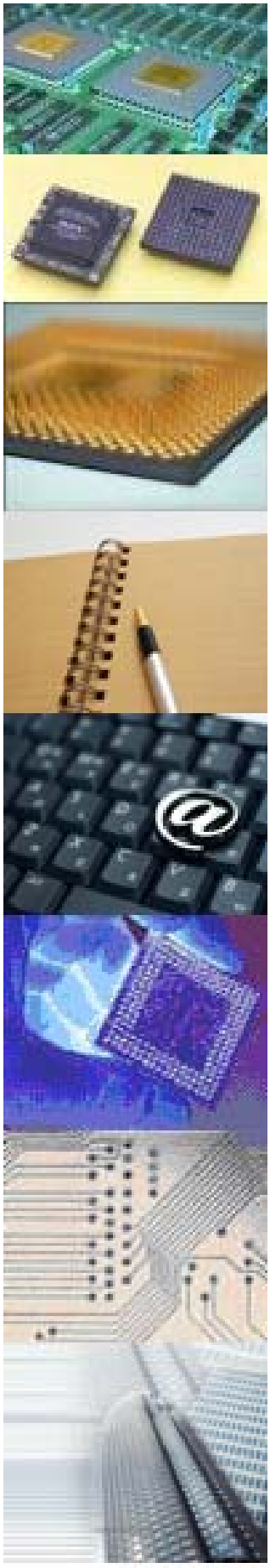


EDACN技术月刊

逻辑设计与集成电路

MAY 2005

VOL.1 No.2

售价 EDA：1.00

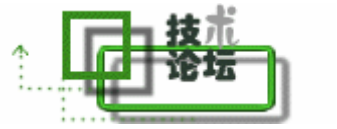
Make EDA Serve You !

http://www.edacn.net

★面向新手　★实用为主　★论坛刊物

版权说明

EDACN 论坛
EDA 先锋工作室



Make EDA Serve You !

欢迎登陆EDACN论坛：http://www.edacn.net

刊名： EDACN 技术月刊（逻辑设计与集成电路）
主办： EDACN 论坛技术月刊编辑部
EDA 先锋工作室
发布日期： 2005 年 5 月 1 日
卷期编号： 第 1 卷　总第 2 期
定价： 1 EDA 元
总编： mail007
执行总编： hyzhangwang
编辑部： ahan　myname
（排名不分先后） wenxinniwo　wuxinhua1982
yangfeng
本期特约作者： itator　westor
编辑部信箱： monthly@edacn.net

目录

主题讨论

专题讨论

原创文章

- **testbench_vantage** (itator)
- 时钟设计的革命 (westor)

业界新闻

- Cyclone II FPGA和Nios II嵌入式处理器具有低成本性能优势
- 赛灵思新版开发工具支持更多FPGA系列器件

资源共享

- **HDL编码风格与编码指南**
- 华为内部资料（硬件工程师手册）
- **148 个verilog hdl小程序（有很多testbench）——适合初学者**
- **我是渔鱼,为人民服务!清华的VHDL讲义!!!!**

FPGA 设计的指导性原则（连载之二）

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=9&topic=131
（主题编号：0900131；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题内容由 westor 原创并在 EDACN 论坛里陆续发布，其主要内容构成后来经典网文《FPGA 设计指导原则》的雏形。由于本主题内容非常丰富，限于篇幅，将采用连载方式刊出。此文是连载之二，关于 FPGA 的设计思想与操作技巧，主要内容有乒乓操作、串并转换、流水线操作和数据接口的同步方法四个部分组成。

前面 3 个小节，简单介绍一些 FPGA 的设计思想与操作技巧。FPGA 的设计思想和技巧多种多样，篇幅所限，我们不可能将日常工作中涉及的所有设计思想和技巧都一一讨论，在此仅仅挑选了 3 个有代表性的方法加以简要介绍，希望读者能够通过日常工作实践，总结出更多的设计思想与技巧。

1.5 基本设计思想与技巧之一：乒乓操作

“乒乓操作”是一个常常应用于数据流控制的处理技巧。典型的乒乓操作方法如图 0900131－1 所示。

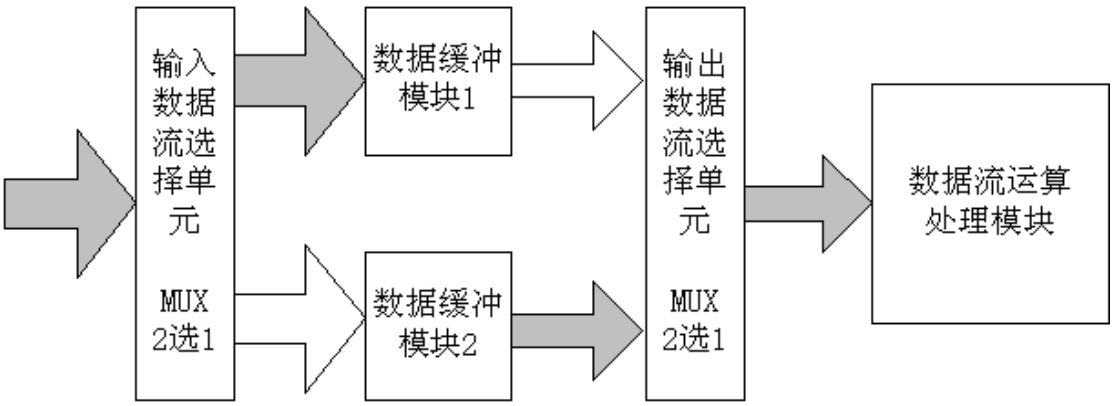


图 0900131－1　乒乓操作示意图

乒乓操作的处理流程描述如下：输入数据流通过“输入数据选择单元”，等时地将数据流等时分配到两个数据缓冲区。数据缓冲模块可以为任何存储模块，比较常用的存储单元为双口 RAM（DPRAM）、单口 RAM(SPRAM)、FIFO 等。在第一个缓冲周期，将输入的数据流缓存到“数据缓冲模块 1”。在第 2 个缓冲周期，通过“输入数据选择单元”的切换，将输入的数据流缓存到“数据缓冲模块 2”，与此同时，将“数据缓冲模块 1”缓存的第 1 个周期的数据通过“输入数据选择单元”的选择，送到“数据流运算处理模块”被运算处理。在第 3 个缓冲周期，通过“输入数据选择单元”的再次切换，将输入的数据流缓存到“数据缓冲模块 1”，与此同时，将“数据缓冲模块 2”缓存的第 2 个周期的数据通过“输入数据选择单元”的切换，送到“数据流运算处理模块”被运算处理。如此循环，周而复始。

乒乓操作的最大特点是，通过“输入数据选择单元”和“输出数据选择单元”按节拍、相互配合的切换，将经过缓冲的数据流没有时间停顿的送到“数据流运算处理模块”，被运

算与处理。把乒乓操作模块当作一个整体，站在这个模块的两端看数据，输入数据流和输出数据流都是连续不断的，没有任何停顿，因此非常适合对数据流进行流水线式处理。所以乒乓操作常常并应用于流水线式算法，完成数据的无缝缓冲与处理。

乒乓操作的第二个优点是可以节约缓冲区空间。比如在 WCDMA 基带应用中，1 帧（Frame）是由 15 个时隙（Slot）组成的，有时需要将 1 整帧的数据延时一个时隙后处理，比较直接的办法是将这帧数据缓存起来，然后延时 1 个时隙，进行处理。这时缓冲区的长度是 1 整帧数据长，假设数据速率是 3.84Mb/s，1 帧长 10ms，则此时需要缓冲区长度是 38400bit。如果采用乒乓操作，只需定义两个能缓冲 1 个 slot 数据的 RAM（单口 RAM 即可），当向一块 RAM 写数据的时候，从另一块 RAM 读数据，然后送到处理单元处理，此时，每块 RAM 的容量仅需 2560bit 即可。2 块 RAM 加起来也只有 5120bit 的容量。

另外巧妙的运用乒乓操作，还可以达到用低速模块处理高速数据流的效果。如图 0900131－2 所示，数据缓冲模块采用了双口 RAM，并在 DPRAM 后引入了一级数据预处理模块，这个数据预处理可以根据需要是各种数据运算，比如在 WCDMA 设计中，对输入数据流的解扩、解扰、去旋转等。假设端口 A 的输入数据流的速率为 100Mb/s，乒乓操作的缓冲周期是 10ms。我们下面一起分析一下各个节点端口的数据速率。

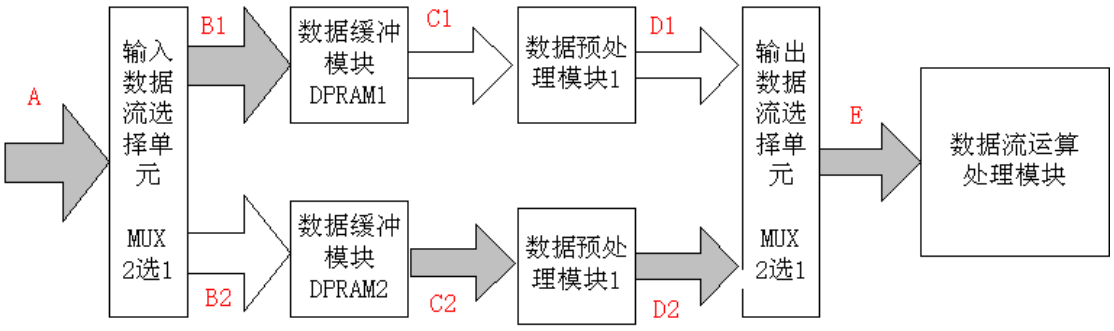


图 0900131－2　利用乒乓操作降低数据速率

输入数据流 A 端口处数据速率为 100Mb/s，在第 1 个缓冲周期 10ms 内，通过“输入数据选择单元”，从 B1 到达 DPRAM1。B1 的数据速率也是 100Mb/s，在 10ms 内，DPRAM1 要写入 1Mb 数据。同理在第 2 个 10ms，数据流被切换到 DPRAM2，端口 B2 的数据速率也是 100Mb/s，DPRAM2 在第 2 个 10ms 被写入 1Mb 数据。周而复始，在第 3 个 10ms，数据流又切换到 DPRAM1，DPRAM1 被写入 1Mb 数据。

仔细分析一下，就会发现到第 3 个缓冲周期时，留给 DPRAM1 读取数据并送到“数据预处理模块 1”的时间一共是 20Ms。有的同事比较困惑于 DPRAM1 的读数时间为什么是 20ms，其实这一点完全可以实现。首先在第 2 个缓冲周期，向 DPRAM2 写数据的 10ms 内，DPRAM1 可以进行读操作；另外在第 1 个缓冲周期的第 5ms 起（绝对时间为 5ms 时刻），DPRAM1 就可以边向 500K 以后的地址写数，边从地址 0 读数，到达 10ms 时，DPRAM1 刚好写完了 1Mb 数据，并且读了 500K 数据，这个缓冲时间内 DPRAM1 读了 5ms 的时间；另外在第 3 个缓冲周期的第 5ms 起（绝对时间为 35ms 时刻），同理可以边向 500K 以后的地址写数，边从地址 0 读数，又读取了 5 个 ms，所以截止 DPRAM1 第一个周期存入的数据被完全覆盖以前，DPRAM1 最多可以读取了 20ms 时间，而所需读取的数据为 1Mb，所以端口 C1 的数据速率为：1Mb/20ms=50Mb/s。因此“数据预处理模块 1”的最低数据吞吐能力也仅仅要求为 50Mb/s。同理“数据预处理模块 2”的最低数据吞吐能力也仅仅要求为 50Mb/s。换言之，通过

乒乓操作，“数据预处理模块”的时序压力减轻了，所要求的数据处理速率仅仅为输入数据速率的 1/2。

通过乒乓操作实现低速模块处理高速数据的实质是：通过 DPRAM 这种缓存单元，实现了数据流的串并转换，并行用“数据预处理模块 1”和“数据预处理模块 2”处理分流的数据，是面积与速度互换原则的有一个体现！

1.6 基本设计思想与技巧之二：串并转换

串并转换是 FPGA 设计的一个重要技巧，从小的着眼点讲，它是数据流处理的常用手段，从大的着眼点将它是面积与速度互换思想的直接体现。串并转换的实现方法多种多样，根据数据的排序和数量的要求，可以选用寄存器、RAM 等实现。前面在乒乓操作图 0900131－2 的举例，就是通过 DPRAM 实现了数据流的串并转换，而且由于使用了 DPRAM，数据的缓冲区可以开的很大。对于数量比较小的设计可以采用寄存器完成串并转换。如无特殊需求，应该用同步时序设计完成串并之间的转换。比如数据从串行到并行，数据排列顺序是高位在前，可以用下面的编码实现：

```
prl_temp <= {prl_temp,srl_in};
```

其中，prl_temp 是并行输出缓存寄存器，srl_in 是串行数据输入。

对于排列顺序有规定的串并转换，可以用 case 语句判断实现。对于复杂的串并转换，还可以用状态机实现。串并转换的方法总的来说比较简单，在此不做更多的解释。

1.7 基本设计思想与技巧之三：流水线操作

流水线处理是高速设计中的一个常用设计手段。如果某个设计的处理流程分为若干步骤，而且整个数据处理是“单流向”的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入则可以考虑采用流水线设计方法提高系统的工作频率。

流水线设计的结构示意图如图 0900131－3 所示：

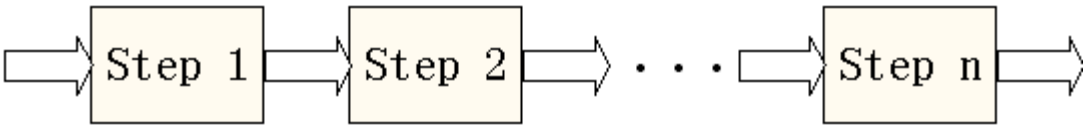


图 0900131－3　流水线设计的结构示意图

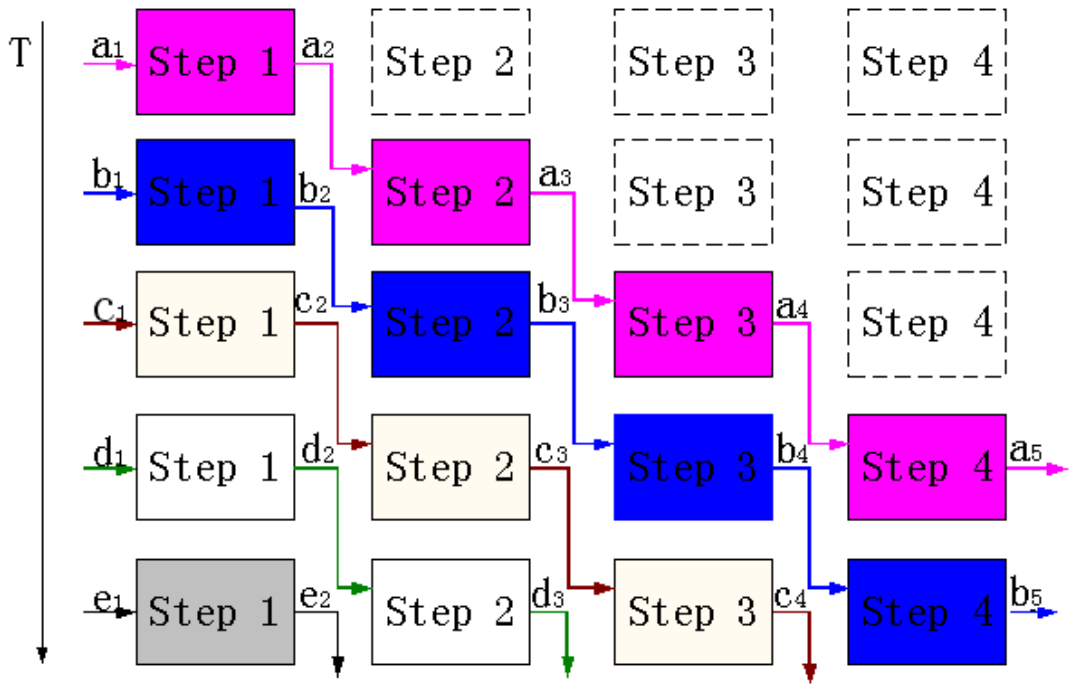


图 0900131－4　流水线设计时序示意图

其基本结构为：将适当划分的 n 个操作步骤单流向串联起来。流水线操作的最大特点和要求是，数据流在各个步骤的处理，从时间上看是连续的，如果将每个操作步骤简化假设为通过一个 D 触发器（就是用寄存器打一个节拍），那么流水线操作就类似一个移位寄存器组，数据流依次流经 D 触发器，完成每个步骤的操作。流水线设计时序示意图如图 0900131－4 所示：

