FPGA总结

特点及使用范围

CPU和GPU都属于冯·诺依曼结构，指令译码执行，共享内存。FPGA之所以比CPU、GPU更快，本质上是因为其无指令，无共享内存的体系结构所决定的。

冯氏结构中，由于执行单元可能执行任意指令，就需要有指令存储器、译码器、各种指令的运算器、分支跳转处理逻辑。而FPGA的每个逻辑单元的功能在重编程时就已经确定，不需要指令。

冯氏结构中使用内存有两种作用：①保存状态。②执行单元间的通信。

1）保存状态：FPGA中的寄存器和片上内存（BRAM）是属于各自的控制逻辑的，无需不必要的仲裁和缓存。

2）通信需求：FPGA每个逻辑单元与周围逻辑单元的连接在重编程时就已经确定了，并不需要通过共享内存来通信。

* 计算密集型任务中：

**在数据中心，FPGA相比GPU的核心优势在于延迟。**FPGA为什么比GPU的延迟低很多？本质上是体系结构的区别。FPGA同时拥有流水线并行和数据并行，而GPU几乎只有数据并行（流水线深度受限）。

Eg;处理一个数据包有10个步骤，FPGA可以搭建一个10级流水线，流水线的不同级在处理不同的数据包，每个数据包流经10级之后处理完成。每个处理完成的数据包可以马上输出。而GPU的数据并行方法是做10个计算单元，每个计算单元也在处理不同的数据包，但是所有的计算单元必须按照统一的步调，做相同的事情（SIMD）。这就要求10个数据包必须同进同出。当任务是逐个而非成批到达的时候，流水线并行比数据并行可实现更低的延迟。因此对流水式计算的任务，FPGA比GPU天生有延迟方面的优势。

ASIC在吞吐量、延迟、功耗单个方面都是最优秀的。但是其研发成本高，周期长。FPGA的灵活性可以保护资产。数据中心是租给不同租户使用的。有的机器上有神经网络加速卡，有的有bing搜索加速卡，有的有网络虚拟加速卡，任务的调度和运维会很麻烦。使用FPGA可以保持数据中心的同构性。

* 通信密集型任务中;

FPGA相比GPU、CPU的优势更大。

①吞吐量：FPGA可以直接接上40Gbps或者100Gbps的网线，以线速处理任意大小的数据包；而CPU则需要网卡把数据包接收过来；GPU也可以高性能处理数据包，但GPU没有网口，同样需要网卡，这样吞吐量受到网卡和（或）者CPU的限制。

②延迟:网卡把数据传给CPU，CPU处理后传给网卡，再加上系统中的时钟中断和任务调度增加了延迟的不稳定性。

在数据中心里 FPGA 的主要优势是稳定又极低的延迟，适用于流式的计算密集型任务和通信密集型任务。

FPGA 和 GPU 最大的区别在于体系结构，FPGA 更适合做需要低延迟的流式处理，GPU 更适合做大批量同构数据的处理。

成也萧何，败也萧何。缺少指令同时是 FPGA 的优势和软肋。每做一点不同的事情，就要占用一定的 FPGA 逻辑资源。如果要做的事情复杂、重复性不强，就会占用大量的逻辑资源，其中的大部分处于闲置状态。这时就不如用冯·诺依曼结构的处理器。

FPGA 和 CPU 协同工作，局部性和重复性强的归 FPGA，复杂的归 CPU。

DSP适用于系统较低取样速率、低数据率、多条件操作、处理复杂的多算法任务、使用C语言编程、系统使用浮点。）适合于较低采样速率下多条件进程、特别是复杂的多算法任务。

FPAG适用于系统高速取样速率（≥几MHZ）、高数据率、框图方式编程、处理任务固定或重复、使用定点。） 适合于高速采样频率下，特别是任务比较固定或重复的情况以及试制样机、系统开发的场合。

