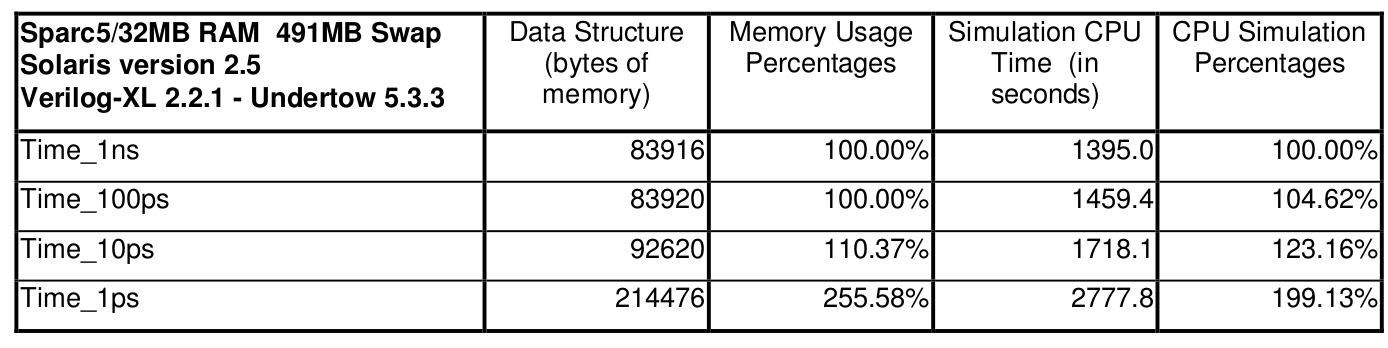
##### 跨模块的端口互联与非互联相比

跨模块的情况下，端口互联的情况比非互联的情况要慢。



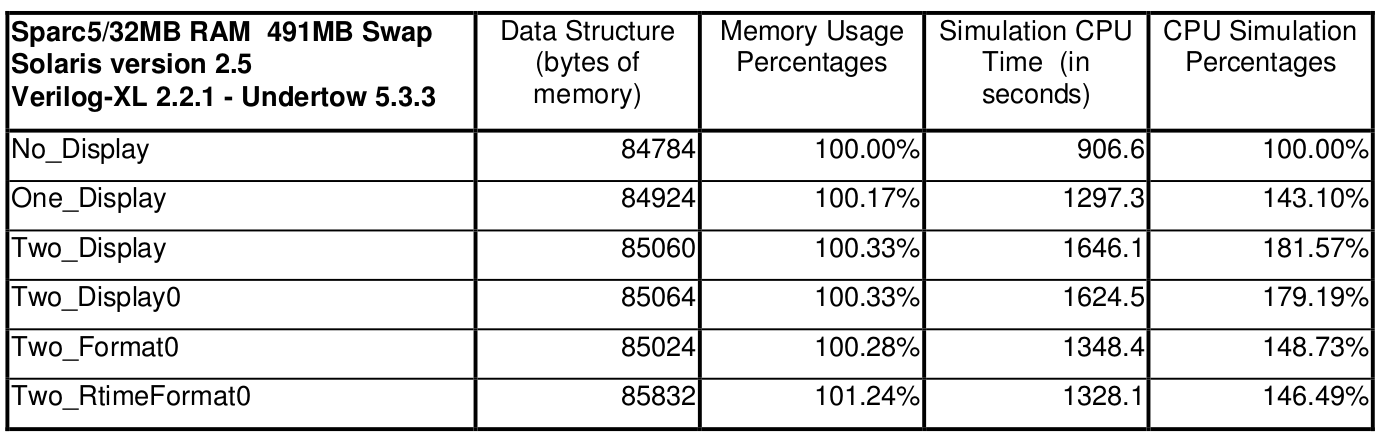
##### Timescale仿真精度

仿真精度越高，仿真时间越长。



##### 添加display打印信息

添加display打印信息越多，仿真越慢



## HLS硬件加速篇

讲讲用HLS如何实现硬件加速。

## 技术随想脑洞

### 灵活标准IP库制定

Tips1：通过（高级语言/GUI）生成灵活的标准ip库。  
 特别说明：这个ip库可以根据专用应用场景，进行定制化使用，目标是节省资源，干更多的事儿。

小样1：生成axi 控制器，灵活可配的那种。

小样2：生成以太网控制器，比如eth\_mux，二进一处，三进一出，多进一出等等;

小样3：以太网报文解包器，比如可以通过界面指定修改字段，由python/C++等编写代码来自动生成;

小样4：crc生成器，这个一直都有标准界面，设置好后，直接生成，比纯手工写代码强好多倍。

小样5：ddr通用控制器。

### 遗传进化的思想进行优化《失控这本书的随想》

软件优化和硬件优化一样，

目前我们写的代码都是人类的思维方式，有没有想过，同样的硬件配置条件下，你的代码在完成既定功能时，性能不是最优的，考虑通过建立生态环境 ，通过随机变异的方式，让机器自己学习，产生特定的机器码，它可能不是最优的，但他是比人类写的代码要厉害的代码。

这个事儿要是做成了，那么 人类社会的奇点就到来了。

## UVM验证篇

## 技术债问题

### git

使用git管理版本，这个要搭建起来。

### 仿真器平台脚本开发

要求实现多种功能

1. 支持无UVM，无IP CORE最简单仿真
2. 支持UVM，无IP CORE仿真
3. 支持UVM，有IP CORE仿真
4. 支持HLS，生成IPCORE直接内嵌到仿真环境的仿真
5. 支持Modelsim+Verdi联合仿真
6. 支持仅仅ModelSim仿真
7. 支持VCS仿真等等
8. 支持vivadosim仿真
9. 支持Linux环境仿真，platform
10. 支持windows环境仿真

I.仿真工具必须使用超级简单，特别容易上手使用。

J.做最简单的例子，直接就能上手的那种

K.要是能够run

HLS的话，有他自己的一套开发东西和环境。

vivado编译的话，需要添加cpu等信息，也不太方便

Sdk开发的话，源代码不一样。

### xfDNN代码啃读工作

##### 模板标题4

见下文

###### 模板标题5

见下文

模板标题6

见下文

模板标题6

见下文

模板标题7

见下文

对于WPS来说，可以从”文件”-->”页面设置”-->”文档网络”，然后选择“无网络”