流水线设计的一个关键在于，整个设计时序的合理安排。要求每个操作步骤的划分合理。如果前级操作时间恰好等于后级的操作时间，设计最为简单，前级的输出直接汇入后级的输入即可。如果前级操作时间大于后级的操作时间，则需要对前级的输出数据适当缓存，才能汇入后级的输入端。如果前级操作时间恰好小于后级的操作时间，则必须通过复制逻辑，将数据流分流，或者在前级对数据采用存储、后处理方式，否则会造成后级数据溢出。
在 WCDMA 设计中经常使用到流水线处理的方法，如 RAKE 接收机、搜索器、前导捕获等。流水线处理方式之所以频率较高，是因为复制了处理模块，它是面积换取速度思想的又一种具体体现。

1.8 基本设计思想与技巧之四：数据接口的同步方法

数据接口的同步在是 FPGA/CPLD 设计的一个常见问题，也是一个重点和难点。很多设计工作不稳定都是源于数据接口的同步有问题。

在电路图设计阶段，有一些设计者养成了手工加入 BUFT 或者非门调整数据延迟，从而保证本级模块的时钟对上级模块数据的建立、保持时间的要求。还有一些设计者为了有稳定的采样，生成了很多相差 90 度的时钟信号，时而用正沿打一下数据，时而用负沿打一下数据，用以调整数据的采样位置。这两种做法都是万万取不得的。这些做法，一旦芯片更新换代，或者移植到其它器件族的芯片上，采样实现必须从新设计。而且这两种做法造成电路实现的余量不够，一旦外界条件变换（比如温度升高），采样时序就有可能完全紊乱，造成电路瘫痪。

下面简单介绍几种不同情况下的数据接口的同步方法。

• 输入、输出的延时（芯片间、PCB 布线、一些驱动接口元件的延时等）不可测，或者有可能变动，如何完成数据的同步？

对于数据的延迟不可测，或者变动，就需要建立同步机制。可以用一个同步使能，或者同步指示信号。另外使数据通过 RAM 或者 FIFO 的存取，也可以达到数据同步的目的。把数据存放在 RAM 或 FIFO 的方法如下，将上级芯片提供的数据随路时钟作为写信号，将数据写入 RAM 或者 FIFO，然后使用本级的采样时钟（一般是数据处理的主时钟），将数据读出来即可。这种做法的关键是数据写入 RAM 或者 FIFO 要可靠，如果使用同步 RAM 或者 FIFO，就要求有应该有一个与数据相对延迟关系固定的随路指示信号，这个信号可以是数据的有效指示，也可以是上级模块将数据打出来的时钟。对于慢速数据，也可以采样异步 RAM 或者 FIFO，但是这种做法不推荐。

数据是有固定格式安排的，很多重要信息在数据的起始位置。

这种情况在通信系统非常普遍，通信系统中，很多数据是按照“帧”组织的。而由于整个系统要求对时钟要求很高，常常专门设计一块时钟板完成高精度时钟的产生与驱动。而数据又是有起始位置的，如何完成数据的同步，并发现数据的“头”？

数据的同步方法完全可以采用上一点的方法，采用同步指示信号，或者使用 RAM、FIFO 缓存一下。找到数据头的方法有两种，第一种很简单，随路传输一个数据起始位置的指示信号即可，对于有些系统，特别是异步系统，则常常在数据中插入一段同步码（比如训练序列），接收端通过状态机检测到同步码后，就能发现数据的“头”了，这种做法叫做“盲检测”。

• 上级数据和本级时钟是异步的，也就是说上级芯片或模块和本级芯片或模块的时钟是异步时钟域的。

前面在输入数据同步化中已经简单介绍了一个原则：如果输入数据的节拍和本级芯片的处理时钟同频，可以直接用本级芯片的主时钟对输入数据寄存器采样，完成输入数据的同步化；如果输入数据和本级芯片的处理时钟是异步的，特别是频率不匹配的时候，则只是要用处理时钟对输入数据做两次寄存器采样，才能完成输入数据的同步化。需要说明的是用寄存器对异步时钟域的数据进行两次采样，其作用是有效的防止了亚稳态（数据状态不稳定）的传播，使后级电路处理的数据都是有效电平。但是这种做法并不能保证两级寄存器采样后的数据是正确的电平，这种方式处理，一般都会产生一定数量的错误电平数据。所以仅仅适用于对少量错误不敏感的功能单元。

为了避免异步时钟域产生错误的采样电平，一般使用 RAM、FIFO 缓存的方法完成异步时钟域的数据转换。最常用的缓存单元是 DPRAM，在输入端口使用上级时钟写数据，在输出端口使用本级时钟读数据，就非常方便地完成了异步时钟域之间的数据交换。

典型的 FPGA 设计流程

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=7&topic=5231
（主题编号：0705231；编辑整理：yangfeng；校对改正：myname；统稿：wenxinniwo）

编者提示：以下简单介绍了FPGA设计的一般流程，有两个版本，仅供参考。

版本1：
1、设计输入
1)设计的行为或结构描述。
2)典型文本输入工具有UltraEdit-32和Editplus.exe.。
3)典型图形化输入工具-Mentor的Renoir。
4)我认为UltraEdit-32最佳。
2、代码调试
1)对设计输入的文件做代码调试，语法检查。
2)典型工具为Debussy。
3、前仿真
1)功能仿真
2)验证逻辑模型(没有使用时间延迟)。
3)典型工具有Mentor公司的ModelSim、Synopsys公司的VCS和VSS、Aldec公司的Active、Cadense公司的NC。
4)我认为做功能仿真Synopsys公司的VCS和VSS速度最快，并且调试器最好用，Mentor公司的ModelSim对于读写文件速度最快，波形窗口比较好用。
4、综合
1)把设计翻译成原始的目标工艺
2)最优化
3)合适的面积要求和性能要求
4)典型工具有Mentor公司的LeonardoSpectrum、Synopsys公司的DC、Synplicity公司的Synplify。
5)推荐初学者使用Mentor公司的LeonardoSpectrum，由于它在只作简单约束综合后的速度和面积最优，如果你对综合工具比较了解，可以使用Synplicity公司的Synplify。
5、布局和布线
1)映射设计到目标工艺里指定位置
2)指定的布线资源应被使用
3)由于PLD市场目前只剩下Altera，Xilinx，Lattice，Actel，QuickLogic，Atmel六家公司，其中前5家为专业PLD公司，并且前3家几乎占有了90%的市场份额，而我们一般使用Altera，Xilinx公司的PLD居多，所以典型布局和布线的工具为Altera公司的Quartus II和Maxplus II、Xilinx公司的ISE和Foudation。
4)Maxplus II和Foudation分别为Altera公司和Xilinx公司的第一代产品，所以布局布线一般使用Quartus II和ISE。
6、后仿真
1)时序仿真

2)验证设计一旦编程或配置将能在目标工艺里工作（使用时间延迟）。
3)所用工具同前仿真所用软件。
7、时序分析
4)一般借助布局布线工具自带的时序分析工具，也可以使用Synopsys公司的 PrimeTime软件和Mentor Graphics公司的Tau timing analysis软件。
8、验证合乎性能规范
1)验证合乎性能规范，如果不满足，回到第一步。
9、版图设计
1)验证版版图设计。
2)在板编程和测试器件

版本 2：

1、 使用 modelsim 进行功能仿真，导入源程序和 testbench 进行仿真，并保存波形文件（.wlf）。
2、使用 synplify pro 对硬件描述语言编译并生成 netlist。综合前要注意对器件的选择，方法是在 project－>implementation option 中对要下载的器件和网表的生成情况进行选择。综合后的网表有两种： RTL 级网表和门级网表（gate netlist），通过对网表的分析可以对设计的实现方式有初步的了解，并分析其中的错误和不合理的地方，另外还可以对关键路径的 delay 和 slack 进行分析。使用 synplify pro 要先新建工程，注意修改工作目录，然后添加所要编译的文件，要注意 top 文件要最后一个添加，这样才可以保证生成的文件是以 top 文件来命名的。
3、使用 quartus II 根据 netlist 进行布线，并进行时序分析。在使用 quartus II 前要做一些必要的设置，在 assignments－>eda tools setting 中的 simulation 中选择 modelsim，并选择选项 run this tools automatically after compilation。如果没有提前做这些设置，可以 quartus 做完编译布线后，做同样的设置，然后运行 EDA netlist writer 和 eda simulation tool。在使用 synplify pro 得到满意的 netlist 后，可以在 synplify pro 中通过 option－> quartus II 直接调用 quartus II，quartus II 对 synplify pro 生成的.vqm 文件进行编译，布线。然后根据设计要求进行时序分析和引脚调整。
4、使用 modelsim 进行布线后仿真。由于 quartus II 提前做了设置，因此在编译布线完成后，会在工作目录下生成 modelsim 仿真所需要的文件和库（modelsim_work），在 modelsim 中将产生的文件和库所在的文件夹设置为当前目录，modelsim_work 库会自动导入，新建工程会提示所使用的 modelsim.ini 文件，应使用 quartus 生成的，然后导入文件（包括 testbench），进行编译，仿真的时候在 library 中添加 modelsim_work 库，在 sdf 选项中可以添加 quartus 生成的延迟信息文件.sdo，注意作用域的选择，如果 testbench 中调用被测试模块的语句是 send3a tb，那么作用域应该写 tb，在 option 选择中可以选择是否看代码覆盖率。另外，还可以将布线后的仿真结果与功能仿真的结果进行对比。下图就是小型 Soc 中 send3a 模块前后仿真的对比图

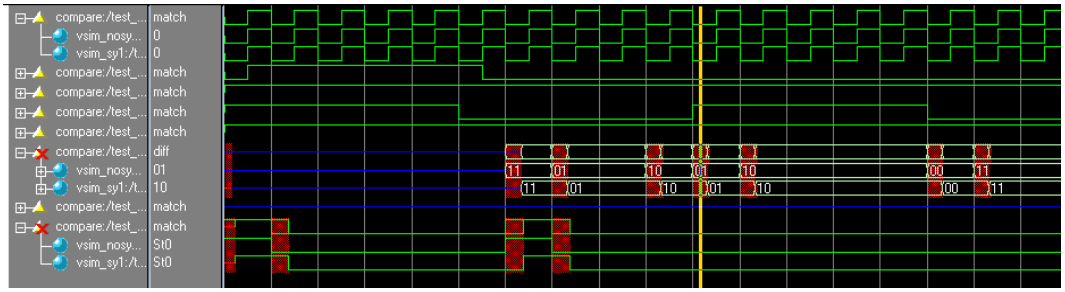


图 0705231-1

从图中可以看出，除了有一定的延迟外，输出波形不变。

5、将 quartus 的波形转化成 testbench 的方法：画好波形后，通过 file－>export 可以将波形输出到 quatus 的工作目录，verilog 语言扩展名为.vt，修改为.v 后可以在 modelsim 中使用，需要说明的是如果波形中包括输出端口的话，输出的 testbench 包含三个模块，一般情况下，只需将输入波形画好后，输出到 testbench 就可以了。

大型复杂 FPGA 设计推荐设计方式——Modular Design

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=13&topic=1
（主题编号：1300001；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

westor：

1、简介

随着可编程技术的发展，FPGA/CPLD 被广泛应用于电子设计的各个领域。越来越多复杂系统的核心电路利用 FPGA 设计完成，这些复杂系统经常需要使用百万门以上的大规模 FPGA 来设计。另一方面，为了对市场需求做出最迅速的反映，要求这些电子产品的设计周期尽量缩短。只有以第一时间推出成熟稳定的产品，才能获得更大的市场份额。于是一方面需要百万门以上的大规模 FPGA 以满足设计需要，另一方面需要在最短的时间内高质量地完成设计以满足市场需求，这两者出现了矛盾。

解决上述矛盾的出路是采用模块化设计方法，投入更多的人力，进行并行工作、协同设计。FPGA/CPLD 模块化设计方法的基本思想是：将大规模复杂系统按照一定规则划分成若干模块，然后对每个模块进行设计输入、综合，并将实现结果约束在预先设置好的区域内，最后将所有模块的实现结果有机地组织起来，完成整个系统的设计。

Xilinx ISE 5.x 系列中的模块化设计方法叫做“Modular Design”。在 ISE 5.2 以前的版本中，Modular Design 是需要单独付费购买的工具包，在 ISE 5.2 版本中，Modular Design 已经直接作为标准工具内嵌到 ISE 软件中，安装 ISE 后即可直接使用，大大提高了 Modular Design 的用户数量，提高了 Xilinx Modular Design 工具的知名度。Modular Design 的最显著优势有两个：一是协同设计，即所有设计小组成员可以在最大程度上互不干扰地设计自己的子模块，从而加速了项目进度；二是在调试、更改某个有缺陷的子模块时，并不会影响到其它模块的实现结果，从而保证了设计的稳定性与可靠性。

Modular Design 支持的器件族有 Virtex-II Pro、Virtex/-II/-E 和 Spartan-II/-IIE。

2、模块化设计方法的设计流程

一个完整的 FPGA/CPLD 设计流程包括电路设计与输入、功能仿真、综合、综合后仿真、实现、布线后仿真和下板调试等主要步骤。Modular Design 的设计流程也遵循上述 FPGA/CPLD 设计流程。只是在设计输入、综合、实现等主要操作步骤上有一些特殊的要求和操作方法。

Modular Design 设计流程大致可以分为以下步骤：

Modular Design 的设计输入与综合步骤

Modular Design 设计输入与综合主要完成以下两个方面的设计：

顶层模块的设计

项目管理者需要完成顶层模块的设计输入与综合，为进行 Modular Design 实现阶段的第一步——初始预算阶段（Initial Budgeting Phase）做准备。

子模块的设计

每个项目成员相对独立地并行完成各自子模块的设计输入和综合，为进行 Modular Design 实现阶段的第二步——子模块的激活模式实现（Active Module Implementation）做准备。

Modular Design 的实现步骤

Modular Design 的实现步骤是整个 Modular Design 设计流程中最重要、最特殊的，它包

括以下 3 个步骤：

(1) 初始预算（Initial Budgeting）

在该阶段，项目管理者对顶层模块完成顶层约束。顶层约束包括对整个设计的全局区域约束、对每个子模块的规模和区域的约束、对每个模块的输入/输出约束、对整个设计的时序约束等内容。仅需要对设计的顶层模块进行初始预算估计，对子模块则没有必要。

1、子模块的激活模式实现（Active Module Implementation）

2、在该阶段，每个项目成员并行完成各自子模块的实现。

3、模块的最后合并（Final Assembly）

在该阶段，项目管理者将设计的顶层和子模块的激活模式实现结果有机地组织起来，完成整个设计的实现步骤。

在进行大规模复杂系统的设计时，采用 Modular Design 设计方法能在最大程度上发挥项目组所有工程师的作用，提高设计效率，缩短项目开发周期。但是 Modular Design 设计方法对项目管理者提出了更高的要求：项目管理者应该合理地划分模块，对项目进度做出很好的权衡，使所有子模块设计任务并行完成，从而不耽误模块的最后合并（Final Assembly）阶段的工作进度。