对于某些高主频的应用，FPGA就无能为力了。现在虽然理论上FPGA可以支持的500MHz，但在实际设计中，往往200MHz以上工作频率就很难实现了。

<https://www.zhihu.com/question/20298642>

CPU的主频可以达到GHz级，但一般的FPGA芯片的频率只能在百 MHz 级

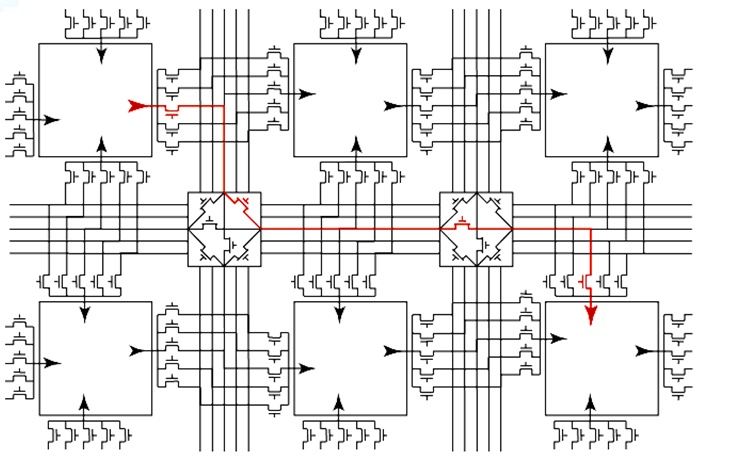
这是由FPGA的结构特点决定的：

1、芯片支持的频率高低是由内部寄存器与寄存器之间的走线延迟决定的；

2、FPGA是可编程器件，布线资源是要尽量设计成可通用，所以其走线延迟会比较大，也是速度上不去的原因，CPU是ANSIC ，其布线资源是定制的，所以可以更高频率；

3、现在有一些FPGA器件支持接口上的高速，内部通过串并转换到低速处理；

FPGA为了能保证内部Cell之间的连通性的，所以走线复杂，走线上的延迟较大



<http://blog.sciencenet.cn/blog-1225851-976759.html>

FPGA相对于CPU与GPU有明显的能耗优势，主要有两个原因。首先，在FPGA中没有Instruction Fetch与Instruction Decode，在Intel的CPU里面，由于使用的是CISC架构，仅仅Decoder就占整个芯片能耗的50%；在GPU里面，Fetch与Decode也消耗了10%～20%的能源。其次，FPGA的主频比CPU与GPU低很多，通常CPU与GPU都在1GHz到3GHz之间，而FPGA的主频一般在500MHz以下。如此大的频率差使得FPGA消耗的能源远低于CPU与GPU。

由于FPGA是可硬件编程的，相对于ASIC而言，使用FPGA可以对硬件逻辑进行迭代更新。但是FPGA也会被诟病，因为把算法写到FPGA硬件并不是一个容易的过程，相比在CPU与GPU上编程技术门槛高许多，开发周期也会长很多。

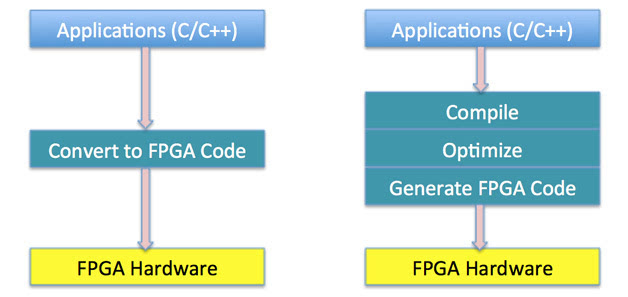


图4 传统FPGA开发流程与C-to-FPGA开发流程

图4显示了传统FPGA开发流程与C-to-FPGA开发流程的对比。在传统的FPGA开发流程中，我们需要把C/C++写成的算法逐行翻译成基于Verilog的硬件语言，然后再编译Verilog，把逻辑写入硬件。随着近几年FPGA技术的发展，从C直接编译到FPGA的技术已经逐渐成熟，并已在百度广泛被使用。在C-to-FPGA开发流程中，我们可以在CC++的代码中加Pragma, 指出哪个计算Kernel应该被加速，然后C-to-FPGA引擎会自动把代码编译成硬件。在我们的经验中，使用传统开发流程，完成一个项目大约需要半年时间，而使用了C-to-FPGA开发流程后，一个项目大约两周便可完成，效率提升了10倍以上。