Modular Design 模块划分的基本原则为：子模块功能相对独立，模块内部联系尽量紧密，而模块间的连接尽量简单。所以对于那些难以满足模块划分准则的具有强内部关联的复杂设计，并不适合采用 Modular Design 设计方法。

Coding Style 与综合前后仿真

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=7&topic=73
（主题编号：0700073；编辑整理：yangfeng；校对改正：myname；统稿：wenxinniwo）

编者提示：大家进行 FPGA 设计验证的时候经常做功能仿真以及布局布线后仿真，但对综合后仿真大都不太重视，那么有没有必要做综合后仿真呢？有哪些原因可能引起综合前后仿真结果不同呢？希望以下的讨论对您有所启发。

Westor：
有很多朋友认为只需要做功能仿真（所谓的前仿真），和布局布线后仿真（所谓的后仿真），这样在功能仿真中重点验证逻辑功能，在布局布线后仿真中重点验证时序，可以省略综合后仿真。您的观点呢？

A 下面是一段综合后仿真的定义：
综合完成后需要检查综合结果是否与原设计一致，需要做综合后仿真。在仿真时，把综合生成的延时文件反标到综合仿真文件中去，可估计门延时带来的影响。综合后仿真虽然比功能仿真精确一些，但是只能估计门延时，而不能估计线延时，仿真结果与布线后的实际情况还有一定的差距，并不十分准确。这种仿真的主要目的在于检查综合器的综合结果是否与设计输入一致。

B 这里我想结合一个例子说明这个问题：
前面已经反复强调，综合后仿真的真正目的并不在于验证门延时，而是验证综合结果是否与自己的设计意图一致。首先有一点要明确，每个综合器的综合效果是不同的，我这里说的不同的重点并不在于优化程度等因素，而是说综合器对一些代码的理解是不同的！特别是一些所谓的“歧意代码”！如果您的 Coding style 不是很好，或者代码不算标准，那么就可能造成不同的综合器综合出的逻辑相差很远，这时就要通过做综合后仿真来验证“综合结果是否与自己的设计意图一致。”下面这个例子摘自我的一篇劣作，一个学员对自己的代码用 XST，和 Synplify 两种综合工具综合，出来的结果完全不一致，他认为 Synplify 综合的结果是正确的，XST 综合结果是错误的。我通过分析认为本质愿意是他的代码属于“歧意代码”，是他的 Coding style 造成了这个问题。 下面将附上该文章的前面部分，并不下结论，如有兴趣可自行综合、仿真。

C 为什么 XST 与 Synplify 的综合结果不一样？
一位同学反映，他设计的状态机使用 Synplify 综合和 XST（Xilinx Synthesis Technology，是 Xilinx ISE 内嵌的综合工具）综合的结果不一致。对 Synplify 的综合结果布局布线后上板调试完全正确；而对 XST 的综合结果布局布线后上板调试发现了错误。所以他认为 XST 的综合结果有误，并询问为什么 XST 会综合出错误的结果。在帮助他解决这个疑问后，发现其中涉及的许多问题很具代表性，所以和大家一起分析一下这个疑问，希望对朋友们的设计有参考价值。下面是这位同学设计的状态机源代码：

```
module RxState   (Reset_b,RxClk,Rx4bit,RxDV,CurrentState);
//port declare
input Reset_b,RxClk,RxDV;
```

```
input [3:0] Rx4bit;
output [2:0] CurrentState;
// wire declare
wire   RXDEQD = Rx4bit == 4'hd;
wire   RXDEQ5 = Rx4bit == 4'h5;
parameter [2:0]
 IDLE        = 3'H0,
 PREAMBLE   = 3'H1,
 SFD         = 3'H2,
 DATA0       = 3'H3,
 DATA1       = 3'H4,
 DROP        = 3'H5;
// Rx State machine
reg [2:0]   CurrentState,NextState;
always @( posedge RxClk or negedge Reset_b)
begin
 if(!Reset_b)
   CurrentState <=   DROP;
 else
   CurrentState <=   NextState;
end
always @( CurrentState or RxDV or RXDEQ5 or RXDEQD)
begin
 case(CurrentState)
   IDLE     : if(RxDV   )   NextState <=   PREAMBLE;
   PREAMBLE: if(RXDEQ5)   NextState <=   SFD;
   SFD       :
     begin
       if(RXDEQD)
         NextState       <=   DATA0;
     end
   DATA0     : NextState <=   DATA1;
   DATA1     :
     begin
       NextState       <=   DATA0;
         if(!RxDV)
           NextState <=   IDLE;
     end
   DROP      : if(!RxDV)     NextState <=   IDLE;
   default : NextState   <=   IDLE;
 endcase
end
endmodule
```

其设计是一个数据通讯中同步接收装置的状态机，当“Reset_b”复位后，进入“DROP”状

态；当接收指示信号“RXDV”有效后，从“IDLE”状态进入接收前缀信号状态“PREAMBLE”；当控制信号“RxDEQ5”有效后进入“SFD”状态接收一些指示关键字；当控制信号“RxDEQ”有效时，进入数据接收状态“DATA0”；接收完“DATA0”后接收“DATA1”，直到接收指示信号“RxDV”无效，返回到“IDLE”状态。

D 提示

我认为的两方面问题: 1 竞争冒险的问题。2 锁存器的使用。(该文的小结，仅供参考。)

通过该问题的解决我们引出了在逻辑设计中几个问题，有些观点需要进一步澄清，希望读者能够通过对这些问题的重视，提高设计质量与效率。

1、 综合后仿真的概念与作用。

综合后仿真的最主要作用在于验证综合器的综合结果是否与设计意图一致。仿真时，把综合生成的延时文件反标到综合仿真文件中去，可估计逻辑门延时带来的影响。该仿真只能估计门延时，而不能估计线延时，仿真结果与布线后的实际情况还有一定的差距，并不十分准确。有些设计者认为综合器是永远可靠的，综合出的电路应该与设计意图一致，综合后仿真与功能仿真结果一致。这种观点是错误的，只要当设计者有相当的代码功底，代码风格合理，不出现让综合器误解的描述时，综合结果才会和设计意图完全一致。在一般情况下建议不要省略此步骤。

2、代码设计风格对设计的影响。

在学习逻辑设计之初，相信老师们总是强调代码风格的重要性。有些设计者认为现在综合器的优化功能越来越强，对大部分设计都可以优化。这种观点也是不对的。综合器的优化结果主要是依靠设计者的代码风格，好的代码风格的优化结果更上一层楼；不良代码风格的优化结果却常常南辕北辙。本例这位同学的所有问题都源于其代码风格不尽合理。问题的本质并不在于综合器。

3、 状态机设计方法。

状态机的设计方法很多，描述方法不一而同，目前如 StateCAD 等状态机辅助设计软件更是给用户提供了很多的帮助。这里需要强调的是在状态机的设计中使用条件判断语句一定要慎重，尽量使用“if…else”这种完整的判断结构。有时判断语句使用不合理，会造成综合器对设计的误解，不同的综合器的默认综合结果会因为是否使用了 Latch，是否使用了带有优先级的编译码器等结构而不同。

4、 一些有用的辅助分析工具。

最后提一下辅助分析工具的使用。FPGA/CPLD 等可编程逻辑器件的长足发展在一定程度上就是因为其 EDA 辅助设计工具的智能化、易学易用、功能强大等优势。在本例分析综合结果的过程中，并未直接分析综合器的输出网表，而是使用了 RTL 视图进行分析，直观、快捷，提高了分析效率。

关于《为什么XST与Synplify的综合结果不一样？》的几个问题：

1、关于latch

从程序看，我觉得文中的程序是一个网口收帧的程序，像这样的程序，完全可以通过全同步电路完成，不需要使用组合进程，我个人认为组合进程是最难把握的，最容易出现纰漏。源程序综合后出现了latch，大家都很忌讳latch，但有的时候latch无法避免，比如说一个状态机有1和2两个状态，要求在1状态时条件满足就进入2，否则就还是1。那么使用组合进程的话latch就不可避免，否则结果可能就不对了。

2、关于敏感信号列表

很多综合器根本就不睬敏感信号列表，写不写综合结果都一样，顶多给个warning，这

样的话许多组合进程的程序综合后的结果和设想的天差地别，但是假如使用的是时序电路的话由于时钟的关系结果还会和设想的一致

3、关于冒险竞争

文中有一段程序被作者认为是冒险竞争，但是我试了一下（XST和Synplify），并没有出现这样的情况，signal总是在process结束的时候被统一赋值的，而且假如signal有多个赋值语句的话是以最后一句为准的，所以我认为不会有冒险竞争

westor：

1、latch是应该避免，而且从代码看出，这个学员本身还是希望用同步时序实现的，关键是没有认真对待if，else这个语句，其实往往越简单的东西，越容易发生问题。比如if，if else if，等等结构，还是要仔细想一下，特别要和纯软件区分开。

2、“很多综合器根本就不睬敏感信号列表，写不写综合结果都一样，顶多给个warning”，这个观点很危险！各个综合器，特别是综合ASIC电路的综合器，和擅长综合FPGA的综合器还是有一些差别的。一般的看法是：综合器大都默认凡是在判断条件和输入出现的变量都自动加入敏感表，如果用户不加，会报错。所以希望在敏感表上做文章而达到某些用法的用户，要特别小心。另外westor特别反对忽视“Warning”，应该每个Warning都搞清楚，往往Warning是和代码风格和所谓的“歧意电路”有一定联系的。

3。竞争冒险是比较明显的，在前仿真观察竞争冒险，在测试激励的设计上要有一定的技巧。为了方便检验，您可以做一下修改前后的综合后仿真，或者直接在布局布线后仿真，可以比较明显地看到区别。

yangfeng：

可能引起综合前后仿真不一致的情况（翻译自“Verilog Coding Style for Efficient Digital Design”，详细内容请参见原作）：

1、在给一个变量赋值前读一个变量－这将导致前仿真和综合后仿真的结果不一致

例如：

```
input a,b;
reg q,z;
 always @(a or b or z)
  begin
     q=z;
     z=a|b;
  end
```

前仿真的时候，q的值是在上一次电平触发时刻z被赋的值，而不是z当前的值。

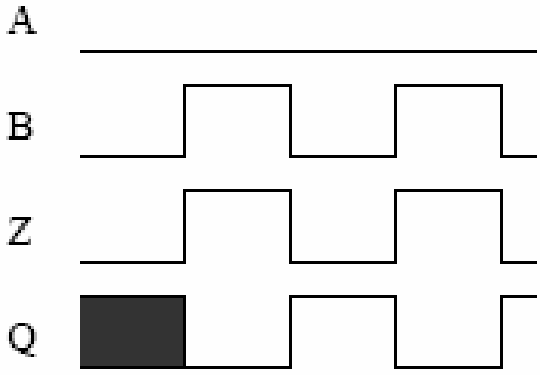


图0700073-1

上述代码综合后，只是综合成一个或门，所以综合后仿真的时候，q的值就是a跟b相或：

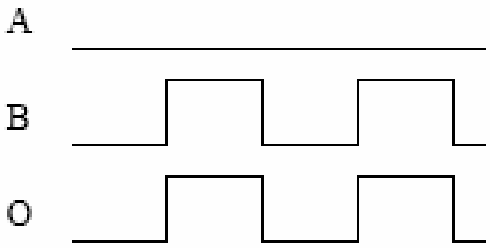


图0700073-2

这就导致了前仿真和综合后仿真的结果不一致，只要先赋值，然后读取，就不会有问题：

```
input a,b;
reg q,z;
 always @(a or b or z)
  begin
     z=a|b;
     q=z;
  end
```

2、当组合逻辑的always块中的敏感列表不正确（不完整或者有多余信号）时，会导致综合前后仿真的结果不一致。

综合工具把所有always块的敏感列表中没有关键字posedge或者negedge的当作组合逻辑。在一个组合逻辑块中，综合后的结果取决于块内的语句，而并不由敏感列表决定。综合工具会比较块内的语句与敏感列表，指出可能存在的综合前后仿真不一致的情况。在敏感列表中添加多余的信号只会增加仿真的允许时间，对结果不会有影响。另一方面，敏感列表如果不完整，那么所缺少的信号变化时将不会触发块内语句执行，但综合工具会仍然产生敏感列表完整情况下的综合结果。这就会导致综合前后仿真的结果不一致。

比较下列代码：

```
module test1 (z,a,b,c,d);
input a, b, c, d;
output z;
wire a, b, c, d;
reg z, temp1, temp2;
always @(a or b or c or d)
begin
temp1 = a | b;
temp2 = c | d;
z = temp1 ^ temp2;
end
endmodule
module test2 (z, x, y);
output z;
input x, y;
reg z;
wire x, y;
```

```
always @(x)
z = ~(x & y);
endmodule
```

3、Full Case Parallel Case可能会引起综合前后仿真的结果不一致

使用这两条约束命令可以使得设计运行速度更快占用面积更小。但在有些情况下可能会引起前后仿真不一致的问题。

例如：

```
module example2(A, B, C, D, Q)
input A, B, C, D;
output Q;
always @(A or B or C or D)
begin
case(1'b1) // synopsys full_case parallel_case
A : Q = C;
B : Q = D;
endcase
end
```

上述代码综合后会生成如下电路：

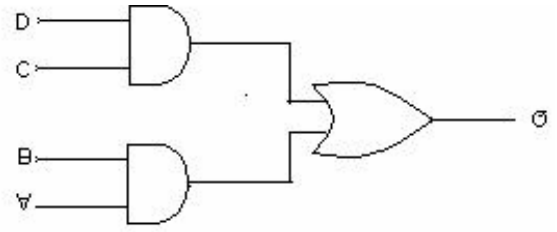


图0700073-3

综合后仿真时当A,B都为0时Q值将会输出0，而前仿真时，显然当A，B都为0时Q值将保持前一状态的值。如果去掉full_case，那么综合后将会生成一个锁存器，以锁存A，B都为0时Q的值，这样综合前后仿真结果一致，但一般来说并不建议使用锁存器。所以除非我们确定A,B不会同时为0，否则需要加入default,以便使得综合前后仿真结果一致。

当case表达式只可能有符合case列表中的一个值时，可以用Parallel Case来优化综合结果，使得综合工具不会产生冗余逻辑，减少占用的芯片面积。如果没有 Parallel Case，那么在综合前后仿真整个case语句都当作一个优先级编码器实现。而如果使用 Parallel Case，那么综合前仿真整个设计是一个优先级编码器，而综合工具会产生一个非优先级编码器（并行编码），从而导致综合前后仿真不一致。

```
module example3 (out0, out1, out2, in);
output out0, out1, out2;
input [2:0] in;
reg [2:0] out0, out1, out2;
always @(in)
begin
{out2, out1, out0} = 3'b0;
casez(in) // synopsys parallel_case
3'b1?? : out0 = 1'b1;
3'b?1? : out1 = 1'b1;
```

```
3'b??1 : out2 = 1'b1;
endcase
end
endmodule
```

4　在时序逻辑中使用非阻塞赋值，可以避免大多数的竞争冒险。

5 不要在always块中赋值语句前加延时。

6 加入translate_off/translate_on很容易出现问题。

7 给变量赋值'X'会使得综合前后仿真不一致，但有时可以利用这点产生很好的综合结果。比如在状态机设计中，可以将那些不可能出现的状态设为'X',这样综合工具将会把出现 x 的状态认为是无关状态，从而优化综合结果。

结论：为了能够成功地实现设计，设计者必须从一开始写代码就注意良好的编码风格，每一行代码都必须透彻地理解，而不要等待仿真的时候出现问题再加以解决。从设计周期的起始阶段就保持良好的设计技术编码风格将会大大减少调试的时间。

yangfeng：

功能仿真加STA能不能代替后仿真?（摘录）

功能仿真加STA（静态时序分析）并不能涵盖后仿真的作用，因为后仿真事实上有检验综合结果是否正确的作用，而功能仿真正确并不能保证综合结果和RTL设计人员的原意一样，综合器能正确综合的前提是RTL代码编写具有良好的代码风格，例如if-else语句完整、case语句完整、组合逻辑敏感列表完整，只有在这样的条件下，综合结果才有保障，否则即使功能仿真正确，综合出来的电路的功能不一定正确。对于综合过程出现的偏差，后仿真可以发现，因为后仿真实质上门级仿真，可以同时检验功能和时序是否正确，后仿真验证能保证实现结果是正确的。后仿真的不足之处在于仿真速度比较慢，因此如果不想做后仿真，对FPGA设计来说，可以做功能仿真、综合后仿真和STA，对IC设计可以做功能仿真、形式验证和STA。

另外需要注意的是,加时序约束要完整,因为STA根据时序约束做检查,如果约束不正确,STA结果就不准确.经常会出现功能验证正确而后仿真结果不正确的问题，一般是由setup time/hold time不满足等时序问题引起的，说明在综合与布局布线过程中没有进行约束或者约束条件不完全，导致STA分析结果不准确、不完全。

例如设计存在两个时钟域，一个快、一个慢，附加约束时一般要最设计整体附加较松的约束，再对局部附加较紧的约束，然后再对慢时钟和快时钟之间的路径进行约束，这一般也是较紧的约束，如果忘了最后一部分约束，那么STA会认为设计人员对这部分路径没有要求，因而不分析这部分路径，这样即使这部分路径的延迟非常大，STA也不会提示错误，但是后仿真就会出现问题。

总而言之,对FPGA设计来说,只有正确地完成综合后仿真(以保证综合结果正确)和STA,才能省略后仿真,否则后仿真仍然是必要的.

STA的意思是静态时序分析（Static Timing Analysis），做FPGA设计时是必须的一个步骤，事实上大家一般都已经做了这一步，我们在FPGA加约束、综合、布局布线后，会生成时序分析报告（在ISE中可以运行Timing Analyzer生成详细的时序报告）， 设计人员会检查时序报告、根据工具的提示找出不满足setup/hold time的路径，以及不符合约束的路径，这个过程就是STA。细致全面的STA可以保证设计的时序符合要求，只要代码robust（综合结果符合设计原意），可以省略后仿真。

数据接口设计

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=9&topic=75
（主题编号：0900075；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题由 wester 就很多网友对数据接口的设计提出的问题作了细致的回答，对大家理解不同时钟域的数据交换的设计原理和设计技巧有很大帮助。

julian_hz：

比如说，实现一个系统，系统由若干个模块组成。

两个模块之间如果有数据传送，是否需要在模块上相应地加上用于同步的使能端，用来指示当前输入数据是有效的。我目前做的模块中，输入和输出都是有相对延时的，我考虑如果不加这样的指示信号，那么输出端就可能从零初始态开始读，导致数据序列的错误。

你说，这个标识信号在设计中是必要的嘛？还是可以先不用考虑这个，等模块全部连接起来之后，考虑一个总体的延时，再加触发器校正？

westor：

这个问题很有代表性，我帮别人调了很多系统，有的规模很大，有的规模相对小些，其中很多不稳定的问题都在数据接口上。

1.首先说一个最复杂的情况，其他情况就是这种情况的子集，所需要做的措施也都包含在这个情况里面。这个情况是：

a 输入、输出的延时（芯片间、PCB 布线、一些驱动接口元件的延时等）不可测，或者变动。

b 数据是有头的，比如数据是按“帧”组织的，有帧头，很多整帧的信息在帧头的固定延时处，必须要保证数据和帧头对齐，数据才有效。这种情况在大型系统，特别是通信系统非常普遍，帧、和时钟等由时钟驱动板提供（精度高，驱动能力强），称为时钟通路，而数据由数据通路传输（比如射频到基带），数据而且有可能为了传输要做一些复用与解复用。

c 本级数据和下级数据是异步时钟域的，也就是说本级芯片和下级芯片的时钟是异步时钟域。

处理方法：

A．对于数据的延时不可测，或者变动，就需要建立同步机制。你说的用一个同步使能，或者同步指示信号是一个好办法。还可以将数据存放在 DPRAM 或者 FIFO 里面，就解决了延时问题。

B．对于有头的数据的同步非常重要，常用的方法除了 A 提到的外，还可以专门在数据通路做一个同步解数据的指示信号，特别在复用和解复用时，非常重要。另外对于异步的，可以用状态机，也就是通常说的盲检测，检测到一串码字，就是头了。用 DPRAM 和 FIFO 也是好办法，做全同步传输也行，握手信号复杂些。

C．前面说的是一些设计原理，现在专门说设计技巧。

首先坚决反对任何认为时延可测，就插入几个 buffer，搞延时，或者做一堆时钟，正沿打一下，负沿打一下。这些做法都是最忌讳的，玩玩小设计还凑合，系统一复杂，或者一旦要更换芯片（特别是换器件族和厂商），一定不行！还是那句老话，要用纯同步时序设计！大原则已经在前面帖子说了一堆了，这里说几个细节，记住了受益无穷。首先同步设计要输

入和时钟同步，也就是说，所有输入的数据，你都要用你内部的主时钟采样之，也就是我们常说的用寄存器打一下。如果数据和你的主时钟是异步时钟域的，就要打两下。另外，不要以为使用寄存器就是全同步设计了，全同步不是那么一回事儿。最后对于高速设计，一定要附件相应的约束，如周期、setup，hold 等，要保证数据采用可靠稳定。

julian_hz：

所有输入的数据，都要用内部的主时钟采样之，也就是我们常说的用寄存器打一下。如果数据和你的主时钟是异步时钟域的，就要打两下。

还有如果整个系统有很多不同频率的时钟，是不是只要都是主时钟分频出来的都是可以保证可靠的？

westor：

“异步时钟域，打两下”这是一个同步设计的经验，如果进入芯片的数据和本级芯片的时钟异步，为了保证可靠采用，避免毛刺，一般要用寄存器打两次，原因论述太麻烦，先略，以后补述。

如果是 FPGA 设计，我们的时钟树的结构一般是由主时钟分频或者倍频产生分值时钟（ASIC 因为规模和功耗，采用不同方法），而且主时钟一般用全局时钟资源驱动，用 PLL 或者 DLL 完成分频和倍频，由于大多数 FPGA 厂商在芯片内部都有专用的全局时钟资源和长线资源，而且提供 PLL 和 DLL 的硬宏，所以一般采用 FPGA 厂商提供的 IP 直接分频，注意一下时钟反馈端和全局 buffer 的使用即可。在约束的时候，比较好的约束方法是派生约束法，即说明派生时钟和主时钟的关系（频率、相位）。

cxu：

“异步时钟域，打两下”是不是两级同步化电路？

westor：

异步时钟域的数据采样，打两下，就是说用两级寄存器同步化，这样才能更好的保证数据采样和同步化。

您说同步化电路不能用于多个 bit 输入数据的采样，这个说法我没有见过，请将原文发给我，我分析一下。不知说的是不是多个 bit 的频率和相位各不相同的情况。

事实上，在设计电路时，输入数据总线都是多 bit 宽度的，而且正因为多个 bit 输入不能完全保证对齐（但是频率是一致的），才更需要用若干级寄存器采样，保证同步时序设计的输入同步。

草梢：

我想接收实时的低速串行数据 8M 左右，有帧格式的有同步时钟的，接收放入 FIFO，再用较高速 25M 并行取出来给其他芯片处理，这样的话有两个时钟，而其实接收和取数并没有很大关系，在一个 FPGA 里实现好吗？

westor：

在一个 FPGA 里实现没有关系，只要把 FIFO 的时序安排好即可。你说的这种用 FIFO 的处理方法，就是前面提到的数据接口常用方法之一。

（编者注：关于数据接口的详细内容可以在 westor 的经典网文《FPGA 设计指导原则》中找到。）

关于有限状态机编码的技巧和注意事项

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=14&topic=2015
（主题编号：1402015；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题是westor原创的关于有限状态机编码的技巧和注意事项，内容精练，通俗易懂，可操作性很强。关于状态机编码的详细的英文资料读者可以在 http://www.edacn.net/cgi-bin/post.cgi 下载。

westor：

贴 1 状态机的编码

a、状态机的编码。Biary、gray-code 编码使用最少的触发器，较多的组合逻辑。而 one-hot 编码反之。由于 CPLD 更多的提供组合逻辑资源，而 FPGA 更多的提供触发器资源，所以 CPLD 多使用 gray-code，而 FPGA 多使用 one-hot 编码。另一方面，对于小型设计使用 gray-code 和 binary 编码更有效，而大型状态机使用 one-hot 更高效。

B、在代码中添加综合器的综合约束属性或者在图形界面下设置综合约束属性可以比较方便地改变状态的编码。

VHDL 的示例：

Synplicity:

```
attribute syn_encoding : string;
attribute syn_encoding of <signal_name> : type is "value ";
```

The syn_encoding attribute has 4 values : sequential, onehot, gray and safe.

Exemplar:

```
-- Declare TYPE_ENCODING_style attribute
-- Not needed if the exemplar_1164 package is used
type encoding_style is (BINARY, ONEHOT, GRAY, RANDOM, AUTO);
attribute TYPE_ENCODING_style : encoding_style;
...
attribute TYPE_ENCODING_style of <typename> : type is ONEHOT;
```

Verilog 示例：

Synplicity:

```
Reg[2:0] state; /* synthesis syn_encoding = "value" */;
// The syn_encoding attribute has 4 values : sequential, onehot, gray and safe.
```

Exemplar:

```
Parameter /* exemplar enum <type_name> */ s0 = 0, s1 = 1, s2 = 2, s3 = 3, S4 = 4;
Reg [2:0] /* exemplar enum <type_name> */ present_state, next_state
```

贴 2：状态机的编码风格

a、关于 FSM 的编码方法。FSM 分两大类：米勒型和摩尔型。组成要素有输入（包括复位），状态（包括当前状态的操作），状态转移条件，状态的输出条件。 设计 FSM 的方法和技巧多种多样，但是总结起来有两大类：第一种，将状态转移和状态的操作和判断等写到一个模块（process、block）中。另一种是将状态转移单独写成一个模块，将状态的操作和判断等写到另一个模块中（在 Verilog 代码中，相当于使用两个“always” block）。其中较好

的方式是后者。其原因如下。 首先 FSM 和其他设计一样，最好使用同步时序方式设计，好处不再赘述。而状态机实现后，状态转移是用寄存器实现的，是同步时序部分。状态的转移条件的判断是通过组合逻辑判断实现的，之所以第二种比第一种编码方式合理，就在于第二种编码将同步时序和组合逻辑分别放到不同的程序块（process，block）中实现。这样做的好处不仅仅是便于阅读、理解、维护，更重要的是利于综合器优化代码，利于用户添加合适的时序约束条件，利于布局布线器实现设计。

（编者注：米勒型和摩尔型就是通常在书上看到的 Mealy 状态机和 Moore 状态机）

b、初始化状态和默认状态。 一个完备的状态机（健壮性强）应该具备初始化状态和默认状态。当芯片加电或者复位后，状态机应该能够自动将所有判断条件复位，并进入初始化状态。需要注明的一点是，大多数 FPGA 有 GSR（Global Set/Reset）信号，当 FPGA 加电后，GSR 信号拉高，对所有的寄存器，RAM 等单元复位/置位，这是配置于 FPGA 的逻辑并未生效，所以不能保证正确的进入初始化状态。所以使用 GSR 企图进入 FPGA 的初始化状态，常常会产生种种不必要的麻烦。一般的方法是采用异步复位信号，当然也可以使用同步复位，但是要注意同步复位的逻辑设计。解决这个问题的另一种方法是将默认的初始状态的编码设为全零，这样 GSR 复位后，状态机自动进入初始状态。 令一方面状态机也应该有一个默认（default）状态，当转移条件不满足，或者状态发生了突变时，要能保证逻辑不会陷入“死循环”。这是对状态机健壮性的一个重要要求，也就是常说的要具备“自恢复”功能。对应于编码就是对 case，if－else 语句要特别注意，要写完备的条件判断语句。VHDL 中，当使用 CASE 语句的时候，要使用“When Others”建立默认状态。使用“IF...THEN...ELSE”语句的时候，要用“ELSE”指定默认状态。Verilog 中，使用“case”语句的时候要用“default”建立默认状态，使用“if...else”语句的注意事项相似。 另外提一个技巧：大多数综合器都支持 Verilog 编码状态机的完备状态属性－－“full case”。这个属性用于指定将状态机综合成完备的状态，如 Synplicity 的综合工具（Synplify/Synplify Pro,Amplify，etc）支持的命令格式如下：

```
case (current_state) // synthesis full_case
2’b00 : next_state <= 2’b01;
2’b01 : next_state <= 2’b11;
2’b11 : next_state <= 2’b00;
//这两段代码等效
case (current_state)
2’b00 : next_state <= 2’b01;
2’b01 : next_state <= 2’b11;
2’b11 : next_state <= 2’b00;
default : next_state <= 2bx;
```

Synplicity 还有一个关于状态机的综合属性，叫“// synthesis parallel_case”其功能是检查所有的状态是“并行的”（parallel），也就是说在同一时间只有一个状态能够成立。

C、使用完备的“if...else”还有一个重要的好处，就是可以避免生成非目的性的“锁存器”（Latch）。详见 westor 的一篇文章“为什么 XST 与 Synplify 的综合结果不一样？”

D、状态机的定义可以用 parameter 定义，但是不推荐使用`define 宏定义的方式，因为‘define 宏定义在编译时自动替换整个设计中所定义的宏，而 parameter 仅仅定义模块内部的参数，定义的参数不会与模块外的其他状态机混淆。

做 distributed ram 时遇到的几个不太明白的信号

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=13&topic=1230
（主题编号：1301230；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题解释了分布式随机存储器设计中几个重要的信号。

llyy：

Inverting Control Pins:

The two control pins, WCLK and WE, each have an individual inversion option. Any control signal, including the clock, can be active at logic level 0 (negative edge for the clock) or at logic level 1 (positive edge for the clock) without requiring other logic resources.

Global Set/Reset — GSR:

The global set/reset (GSR) signal does not affect distributed RAM modules.

Global Write Enable — GWE:

The global write enable signal, GWE, is asserted automatically at the end of device configuration to enable all writable elements. The GWE signal guarantees that the initialized distributed-RAM contents are not disturbed during the configuration process. Because GWE is a global signal and automatically connected throughout the device, the distributed RAM primitive does not have a GWE input pin.