用于深度神经网络加速

硬件结构及实现原理

FPGA同步和异步电路

在学习一门技术之前往往应该从它的编程语言入手，比如学习单片机时，往往从汇编或者C语言入门。所以不少开始接触FPGA的开发人员，往往是从VHDL或者Verilog开始入手学习的。但小编认为，若能先结合《数字电路基础》系统学习各种74系列逻辑电路，深刻理解逻辑功能，对于学习HDL语言大有裨益，往往会起到事半功倍的效果。

当然，任何编程语言的学习都不是一朝一夕的事，经验技巧的积累都是在点滴中完成，FPGA设计也无例外。下面继续分享一些FPGA经验。

FPGA中的Block RAM：

3种块RAM结构，M512 RAM(512bit)、M4K RAM(4Kbit)、M-RAM(64Kbit)。

M512 RAM：适合做一些小的Buffer、FIFO、DPRAM、SPRAM、ROM等;

M4K RAM： 适用于一般的需求;

M-RAM： 适合做大块数据的缓冲区。

Xlinx 和 LatTIce FPGA的LUT可以灵活配置成小的RAM、ROM、FIFO等存储结构，这种技术被称为分布式RAM。

补充：但是在一般的设计中，不提倡用FPGA/CPLD的片内资源配置成大量的存储器，这是处于成本的考虑。所以尽量采用外接存储器。

善用芯片内部的PLL或DLL资源完成时钟的分频、倍频率、移相等操作不仅简化了设计，并且能有效地提高系统的精度和工作稳定性。

**异步电路和同步时序电路的区别。**

异步电路：

电路核心逻辑有用组合电路实现;

异步时序电路的最大缺点是容易产生毛刺;

不利于器件移植;

不利于静态时序分析(STA)、验证设计时序性能。

同步时序电路：

电路核心逻辑是用各种触发器实现;

电路主要信号、输出信号等都是在某个时钟沿驱动触发器产生的;

同步时序电路可以很好的避免毛刺;

利于器件移植;

利于静态时序分析(STA)、验证设计时序性能。

同步设计中，稳定可靠的数据采样必须遵从以下两个基本原则：

(1)在有效时钟沿到达前，数据输入至少已经稳定了采样寄存器的Setup时间之久，这条原则简称满足Setup时间原则;

(2)在有效时钟沿到达后，数据输入至少还将稳定保持采样寄存器的Hold时钟之久，这条原则简称满足Hold时间原则。

FPGA常用设计思想与基本模块划分

常用设计思想与技巧

(1)乒乓操作;

(2)串并转换;

(3)流水线操作;

(4)异步时钟域数据同步。是指如何在两个时钟不同步的数据域之间可靠地进行数据交换的问题。数据时钟域不同步主要有两种情况：

①两个域的时钟频率相同，但是相差不固定，或者相差固定但是不可测，简称为同频异相问题。

②两个时钟频率根本不同，简称异频问题。

两种不推荐的异步时钟域操作方法：一种是通过增加Buffer或者其他门延时来调整采样;另一种是盲目使用时钟正负沿调整数据采样。

模块划分基本原则

(1)对每个同步时序设计的子模块的输出使用寄存器(用寄存器分割同步时序模块原则);

(2)将相关逻辑和可以复用的逻辑划分在同一模块内(呼应系统原则);

(3)将不同优化目标的逻辑分开;

(4)将送约束的逻辑归到同一模块;

(5)将存储逻辑独立划分成模块;

(6)合适的模块规模;