这是我在 Spartan-3 关与 distributed ram 的手册上看到的一段话，我不太明白这三个信号的具体意义。

agump：

我的理解，如果哪位大侠理解这些信号，请指点：

1、WCLK 和 WE 信号分别是分布式 RAM 的写时钟信号和写使能信号。WCLK 可以是上升沿有效，WE 可以是高电平有效。手册中提到的反转选项，是指这两个信号可以根据需要，分别的设置为 WCLK 下降沿有效，WE 低电平有效。

2、对于块 RAM，GSR 信号有效会使块 RAM 的输出取 init 属性的值，而 GSR 信号对分布式 RAM 不起任何作用。（可以阅读两种 RAM 的仿真模型）

3、GWE 信号在装载配置文件的过程中无效，这样可使得器件在装载配置文件的过程中所有可写元素（寄存器，RAM 等）都不能写（因为 GWE 信号无效），可防止装载过程中初始化过的可写元素内容被破坏（因为这时器件逻辑不受控），在器件配置完成之后，GWE 回复有效，这时可写元素才能写入。对于分布式 RAM 而言，GWE 信号可保证其内容被正确初始化，也只起这个作用，使用是自动的，无需用户关心，所以在分布式 ram 的原语中也不提供这个引脚供用户使用。可以参考配置方面对此信号的描述。

GWE 信号无需关心，这个信号掩埋在芯片中，就是在配置过程中自动使用的，是 Xilinx 的芯片实现芯片配置的一种技术手段，我想在 datasheet 甚至是可以不用提的，就是说你的代码中完全可以不用提到这个信号。至于 GSR 就要复杂得多。有一点是肯定的，如果使用分布式 RAM，像 GWE 信号一样，也可以在代码中不用提到 GSR 信号。

Source Insight 兼容 VHDL 与 VERILOG

相关主题连接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=13&topic=3110
（主题编号：1303110；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题是网友笑嘻嘻对 Source Insight 这个 HDL 编辑器做出了改进，提高了编辑器对 VERILOG HDL 和 VHDL 的支持，并提供了详细的使用方法。扩展包.clf 文件读者可以到相关链接下载。*

笑嘻嘻：

这一阵子本人对 source insight 的语言扩展规则做了较深入的研究，基本上已经摸清了一些简单的变量定义，在此基础上做了一个 Verilog HDL 的最新扩展包 v1.1，使得 source insight 对 Verilog HDL 的支持大大增强：用颜色进行语法检测、编辑时的变量自动匹配，这些功能将 source insight 的优点都发掘了出来。

source insight 对 VHDL 的支持不是很强，只是简单的表达了模块与例化的定义，对其强项变量的定义不支持，留下了一个较大的遗憾。

为了照顾使用 VHDL 的各位工程师朋友，我做了一个 VHDL 的扩展包，用于增强 source insight 对 VHDL 的支持，目前效果还不错，基本上实现了：用颜色进行语法检测、编辑时的变量自动匹配。

目前仍在优化与完善之中，先给大家一个预览，稍后将推出稳定的版本。

Verilog HDL 语言功能描述：

1、支持 input、output、inout 端口定义及建立 symbol 索引，单行最多支持到 10 个 symbol；
2、支持 wire、reg 变量定义及建立 symbol 索引，单行最多支持到 10 个 symbol；
3、支持 module、task、function 模块定义及建立 symbol 索引；
4、支持模块例化定义及建立 symbol 索引；
5、支持 parameter 定义及建立 symbol 索引，一行只能定义一个 parameter；
6、所有被定义为 symbol 的字符串会在 symbol window 显示，并可以点击跳转；
7、各种类型的 symbol 在整个代码中以各自定义的颜色显示；
8、光标移动到被调用的 symbol，context window 会显示该 symbol 的定义代码；
9、编辑代码时自动激活相匹配的 symbol 索引，快速输入变量；
10、支持 project；
11、综合以上功能，可进行视觉语法检测。
以上功能仅限于可综合的 Verilog HDL 代码，对于其他暂时还不支持。

VHDL 语言功能描述：

1、支持 in、out、inout 端口定义及建立 symbol 索引，单行定义 symbol 不限量；
2、支持 signal 变量定义及建立 symbol 索引，单行最多支持到 6 个 symbol；
3、支持 component、package、function、entity、architecture、process、block 模块定义及建立 symbol 索引；
4、支持 attribute 定义及建立 symbol 索引；
5、支持 use 库定义及建立 symbol 索引；

6、支持 type 结构定义及建立 symbol 索引；

7、支持 constant 参数定义及建立 symbol 索引；

8、支持参数（natural、string、integer、positive）定义及建立 symbol 索引，一行只能定义一个参数；

9、所有被定义为 symbol 的字符串会在 symbol window 显示，并可以点击跳转；

10、各种类型的 symbol 在整个代码中以各自定义的颜色显示；

11、光标移动到被调用的 symbol，context window 会显示该 symbol 的定义代码；

12、编辑代码时自动激活相匹配的 symbol 索引，快速输入变量；

13、支持 project；

14、综合以上功能，可进行视觉语法检测。

以上功能仅限于可综合的 VHDL 代码，对于其他暂时还不支持。

在 source insight 加入扩展包*.clf 的方法：

1、以前的版本请删除 options－>preferences－>languages－>delete－>Verilog HDL

2、options－>preferences－>languages－>import－>*.clf

3、document options－>add type：

document type：Verilog HDL

file filter：*.v

language：Verilog HDL v1.1.2

选中：symbol window

4、ok！

如何实现信号延时？

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=7&topic=5482
（主题编号：0705482；编辑整理：yangfeng；校对改正：myname；统稿：wenxinniwo）

编者提示：在论坛里经常见到关于实现信号延时的讨论,为此我们总结了实现信号延时经常使用的方法,供大家参考。

Wjmn：
实现信号延时的方法汇总：
1、门延时数量级的延时（几个 ns），可用逻辑门来完成，但告诉综合器不要将其优化掉（不精确，误差大，常常不被推荐）。比如用两个非门(用 constraint 来告诉 synthesizer 不要综合掉这些逻辑)。
2、使用 delay cell，lcell。
3、采用更快的时钟，通过计数器来实现,对于比较小的延时,用两个 DFF 级联就可以。
4、用 fifo 或 ram 实现。
5、通过移位寄存器实现，延迟的时间和移位寄存器的位数有关。
6、使用负时钟驱动DFF来获得半个时钟周期的延时。
7、使用合适的buffer甚至用几个buffer的串联。
8、VHDL Component Declaration:
```
COMPONENT LCELL
PORT (a_in : IN STD_LOGIC;
  a_out: OUT STD_LOGIC);
END COMPONENT;
```
9、在寄存器赋值时加一点延时对仿真有好处，可以完全消除竞争现象。
比如这样的逻辑：
```
always @(posedge clk)
abc_r <= #`HOLD_DELAY abc_nxt;
```
是有利于仿真的。
编辑注：对这点持保留态度。
10、在综合时将忽略所有的延时语句。

Westor：
异步电路产生延时的一般方法是插入一个Buffer、两级非门等，这种延时调整手段是不适用于同步时序设计思想的。首先要明确一点HDL语言中的延时控制语法，例如：#5 a<=4'b0101;其中的延时5个时间单位，是行为级代码描述，常用于仿真测试激励，但是在电路综合是会被忽略，并不能启动延时作用。同步时序电路的延时一般是通过时序控制完成的。换句话说，同步时序电路的延时被当作一个电路逻辑来设计。对于比较大的和特殊定时要求的延时，一般用高速时钟产生一个计数器，根据计数器的计数，控制延时；对于比较小的延时，可以用D触发器打一下，这种做法不仅仅使信号延时了一个时钟周期，而且完成了信号与时钟的初次同步，在输入信号采样和增加时序约束余量中使用。

Cherrydu：
如果要实现一个“可综合”的信号延时（若干个时钟周期），比如给信号 reset 让其下降沿延时若干周期,以用什么方法实现呢？我是用 Verilog 来编这个程序的，我曾经试过在 reset 上升沿到来时，用数时钟数目的方法实现延时，可是在 Verilog 下，异步方法无法实现这个功能，同步的方法，if 语句不能嵌套两个沿的情况.

Rush：
不用嵌套两个沿，触发条件只有 CLK 即可，然后在下面检测 RESET 信号，等他下降沿也就是由 1 变为 0 时（IF RESET=0)开始转入另外一个状态，然后在新状态中计数。

Darkage：
altera 的一个 lcell 的延时并不是固定的，而是要看你将 lcell 的位置放在那里，两个 lcell 的距离越远，延时就越大，不同的类型芯片，不同的级别，不同的温度都是不一样的，这些是要考虑到的。两个 lcell 的延时可以从 1 个多的 ns 都 10 多个 ns 都可以做到的，要是对于环境条件要求不是很严格的时候，可以考虑这种方法，否则就要考虑其他的方案，或者做一些折中了。

丁丁：
用 Xilinx 器件中的 LUT 实现 SRL16 的功能，也就是 1 个查找表实现 16 级时延，替代 16 个寄存器，可以大大节省资源。

[转载]新手学习技巧

相关主题链接：
http://www.edacn.net/cgi-bin/topic.cgi?forum=7&topic=5728
（主题编号：0705728；编辑整理：wuxinhua1982；校对改正：myname；统稿：wenxinniwo）

编者提示：本主题由 hyzhangwang 网友转载，这些都是对初学者的忠告。如果初学者用心领悟，细细回味，将会对你的学习有很大的帮助。

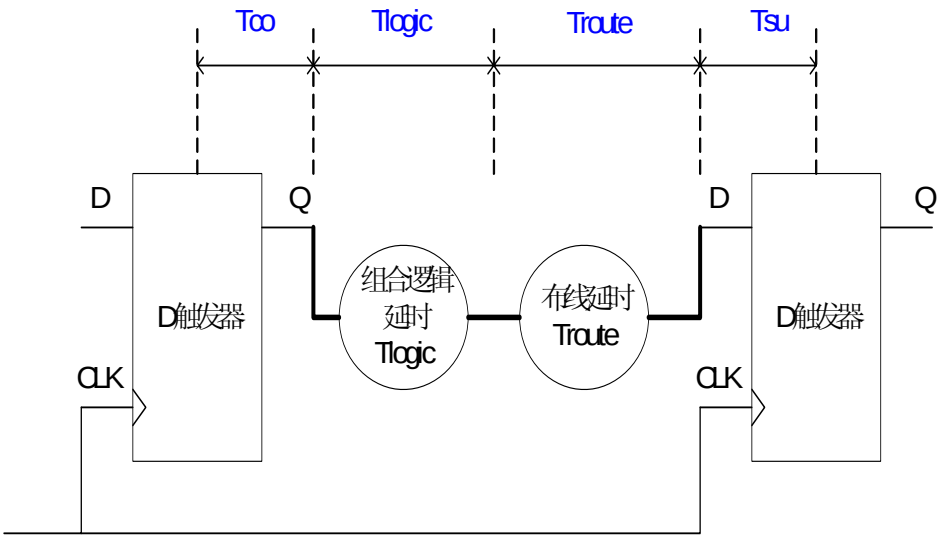
1．不要看到别人的回复，第一句话就说：给个代码吧！你应该想想为什么。当你自己想出来再参考别人的提示，你就知道自己和别人思路的差异。
2．初学者请不要看太多的书那会误人子弟的。先找一本好书系统地学习。很多人用了很久都是只对部分功能熟悉而已，不系统还是不够的。
3．看帮助。不要因为很难，而自己又是初学者所以就不看；帮助永远是最好的参考手册，虽然帮助的文字有时候很难看懂，或不够直观。
4．不要被一些专用词汇所迷惑；最根本的是先了解最基础的知识。
5．不要放过任何一个看上去很简单的小问题--他们往往并不那么简单，或者可以引申出很多知识点；不会举一反三你就永远学不会。
6．知道一点东西，并不能说明你会用，会用是需要实践经验和时间积累的。
7．入门并不难，难的是长期坚持实践和不遗余力地学习和实践。
8．看再多的书也学不会用的，要多实践，多动手。
9．把时髦的技术挂在嘴边，还不如把基本的技术记在心里。
10．在任何时刻都不要认为自己手中的书已经足够了。
11．看得懂的书，请仔细看；看不懂的书，请硬着头皮看。
12．别指望看第一遍书就能记住和掌握什么——请看第二遍、第三遍。
13．请把书上的例子亲手去实践一下。
14．把在书中看到的有意义的例子进行扩充；并将其切实地运用到自己的设计中。
15．不要漏掉书中任何一个练习和实例——请全部做完并记录下思路。
16．别心急，应用确实不容易，水平是在不断的实践中完善和发展的。
17．每学到一个知识点的时候，尝试着对别人讲解这个知识点并让他理解----你能讲清楚才说明你真的理解了。
18．记录下在和别人交流时发现的自己忽视或不理解的知识点。
19．保存好你做过的所有的源文件----那是你最好的积累之一。
20．对于网络，还是希望大家能多利用一下。很多问题不是非要到论坛来问不可，首先你要学会自己找答案，比如 google、百度都是很好的搜索引擎，你只要输入关键字就能找到很多相关资料，别老是等待别人给你答案，看的出你平时一定也很懒！
21．到一个论坛，你要学会去看以前的帖子，不要什么都不看就发帖子问，也许你的问题早就有人问过了，你再问，别人已经不想再重复回答了，作为初学者，谁也不希望自己的帖子没人回的。
22．虽然不是打击初学者，但是这句话还是要说：论坛论坛，就是大家讨论的地方，如果你总期望有高手总无偿指点你，除非他是你亲戚！讨论者，起码是水平相当的才有讨论的说法，如果水平真差距太远了，连基本操作都需要别人给解答，谁还跟你讨论呢。

浮躁的人容易问：我到底该学什么；----别问，学就对了；
浮躁的人容易问：有钱途吗；----建议你去抢银行算了；
浮躁的人容易说：我要中文版！我英文不行！----不行？学呀！
浮躁的人分两种：只观望而不学的人；只学而不坚持的人；
浮躁的人永远不是（也成不了）一个高手。

如何计算设计频率

我们的设计需要多大容量的芯片?我们的设计能跑多快?这是经常困扰工程师的两个问题.对于前一个问题,我们可能还能先以一个比较大的芯片实现原型,待原型完成再选用大小合适的芯片实现.对于后者,我们需要一个比较精确的预估,我们的设计能跑 50M,100M 还是 133M?

首先让我们先来看看 Fmax 是如何计算出来的.图(1)是一个通用的模型用来计算 FPGA 的.我们可以看出,Fmax 受 Tsu , Tco , Tlogic 和 Troute 四个参数影响.
(由于使用 FPGA 全局时钟,时钟的抖动在这里不考虑).



图(1) 时钟周期的计算模型

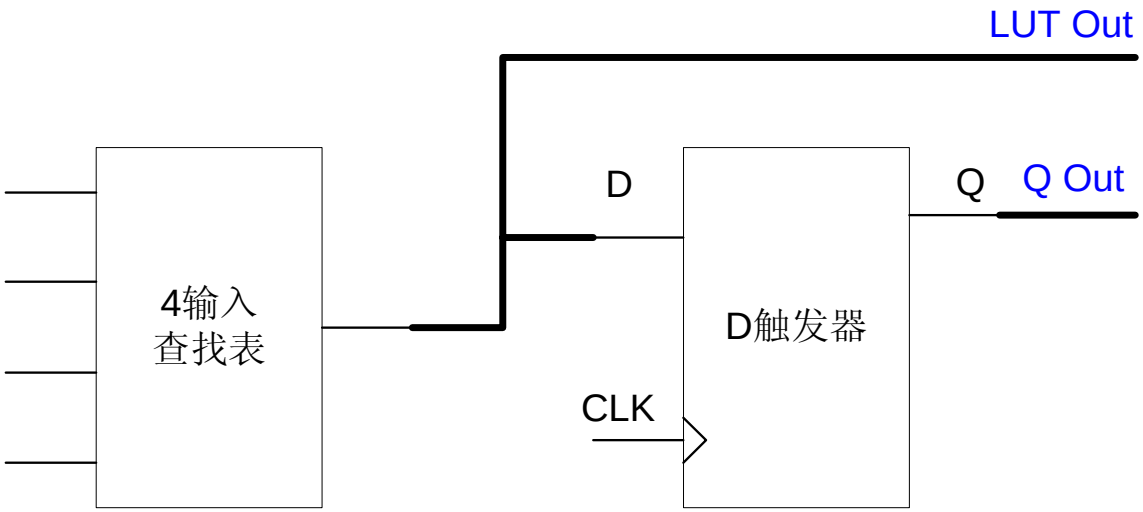
时钟周期 T = Tco + Tlogic + Troute + Tsu
时钟频率 Fmax = 1/Tmax

其中:
Tco : D 触发器的输出延时
Tlogic : 组合逻辑延时
Troute : 布线延时
Tsu : D 触发器的建立时间

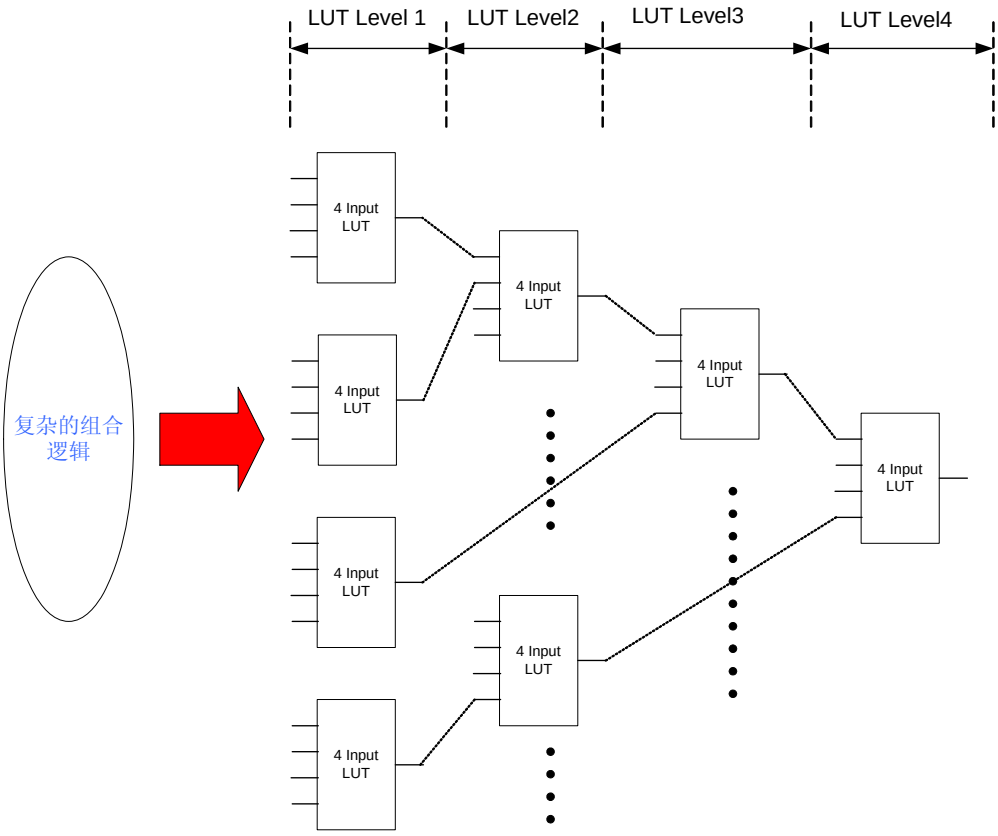
由图(1)可以看出,在影响 Fmax 的四个参数中,由于针对某一个器件 Tsu 和 Tco 是固定的,因此我们在设计中需要考虑的参数只有两个 Tlogic 和 Troute.通过良好的设计以及一些如 Pipeline 的技巧,我们可以把 Tlogic 和 Troute 控制在一定的范围内.达到我们所要求的 Fmax.

经验表明一个良好的设计,通常可以将组合逻辑的层次控制在4层以内,即(Lut Levels <= 4) .而 Lut Levels(组合逻辑的层次)将直接影响 Tlogic 和 Troute 的大小. 组合逻辑的层次多,则 Tlogic 和 Troute 的延时就大,反之, 组合逻辑的层次少,则 Tlogic 和 Troute 的延时就小.

让我们回过头来看看 Xilinx 和 Altera 的 FPGA 是如何构成的.是由 Logic Cell (Xilinx) 或 Logic Element(Altera)这一种基本结构和连接各个 Logic Cell 或 Logic Element 的连线资源构成.无论是 Logic Cell 还是 Logic Element ,排除其各自的特点,取其共性为一个 4 输入的查找表和一个 D 触发器.如图(2)所示.而任何复杂的逻辑都是由此基本单元复合而成.图(3).上一个 D 触发器的输出到下一个 D 触发器的输入所经过的 LUT 的个数就是组合逻辑的层次(Lut Levels).因此,电路中用于实现组合逻辑的延时就是所有 Tlut 的总和.在这里取 Lut Levels = 4 .故 Tlogic = 4 * Tlut .



图(2) FPGA基本逻辑单元



图（ 3 ） 复杂组合逻辑的实现

解决的 Tlogic 以后,我们来看看 Troute 如何来计算.由于 Xilinx 和 Altera 在走线资源的设计上并不一样,并且 Xilinx 没有给出布线延时的模型,因此更难于分析,不过好在业内对布线延时与逻辑延时的统计分析表明, 逻辑延时与布线延时的比值约为 1:1 到 1:2.由于我们所选用的芯片大量的已经进入 0.18um 和 0.13um 深亚微米的工艺,因此我们取逻辑延时与布线延时的比值为 1:2.

Troute = 2 * Tlogic

Tmax = Tco + Tlogic + Troute + Tsu

= Tco + Tsu + 3 * Tlogic

= Tco + Tsu + 12 * Tlut

下表是我们常用的一些 Xilinx 和 Altera 器件的性能估算.我们选取的是各个系列中的最低的速度等级.由于 Altera 的 APEX ,APEX II 系列器件的不同规模的参数不同,我们选取 EP20K400E 和 EP2A15 作代表.

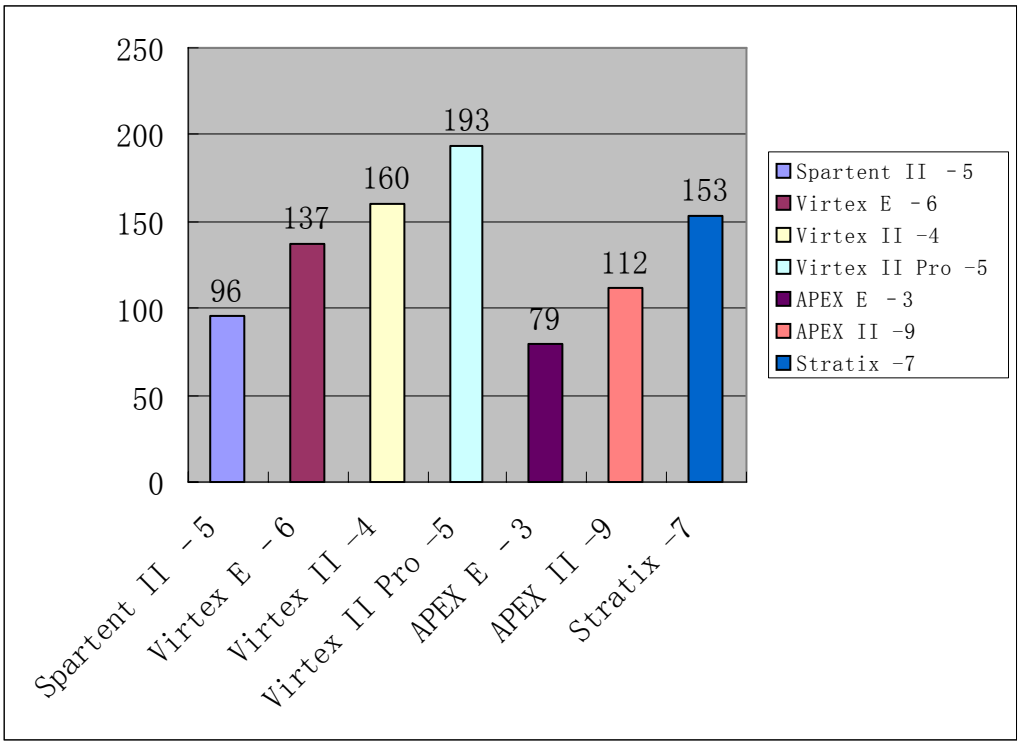
表格 1

	Tsu (ns)	Tco (ns)	Tlut (ns)	Fmax
Spartent II –5	0.7	1.3	0.7	96 M
Virtex E –6	0.63	1.0	0.47	137M
Virtex II -4	0.37	0.57	0.44	160M
Virtex II Pro -5	0.29	0.40	0.37	193M

APEX E –3 #	0.23	0.32	1.01	79M
APEX II -9##	0.33	0.23	0.7	112M
Stratix -7	0.011	0.202	0.527	153M

以 EP20K400E – 3 的数据计算得出.

以 EP2A15-9 的数据计算得出.



FPGA 配置(yangfeng 专稿)

编者提示：以下内容是汇总了论坛里多个关于 FPGA 配置的帖子而成，并没有一个针对该主题的详细讨论，因此没有给出主题链接。

问：对 epsc1 和 epsc4 的配置必须用专用的编程器吗?还是可以直接用 quartus+byteblasterII 下载啊?

回答：byteblasterII 下载电缆就可以，在板上做一个 AS 口

问：如何用单片机配置 FPGA？串行还是并行？

回答：如果用单片机或其他 CPU 配置当然是选择并行了.如果你的单片机在板子上采用并行的方法简单快速，采用串行多此一举了.因为串行的方法主要是减少板子到外面的连线,所以器件大多支持 JTAG.如果单片机放在板子上不但可以上电并行加载还可以设计成动态配置,就是说当你的设计主要用于两种或以上的目的(如两种工作模式)，可利用外面的触发条件完全改变你的设计,不但灵活而且可以大大减少成本,想象一下原来需要 100 万门的器件现在只要 50 万就可以了.增加的只是成本相对 FPAG 来说不值一提的单片机。

FPGA 器件的配置方式和配置文件：

Altera 公司生产的具有 ICR 功能的 FPGA 器件有 FLEX6000、FLEX10K、APEX 和 ACEX 等系列。它们的配置方式可分为 PS（被动串行）、PPS（被动并行同步）、PPA（被动并行异步）、PSA（被动串行异步）和 JTAG（Joint Test Action Group）等五种方式。这五种方式都能适用于单片机配置。PS 方式因电路简单，对配置时钟的要求相对较低，而被广泛应用。下图是 PS 配置方式的时序图。

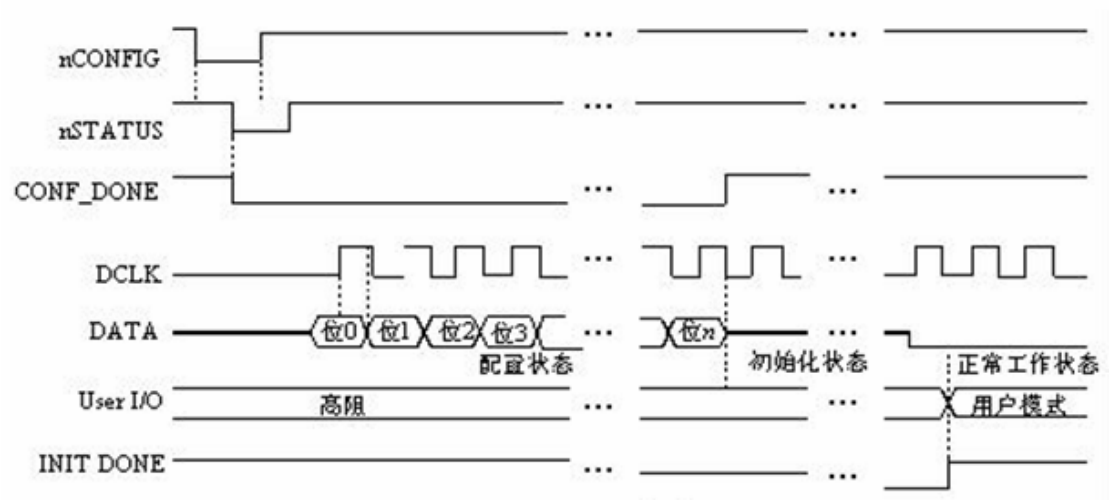


图　PS 方式配置时序

CPU 仅需要利用 5 个 I/O 脚与 FPGA 相连，就实现了 PS 方式的硬件连接，具体信号见下表（信号方向从 CPU 侧看）：

信号名	I/O	说明
Data	O	configuration data
DCLK	O	configuration clock
nCONFIG	O	device reset (a low to high transition starts the configuration within the device)
CONF_DONE	I	Status bit (gets checked after configuration, will be high if configuration complete)
nSTATUS	I	Status bit indicating an error during configuration if low

被动串行工作过程：当 nconfig 产生下降沿脉冲时启动配置过程，在 dclk 上升沿，将数据移入目标芯片。在配置过程中，系统需要实时监测，一旦出现错误，nSTATUS 将被拉低，系统识别到这个信号后，立即重新启动配置过程。配置数据全部正确地移入目标芯片内部后，CONF_DONE 信号跳变为高，此后，DCLK 必须提供几个周期的时钟（具体周期数与 DCLK 的频率有关），确保目标芯片被正确初始化，进入用户工作模式。

Altera 的 MAX+PLUS II 或 Quartus II 开发工具可以生成多种配置或编译文件，用于不同配置方法的配置系统，而对于不同系列的目标器件配置数据的大小也不同，配置文件的大小一般有.rbf文件决定。.rbf文件即二进制文件。该文件包括所有的配置数据，一个字节的 .rbf 文件有 8 位配置数据，每一字节在配置时最低位最先被装载。微处理器可以读取这个二进制文件，并把它装载到目标器件中。Altera 提供的软件工具不自动生成 .rbf 文件，须按照下面的步骤生成：① 在 MAX+PLUS II 编译状态，选择文件菜单的变换 SRAM 目标文件命令；② 在变换 SRAM 目标文件对话框，指定要转换的文件并且选择输出文件格式为.rbf(Sequential)，然后确定。

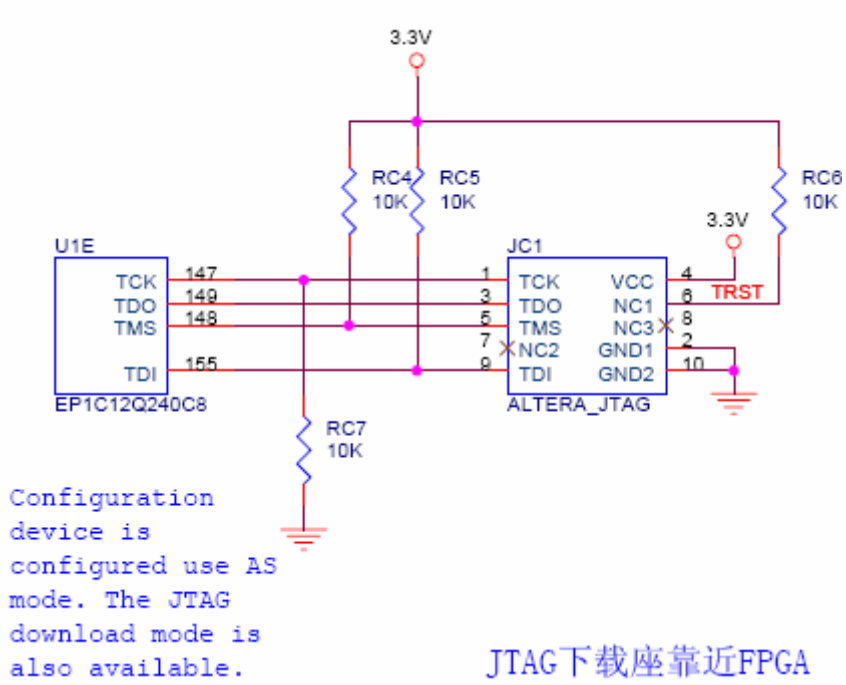
配置操作过程：

CPU 按下列步骤操作 I/O 口线，即可完成对 FPGA 的配置：
1、nCONFIG="0"、DCLK="0"，保持 2μS 以上。
2、检测 nSTATUS，如果为"0"，表明 FPGA 已响应配置要求，可开始进行配置。否则报错。正常情况下，nCONFIG="0"后 1μS 内 nSTATUS 将为"0"。
3、nCONFIG="1"，并等待 5μS。
4、Data0 上放置数据（LSB first），DCLK="1"，延时。
5、DCLK="0"，并检测 nSTATUS，若为"0"，则报错并重新开始。
6、准备下一位数据，并重复执行步骤 4、5，直到所有数据送出为止。
7、此时 Conf_done 应变成"1"，表明 FPGA 的配置已完成。如果所有数据送出后，Conf_done 不为"1"，必须重新配置（从步骤 1 开始）。
8、配置完成后，再送出 10 个周期的 DCLK，以使 FPGA 完成初始化。