(7)顶层模块最好不进行逻辑设计。

同步时序设计时值得注意的事项

<https://blog.csdn.net/woshifennu1234/article/details/81162203>

**同步时序设计时一下事项应值得注意：**

* 异步时钟域的数据转换。
* 组合逻辑电路的设计方法。
* 同步时序电路的时钟设计。
* 同步时序电路的延迟。同步时序电路的延迟最常用的设计方法是用分频或者倍频的时钟或者同步计数器完成所需的延迟，对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，根据计数产生延迟;对于比较小的延迟，可以用D触发器打一下，这样不仅可以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。

另外，还有用行为级方法描述延迟，如“#5 a<=4’0101;”这种常用于仿真测试激励，但是在电路综合时会被忽略，并不能起到延迟作用。

Verilog 定义的reg型，不一定综合成寄存器。在Verilog代码中最常用的两种数据类型是wire和reg型，一般来说，wire型指定的数据和网线通过组合逻辑实现，而reg型指定的数据不一定就是用寄存器实现。

简述组合逻辑的注意事项（??）

(1)避免组合逻辑反馈环路(容易毛刺、振荡、时序违规等)。

解决：A.牢记任何反馈回路必须包含寄存器;B.检查综合、实现报告的warning信息，发现反馈回路(combinaTIonal loops)后进行相应修改。

(2)替换延迟链。

解决：用倍频、分频或者同步计数器完成。

(3)替换异步脉冲产生单元(毛刺生成器)。

解决：用同步时序设计脉冲电路。

(4)慎用锁存器。

解决方式：

A、使用完备的if…else语句;

B、检查设计中是否含有组合逻辑反馈环路;

C、对每个输入条件，设计输出操作，对case语句设置default 操作。特别是在状态机设计中，最好有一个default的状态转移，而且每个状态最好也有一个default的操作。

D、如果使用case语句时，特别是在设计状态机时，尽量附加综合约束属性，综合为完全条件case语句。

小技巧：仔细检查综合器的综合报告，目前大多数的综合器对所综合出的latch都会报“warning”，通过综合报告可以较为方便地找出无意中生成的latch。

FPGA设计者必须要精通的5项基本功

FPGA设计者的5项基本功：仿真、综合、时序分析、调试、验证。

对于FPGA设计者来说，练好这5项基本功，与用好相应的EDA工具是同一过程，对应关系如下：

1. 仿真：Modelsim， Quartus II(Simulator Tool)

2. 综合：Quartus II (Compiler Tool， RTL Viewer， Technology Map Viewer， Chip Planner)

3. 时序：Quartus II (TImeQuest Timing Analyzer， Technology Map Viewer， Chip Planner)

4. 调试：Quartus II (SignalTap II Logic Analyzer， Virtual JTAG， Assignment Editor)

5. 验证：Modelsim， Quartus II(Test Bench Template Writer)

掌握HDL语言虽然不是FPGA设计的全部，但是HDL语言对FPGA设计的影响贯穿于整个FPGA设计流程中，与FPGA设计的5项基本功是相辅相成的。

对于FPGA设计者来说，用好“HDL语言的可综合子集”可以完成FPGA设计50%的工作——设计编码。

练好仿真、综合、时序分析这3项基本功，对于学习“HDL语言的可综合子集”有如下帮助：

通过仿真，可以观察HDL语言在FPGA中的逻辑行为。

通过综合，可以观察HDL语言在FPGA中的物理实现形式。

通过时序分析，可以分析HDL语言在FPGA中的物理实现特性。

对于FPGA设计者来说，用好“HDL语言的验证子集”，可以完成FPGA设计另外50%的工作——调试验证。

1. 搭建验证环境，通过仿真的手段可以检验FPGA设计的正确性。

2. 全面的仿真验证可以减少FPGA硬件调试的工作量。

3. 把硬件调试与仿真验证方法结合起来，用调试解决仿真未验证的问题，用仿真保证已经解决的问题不在调试中再现，可以建立一个回归验证流程，有助于FPGA设计项目的维护。

FPGA 设计者的这5项基本功不是孤立的，必须结合使用，才能完成一个完整的FPGA设计流程。反过来说，通过完成一个完整的设计流程，才能最有效地练习这5项基本功。对这5项基本功有了初步认识，就可以逐个深入学习一些，然后把学到的知识再次用于完整的设计流程。如此反复，就可以逐步提高设计水平。采用这样的循序渐进、螺旋式上升的方法，只要通过培训入了门，就可以自学自练，自我提高。