FPGA 配置电路：

1、JTAG 接口：

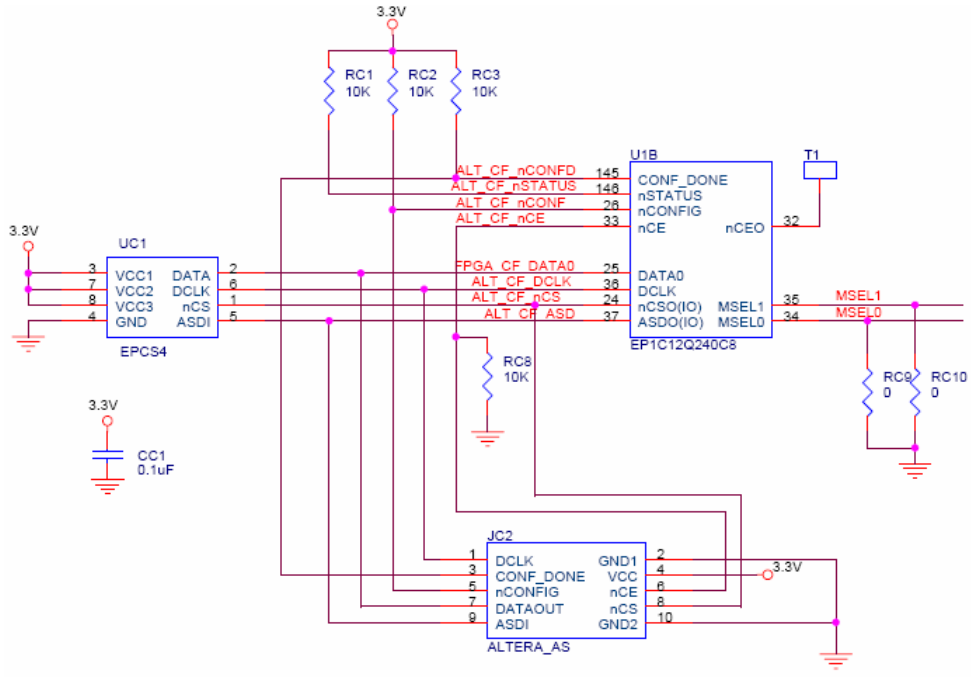
JTAG 接口可以用来调试FPGA，下载速度比较快，而且支持SignalTAP。但是不能用来编程EPCS 芯片。建议调试阶段采用JTAG 模式。电缆可以采用ByteBlaster （MV） 也可以用用ByteBlaster II。



摘自“CYCLONE 原理图”

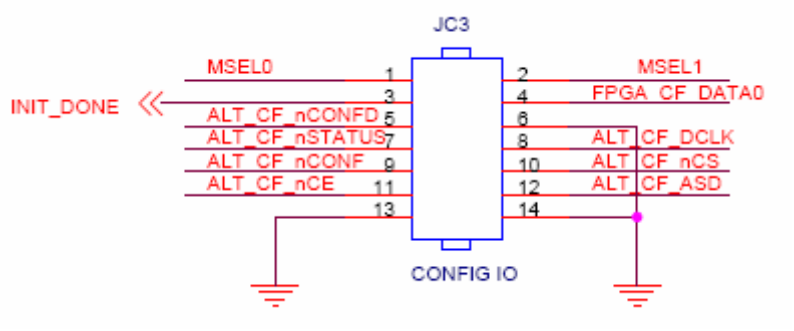
2、AS 接口

AS 接口主要是用来编程EPCS 芯片，同时也可以用来调试。具体过程是首先编程EPCS，然后通过EPCS 配置FPGA，运行程序。需要考虑的是EPCS 的编程次数是有限制的，虽然比EPC 系列要多，但是太频繁的擦除和写入对芯片还是有一定影响的。所以，我们建议在调试结束后，程序固化的时候才使用AS 方式。 如果采用这种方式，必须采用ByteBlasterII。电缆才行。



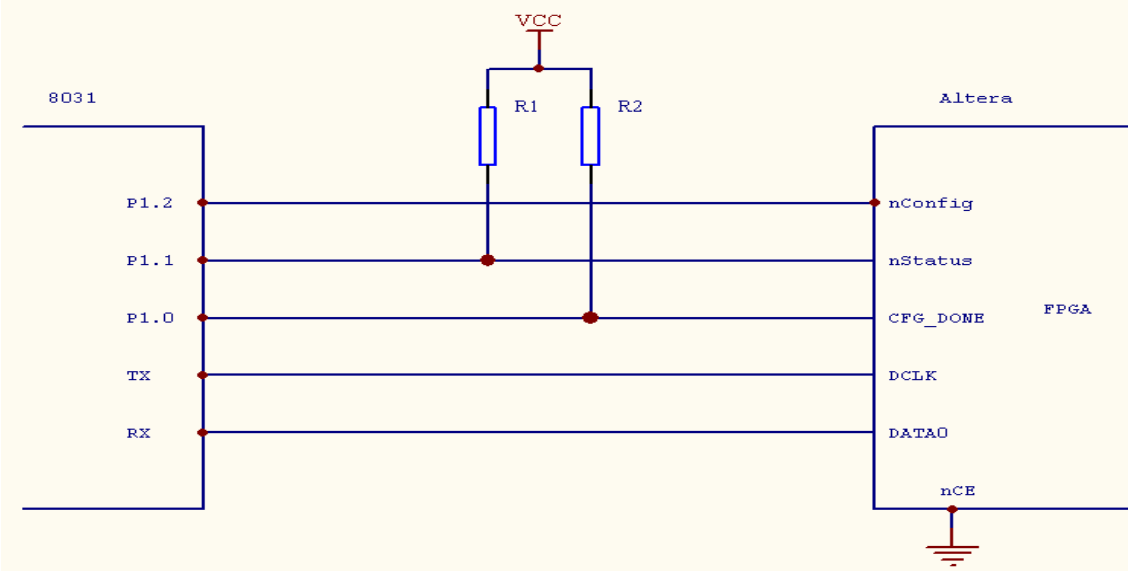
摘自“CYCLONE 原理图”

3、采用 CPLD 或者单片机配置 FPGA



摘自“CYCLONE 原理图”

可以通过这些配置引脚由 CPLD 来对 FPGA 进行配置。利用单片机配置 FPGA 的电路图如下：



与上图对应的用单片机配置 FPGA 的 C 程序如下：

```
/* 用 8031 加载 ALtera 的 FPGA，也可用于 Xilinx 的 FPGA 的加载 */
void load_epld(void)
{
    unsigned char data i;
    unsigned int data j;
    unsigned char xdata * data pt;
    SCON = 0x0;                                    /* 设置 8031 工作在方式 0，同步串行方式 */
    CONFIG = 0;                                    /* 初始化 FPGA */
    i=4             /* 注意 8031 的工作频率和速度，对于 Xilinx 的 FPGA，初始化的时间要长
                                                   一些，具体查下各个 FPGA 的参数 */
    while(STATUS!=0) {
        i--;
        if(i==0) {
            CONFIG = 0;
            FPGA_Init_Error();      /* FPGA 初始化错误处理程序，自己编制 */
```

```
        }
    }
    CONFIG = 1;                                    /* 初始化 FPGA 完毕 */
    i=10;                          /* 注意 8031 的工作频率和速度，各个 FPG 的初始化的时间不
        一样，具体查下各个 FPGA 的参数 */
    while(STATUS!=1) {
        i--;
        if(i==0) {
            CONFIG = 0;                  /* 错误时，使 FPGA 处于初始化状态，保证电路
处于安全状态 */
            FPGA_Init_Error();      /* FPGA 初始化错误处理程序，自己编制 */
        }
    }
    pt=(unsigned char xdata *)EPLD_DATA;    /* EPLD_DATA 为 FPGA 的存放地址 */
    for(j=0; j < EPLD_Number; j++) {          /* EPLD_Number 为 FPGA 的加载字节数，当
                                              大于 64K */
        SBUF = *pt++;                              /* 字节时，要采用页面切换的方式 */
        while(!TI);                                  /* 等待 1 个字节加载完毕 */
        TI=0;
        if(STATUS!=1) {                            /* 加载过程中，检查加载是否正确 */
            CONFIG = 0;                            /* 错误时，使 FPGA 处于初始化状态，
保证电路处于安全状态 */
            FPGA_Load_Error();                     /* FPGA 加载错误处理程序，自己编制 */
        }
    }
    if(CFG_DOWN!=1) {                        /* FPGA 数据加载完毕，检查加载是否正确 */
        CONFIG = 0;                        /* 错误时，使 FPGA 处于初始化状态，保证电路
                                    处于安全状态 */
        FPGA_Done_Error();                         /* FPGA 加载错误处理程序，自己编制 */
    }
}
```

《FPGA/CPLD 设计工具－Xilinx ISE 5.x 使用详解》读书笔记

(yangfeng 专稿)

编者提示:《FPGA/CPLD 设计工具－Xilinx ISE 5.x 使用详解》是 EDACN 论坛领军人物 westor 等于 2003 年推出的代表作，本人在阅读此书过程中将一些精华内容作了笔记，以方便日后的复习，也希望大家能从中获得一些有用的知识。以下是该书的内容简介： 本书以 FPGA/CPLD 设计流程为主线，阐述了如何合理利用 ISE 设计平台集成的各种设计工具，高效地完成 FPGA/CPLD 的设计方法与技巧。全书在介绍 FPGA/CPLD 概念和设计流程的基础上，依次论述工程管理与设计输入、仿真、综合、约束、实现与布局布线、配置调试等主要设计步骤在 ISE 集成环境中的实现方法与技巧。本书立足工程实践，结合作者多年工作经验，选用大量典型实例，并配有一定数量的练习题。本书配套光盘收录了所有实例的完整工程目录、源代码、详细操作步骤和使用说明，利于读者边学边练，提高实际应用能力。

1、正确的编码方法：首先做到对要描述的硬件电路胸有成竹，对硬件的结构和连接十分清晰，然后再用HDL来描述出来。

2、处理时序不满足的设计技巧（Negative Slack)

A、适当的约束在一定程度上可以提高工作频率

分析设计的引脚路径，将关键路径单独约束，而且远远满足设计要求的逻辑松约束，重新综合。另外可以尝试Synplify Pro的一些时序改善工具，如Pipeline,Retiming,FSM Explore等。适当减少关键路径的扇出（Fanout)，因为高扇出需要多级缓冲（Buffer）驱动，会影响速度。取消资源共享（Resource Sharing)选项的选择。多做一些不同综合优化参数下的综合实验，找出频率最高的参数组合方式。

B、使用Amplify等物理约束综合器

对于某些设计，采用Synplicity公司的物理综合器Amplify进行关键路径的物理位置约束，可以较大幅度提高布局布线效率，改善工作频率

C、合理的编码风格(Coding Style)是提高设计质量的根本

实际上提高工作频率的最根本方法还是改善编程代码风格。为了改善设计频率和工作可靠性，一般采用同步时序电路设计。如果工程师对设计的硬件实现方法了然在胸，而且积累了好的代码风格与设计表示技巧，那么可以使设计在速度和面积上达到最优。

3、黑盒子（Black Box）的综合方法

A、对模块只声明模块名和输入输出端口，功能实体为空

B、实现综合约束条件：/*synthesis syn_black_box*/或者attribute syn_black_box of bbox:component is true，在SCOPE中属性设置页面添加综合成黑盒子的属性

4、Probe探针的使用

A、 /*synthesis syn_probe=1*/

B、在RTL视图的网线列表中选中需要添加Probe的信号，将其拖进SCOPE属性的综合属性页面，选择syn_probe属性为1

5、添加约束的方法

A、使用SCOPE添加，特别是使用属性（attribute）设置页面添加综合约束属性

B、HDL代码中添加约束综合命令

C、在主界面重要综合优化参数选项中选择

D、在综合优化参数对话框中选择有效

6、综合约束与布局布线约束的区别：

综合约束文件为SDC，而在ISE中布局布线约束是通过UCF文件实现。随着综合器和布线器的协调性能越来越好，几乎所有的布线约束都可以在综合阶段添加，包括时钟约束、寄存器约束、路径约束、引脚约束、FPGA/CPLD底层硬件原语约束等，其中有些约束在综合阶段不起作用，仅仅由综合器传递到布线器去，在布线器中才真正发挥作用。两者的区别为：综合约束和布线约束的重点不同，所以产生的效果也不同。布线约束与器件底层结构结合的更紧密，直接决定设计在芯片上的实现方法。另外布线约束的有限等级比综合约束要高，两者发生冲突时布线约束优先。

Synplify将SCD文件中的约束条件进行语法转换后，生成NCF约束文件，然后拷贝到ISE中就可。NCF与UCF的区别是UCF约束的优先级比NCF要高。所以如果SDC设计的好，UCF就可以放松要求。

7、 Retiming的原理：

自动向组合逻辑和查找表中插入寄存器，其本质是将组合逻辑和LUTS外的寄存器移入组合逻辑和LUTS中，所以被优化部分的输入和输出之间的寄存器级数并没有变化，不会破坏设计的时序。Retiming只对边沿触发的寄存器有效，不能移动电平敏感的锁存器。

8、Pipelining的原理：

通过向一些逻辑中添加寄存器来减少逻辑级数，改善时序条件。

9、全局时钟资源和第二全局时钟资源

9.1 全局时钟资源使用全铜层工艺实现，设计了专用时钟缓冲与驱动结构，使得全局时钟到达芯片内部的所有的CLB,IOB,BRAM的时延和抖动最小。

9.1.1 与全局时钟资源有关的Xilinx的原语：

IBUFG 输入全局缓冲，与专用全局时钟输入管脚连接的首级全局缓冲，所有从全局时钟管脚输入的信号必须经过IBUFG。

IBUFGDS是IBUFG的差分形式，信号从一对差分全局时钟管脚输入时，必须使用IBUFGDS。

BUFG是全局缓冲，其输入是IBUFG的输出。

BUFGCE是带有时钟使能端的全局缓冲，有一个输入I，使能CE，输出O。

BUFGMUX 是全局时钟选择缓冲，有两个输入时钟，通过控制端选择输出。

BUFGP＝IBUFG+BUFG

BUFGDLL 全局缓冲延迟锁相环，相当于BUFG＋DLL

DCM数字时钟管理单元，完成时钟的同步、分频、倍频、移相、去抖动等。

9.1.2 全局时钟资源的使用方法：

IBUFG+BUFG

IBUFGDS+BUFG

IBUFG+DCM+BUFG

Logic+BUFG

Logic+DCM+BUFG

9.2 使用全局时钟资源的注意事项：

使用IBUFG或者IBUFGDS的充分必要条件是信号从专用全局时钟资源输入。BUFGP同样必须满足这点。这是因为在结构上IBUFG或者IBUFGDS的输入端只跟芯片的全局时钟输入管脚相连。

使用全局时钟资源的方法：一、在程序中直接例化全局时钟资源。二、在综合阶段或实现阶段加约束。在Synplify中可以在SCOPE中约束属性中选择约束对象clk后，选择Attribute中的syn_noclockbuf为1。或者在代码中添加/*synthesis syn_noclockbuf=1*/

9.3 第二全局时钟资源

又叫长线资源，分布在芯片的行、列的栅栏上，其长度和驱动能力仅次于全局时钟资源，在设计中可以将频率高、扇出数多的时钟、使能、高速路径等信号指定为第二全局时钟信号。需要注意的是第二全局时钟资源使用的是芯片内部的布线资源，在设计密度大的情况下使用所有的第二全局时钟资源会给设计的布局布线带来负面影响，导致布线不成功。

10、XPower计算功耗

COMS电路的功耗来源于开关状态切换，XPower计算功耗时给每个开关元件（LUT,FF,BRAM,布线等）建立一个电容模型，根据用户设置的时钟和输入信号开关频率、同步元件的翻转率、特定器件的电容、静态功耗和其他数据估算FPGA的功耗，准确估算功耗的关键是获得准确的信号翻转率数据。

每个元件的功耗可以表示为：P=C*V*V*E*F*1000，单位mw。E为翻转率，E*F是一秒钟信号开关的次数。

11、FPGA配置与下载

FPGA的配置模式有四种：

A、主串模式(Master Serial)：待配置的FPGA产生配置时钟CCLK驱动外部串行PROM，从而读入配置数据。

B、从串模式(Slave Serial)：FPGA接收来自外部PROM的串行配置数据，在外部时钟CCLK的作用下完成配置，多个FPGA可以连接成菊花链(Daisy-chain)，以便从一个数据源获得配置数据。

C、SelectMap模式：FPGA接收来自外部的并行配置数据和配置时钟CCLK。

D、边界扫描模式(JTAG模式）：通过JTAG口配置，可以节省资源，每个TCK传送1bit配置数据。

iMPACT支持四种下载模式：边界扫描(Boundary Scan)模式、从串模式(Slave Serial)、SelectMap模式、Desktop配置模式。通常使用边界扫描模式。

使用并口电缆下载FPGA配置文件到PROM时，主要工作过程时启动iMPACT，在文件模式中PROM Formatter把BIT转为MCS/EXO格式的PROM文件，然后在配置模式中初始化边界扫描链，以便iMPACT读入硬件连接信息，然后就可以下载配置文件。其中准备PROM文件包括两个步骤：一是建立边界扫描链，二是利用PROM Formatter把BIT文件分割或合并成PROM文件。

12、ChipScope Pro综述

ChipScope Pro的基本原理：利用FPGA中未用的BlockRam，根据用户设定的触发条件将信号实时的保存到这些BlockRam中，然后通过JTAG口传到计算机中，最后在计算机上显示时序波形。

13、结构视图（Technology View)

结构视图是将设计用FPGA/CPLD的硬件原语和底层模块描述的门级结构原理图，与RTL视图既相似又不同，结构视图对设计的描述更深刻，是RTL视图向具体器件进行结构映射的结果。

14、时序约束

时序约束包括周期约束（FFS TO FFS）、偏移约束（IPAD TO FFS, FFS TO OPAD）、静态路径约束（IPAD TO OPAD）。通过附加约束条件可以使综合布线工具调整映射和布局布线过程，使设计达到时序要求。附加时序约束的一般策略是先附加全局约束，然后对快速和慢速例外路径附加专门约束。附加全局约束时，先定义设计的所有时钟，然后对各时钟域内的同步元件进行分组，对分组附加周期约束，然后对输入输出PAD附加偏移约束、对全组合逻辑的PAD TO PAD路径附加约束。附加专门约束时，首先约束分组之间的路径，然后约

束快、慢速例外路径和多周期路径以及其它特殊路径。

设计内部电路所能达到的最高运行频率取决于同步元件本身的建立保持时间（这项一般很小，1各ns左右），以及同步元件之间的逻辑和布线延迟

Tclk=Tcko+Tnet+TlogicTsetup-Tclkskew；

原创文章

Testbench_vantage（itator）

时钟设计的革命（westor）

压缩分卷包 1　　压缩分卷包 2

（注：如果您有原创文章愿意在 EDACN 论坛发表，欢迎投稿。）
投稿信箱：monthly@edacn.net

Cyclone II FPGA 和 Nios II 嵌入式处理器具有低成本性能优势

在其业内领先的低成本Cyclone™ FPGA系列和Nios®软核嵌入式处理器成功的基础上，Altera现在推出了第二代产品系列。Cyclone II器件为用户提供更高的逻辑密度和新增硬件性能，比第一代产品成本降低了30%。这些新器件比同类竞争FPGA价格低50%，而速度却提高了50%。此外，Nios II软核嵌入式处理器比最初的Nios处理器功能更强大，而占用的逻辑单元（LE）更少。

ASIC和ASSP设计的局限性

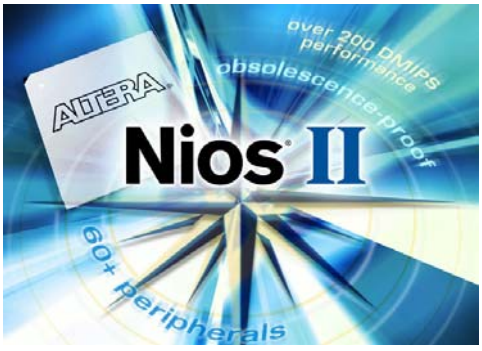
当今的设计工程师按照以往的惯例，完成符合性能和特性竞争要求的产品设计，但是同时也面临着更紧迫的预算和市场份额减少的压力。过去，采用ASIC或者合适的ASSP技术就可以实现满足高级性能需求、最具成本效益的大批量产品设计。ASIC技术曾经是实现最佳性能表现的最低成本途径，但是缺乏设计灵活性，而且产品面市时间较长，开发风险很高。另一方面，ASSP在产品及时面市和保证开发成功上风险较小，但仍旧没有可编程逻辑的灵活性。

现成的ASSP和微控制器具有逐渐过时的缺点。多数硬件嵌入式处理器随着时间的推移而逐渐过时，并最终被淘汰。结果，采用这种处理器的制造商不得不重新设计硬件，并重新编写软件，耗费了大量的时间和成本。由于采用了新的处理器系列，这种重新设计甚至要求一套全新的指令集，导致更多的时间和资金投入。Altera新一代Nios II等软核嵌入式处理器由于可以在未来的FPGA系列中继续使用，因此有助于改变这种系统逐渐过时的缺点，并确保终端用户的软件代码与多代硬件兼容。

Altera致力于为用户提供业内领先的可编程逻辑器件（PLD）、强大的嵌入式处理器、多种知识产权（IP）组合以及完善的开发工具套件，使用户能够根据特定系统要求，以最简单的方式，尽快生成符合他们需要的合适设计。同时，Altera特有的灵活性使用户能够轻松的扩展设计性能和特性来满足系统参数的改变，从而避免出现硬件过时的风险。

Cyclone II和Nios II的优势

Cyclone器件与第一代Nios处理器一起，实现了最具成本效益的片上可编程系统（SOPC）解决方案。Cyclone II FPGA和Nios II处理器将SOPC带入了更高的等级——使设计人员能够构建与中等密度ASIC和ASSP有效竞争的SOPC设计。这样，设计人员可以在大量的新应用中充分利用可编程逻辑的灵活性、产品及时面市以及成本优势来进行设计，而这些应用在以前完全被ASIC和ASSP技术所统治。Nios II/Cyclone II设计组合能够以每逻辑门$0.35的成本实现完整的嵌入式处理系统，其性能超过100 DMIPS。



Nios II系列包括：高性能内核(Nios II/f，“快速”)；低成本内核（Nios II/e，“经济”）；性能/成本均衡的内核（Nios II/s，“标准”）。这三个内核共享一套相同的32比特指令集体系（ISA），与二进制代码

百分之百兼容。它们具有单个相同的免版税许可，增强了设计灵活性，同时将成本降到最小。此外，所有内核都可以使用用户指令，使关键软件子程序在Nios II嵌入式处理器的操作控制下，能够在FPGA中运行。这些指令在硬件中运行时需要占有多个时钟周期，而在Nios II处理器应用中，最少只占用一个时钟周期，大大提高了系统性能和数据吞吐量。

设计流程

Altera认识到用户,特别是在消费类市场上,面临产品及时面市和产品较短生命周期的压力,因此,Altera开发了整套工具——包括SOPC Builder、Nios II集成设计环境(IDE)和Quartus II开发软件,帮助用户加速硬件和软件的开发,实现完整的基于可编程逻辑的SOPC解决方案。与此同时，具有Nios II许可的用户将收到含有软处理器内核和一套软件工具的开发工具包，用于在Altera® FPGA中进行Nios II设计。

SOPC Builder是在Altera FPGA中实施IP的关键工具。该系统级工具使用Altera的MegaWizard®技术自动生成Avalon的交换架构，将设计中的不同功能模块连接在一起。SOPC Builder还会生成定制软件开发工具，根据需要为由Nios II处理器控制的功能模块提供合适的软件头文件。采用SOPC Builder使设计人员从手动连接、验证寄存器和存储器映射体系结构等耗时的任务中解脱出来，从而能够将精力集中在如何优化关键系统功能上，因此开发时间会大大缩短。

Nios II IDE是Nios II软核处理器的主要开发工具，它为软件开发提供了一个集成的设计开发环境，包括一个具有工程管理、源代码开发、基于JTAG调试功能的图形用户界面（GUI），大大简化了大量复杂的Nios II处理器设计。其附加工具包括一个指令集模拟器（无需调试开发板，就可以进行代码调试）、MicroC/OS-II实时操作系统以及小型TCP/IP协议堆栈。

业内领先的Quartus II软件为硬件开发提供了设计采集、综合、仿真和布局布线功能。Quartus II软件具有无与伦比的性能等级和易用性，可帮助设计人员达到设计目的，尽快实现产品及时面市。Quartus II软件是最具成本效益的FPGA开发套件。

设计实例

因特网应用系统是最能够体现Cyclone II FPGA和Nios II处理器组合优势的设计实例。因特网应用包括视频游戏控制台、机顶盒、智能冰箱以及报警系统等。这类系统需要通过以太网进行通信和数据处理。它包括一些定制接口、必须的系统功能、胶合逻辑以及用户接口。

设计这类系统的传统方法是采用数字信号处理器、以太网MAC/PHY、微处理器和FPGA。这种设计方法可实现所有的系统功能，但是需要多个I/O引脚实现FPGA和数字信号处理器接口连接，使系统变得复杂，而且，这种设计方法至少需要四个组成部分，占用了大量的电路板资源。

Cyclone II器件系列和Nios II嵌入式处理器的出现极大地改变了这类系统的设计方法。例如，可以采用Nios II嵌入式处理器内核实现数字信号处理（DSP）功能，从而减少了以前所需元件的四分之一。对于更密集的DSP应用，设计人员可以利用Nios II处理器的定制指令能力在硬件中实现运行效率更高的某些功能。

如果需要更高级的DSP性能，可以采用Altera的IP内核与定制硬件的组合实现全部DSP功能。例如，Cyclone II器件中的嵌入式乘法器，可用来运行大量的数学指令，其速度甚至比最高端的数字信号处理器还要快。Nios II处理器可用于通常由数字信号处理器完成的所有控制功能。

由于Cyclone II器件的密度比第一代Cyclone器件增加了三倍，因此可以集成更多的功能，以提高系统集成度和可靠性，降低系统成本。Nios II处理器还具有性能强化、实施操作系统（RTOS）、中间件等优势，使其能够运行网络服务器，实现现场以太网控制和设备监控。

在同一个Cyclone II FPGA中还可以嵌入第二个Nios II内核，作为人机接口（MMI）处理器来控制小键盘和LCD驱动器。第二个处理器一般比数字信号处理器的速度要低一些，采用两个处理器可降低较慢的MMI处理器对DSP性能的影响。

结论

Nios II处理器和Cyclone II FPGA的出现为以前需要低、中密度ASIC来满足系统性能需求的设计人员提供了更多的选择。采用Altera强大的开发软件套件和丰富的IP组合，设计人员现在可以充分利用可编程逻辑灵活性和产品能够及时面市的优势，在Cyclone II FPGA中嵌入Nios II处理器，对成本敏感的应用进行SOPC设计。更多有关Altera Nios II处理器和Cyclone II FPGA的资料可以在www.altera.com.cn找到。

赛灵思新版开发工具支持更多 FPGA 系列器件

赛灵思公司(Xilinx, Inc)日前宣布推出最新版本的可免费下载的开发系统 ISE WebPACK 7.1i。新版工具支持 32 位版的 Red Hat Enterprise Linux 3 操作系统、赛灵思 Spartan-3、Virtex-4 和 CoolRunner-II 系列产品。通过提供新的可提升性能的功能，WebPACK 7.1i 可降低总体设计成本。

ISE WebPACK 7.1i 采用了 ISE 7.1 中的许多新功能，通过提高易用性极大改善了设计人员的使用体验。新的 Technology Viewer 功能提供了综合/优化后网表(post-synthesis/optimization netlists)的原理视图，使设计人员可以在设计周期中确保设计完整性，从而可加快设计开发速度并降低总体成本。新的 Design Summary 视图则允许设计人员在单个视图中即刻访问最有用的设计信息，同时在新的信息出现时立即进行信息更新。 Message Filtering 功能允许用户将输出和报告定制成只显示最感兴趣的消息。此外，新的 CableServer 功能支持通过网络对器件进行远程配置，从而使设计小组可以从多个不同地点对器件或电路板进行操作。新的 ModelSim Xilinx Edition-III 入门版是 Mentor Graphics 公司下一代 Model Technology 6.0a 系列 HDL 仿真工具的入门级产品。

最后，WebPACK 7.1i 中新包括的 XPower 工具为 CPLD 或 FPGA 设计提供了全面的功率分析功能。XPower 工具与赛灵思免费的 Web Power 工具集成在一起。

ISE WebPACK 7.1i 目前支持新推出的 Spartan-3E 系列(包括 XC3S100E-XC3S500E)中的部分器件。此外，ISE WebPACK 7.1i 还支持 Virtex-4 (XC4VLX15 和 XC4VLX25)、Spartan-3 (XC3S1000 和 XC3S1500)以及 Spartan-3L (XC3S1000L–XC3S1500L) FPGA 系列中的更多器件。

ISE WebPACK 7.1i 可在 Microsoft Windows 2000、Windows XP 以及目前的 32 位 Red Hat Enterprise Linux 3 操作系统上运行，支持所有赛灵思领先的 CPLD 系列产品，包括 CoolRunner-II 和 Spartan-IIE、Spartan-3、Spartan-3E、Virtex-II、Virtex-II Pro 和 Virtex-4 FPGA 系列中的部分器件。

资源共享

编者提示：欢迎各位网友推荐自己或者他人已经发布资料的信息。相关说明，请下载并参考“EDACN论坛资源共享推荐说明”。

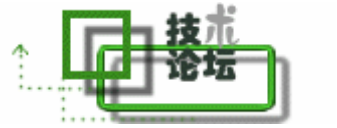
（郑重声明：本栏目为网友发布和获取资源共享信息提供方便而设置。由发布信息和相关内容引发的知识产权问题或者法律纠纷，与 EDACN 论坛无关。）

资料名称	HDL 编码风格与编码指南				
主题链接	http://www.edacn.net/cgi-bin/topic.cgi?forum=17&topic=1141&show=120				
类型	文档	大小	178k	售价	0 EDA 元
发布时间	2004/12/04	发布者	lichunjie	推荐度	★★★★★
简要说明					
作者徐欣、孙广富等,主要讲了编码注意问题及编码的经验.对新手具有很高的参考价值,新手开始就养成良好的编码风格对提高代码的可读性具有不可忽视的作用.					

资料名称	华为内部资料(硬件工程师手册)				
主题链接	http://www.edacn.net/cgi-bin/topic.cgi?forum=17&topic=451&show=0				
类型	文档	大小	885k	售价	4 EDA 元
发布时间	2004/04/13	发布者	gwxbdlj	推荐度	★★★★
简要说明					
硬件设计的基本知识,属于华为内部资料.适合新手熟悉硬件设计流程,对初学者具有一定的参考价值,唯一缺憾是资料不是最新版本.					

资料名称	148 个 verilog hdl 小程序（有很多 testbench）——适合初学者				
主题链接	http://www.edacn.net/cgi-bin/topic.cgi?forum=17&topic=833&show=285				
类型	文档	大小	36.5K	售价	2 EDA 元
发布时间	2004/08/11	发布者	abullience	推荐度	★★★
简要说明					
非常适合新手的学习资料。内部包含简单的 verilog 源文件与相应的 testbench 文件.因现在大公司基本上倾向与使用 verilog 语言,所以向网友推荐.					

资料名称	我是渔鱼,为人民服务!清华的 VHDL 讲义!!!!				
主题链接	http://www.edacn.net/cgi-bin/topic.cgi?forum=17&topic=1051&show=0				
类型	文档	大小	695k	售价	0 EDA 元
发布时间	2004/10/29	发布者	Api0930	推荐度	★★★
简要说明					
使用 VHDL 语言设计数字系统的流程与比较 design 过程比较 detail 的一些东东,值得新手一读.					



Make EDA Serve You !

欢迎登陆EDACN论坛： http://www.edacn.net