市面上出售的有关FPGA设计的书籍为了保证结构的完整性，对 FPGA设计的每一个方面分开介绍，每一方面虽然深入，但是由于缺少其他相关方面的支持，读者很难付诸实践，只有通读完全书才能对FPGA设计获得一个整体的认识。这样的书籍，作为工程培训指导书不行，可以作为某一个方面进阶的参考书。

建立模型，化简逻辑

看逻辑，建模型。

只有在脑海中建立了一个个逻辑模型，理解FPGA内部逻辑结构实现的基础，才能明白为什么写Verilog和写C整体思路是不一样的，才能理解顺序执行语言和并行执行语言的设计方法上的差异。在看到一段简单逻辑的时候应该想到是什么样的功能电路。

用数学思维来简化设计逻辑。

学习FPGA不仅逻辑思维很重要，好的数学思维也能让你的设计化繁为简，所以，看见高数就头疼的同学需要重视一下这门课。举个简单的例子，比如有两个32bit的数据X[31:0]与Y[31:0]相乘。当然，无论Altera还是Xilinx都有现成的乘法器IP核可以调用，这也是最简单的方法，但是两个32bit的乘法器将耗费大量的资源。那么有没有节省资源，又不太复杂的方式来实现呢?我们可以稍做修改：

将X[31:0]拆成两部分X1[15:0]和X2[15:0]，令X1[15:0]=X[31:16]，X2[15:0]=X[15:0]，则X1左移16位后与X2相加可以得到X;同样将Y[31:0]拆成两部分Y1[15:0]和Y2[15:0]，令 Y1[15:0]=Y[31:16]，Y2[15:0]=Y[15:0]，则Y1左移16位后与Y2相加可以得到Y;则X与Y的相乘可以转化为X1和X2 分别与Y1和Y2相乘

这样一个32bit\*32bit的乘法运算转换成了四个16bit\*16bit的乘法运算和三个32bit的加法运算。转换后的占用资源将会减少很多，有兴趣的读者，不妨综合一下看看，看看两者差多少。

关于FPGA供电

**FPGA是一种多电源需求的芯片，主要有3种电源需求：**

* VCCINT：核心工作电压，PCI Express (PCIe) 硬核IP 模块和收发器物理编码子层(PCS) 电源。一般电压都很低，目前常用的FPGA都在1.2V左右。为FPGA的内部各种逻辑供电，电流从几百毫安到几安不等，具体取决于内部逻辑的工作时钟速率以及所占用的逻辑资源。对于这个电源来说，负载时一个高度容性阻抗，对电源的瞬态响应要求很高，而且由于驱动电压低工作电流大，对PCB的布线电阻非常敏感，需要特别注意走线宽度，尽可能减少布线电阻带来的损耗。
* VCCA：通常为2.5V，PLL模拟电源。即使没有PLL，也必须要上电。模拟类的组件对电源的电源抑制比（PSRR）也就是电源噪声，或者说电源纹波非常敏感，所以通常会用一个独立的供电电源。这个电源的电流需求一般都不大，但对电源的噪声容忍度很低。所以应该尽可能的提高其电源纯净度。比如不直接用开关电源供电，先使用LDO稳压后再供给VCCA。
* VCCD\_PLL：通常为1.2V，PLL数字电源。
* VCCIO：FPGA经常要与多种不同电平接口的芯片通信，所以通常都会支持非常多的电平标准。例如1.2，1.5，1.8，2.5，3.0，3.3。VCCIO就是为FPGA的I/O驱动逻辑供电。FPGA为了同时能和多种不同的电平标准接口芯片通信，Vcco通常以BANK为界，互相之间相互独立，也就是说在一颗FPGA芯片上同时存在几种不同的I/O电压。当然同一个BANK只能存在1种I/O电压。在使用中请详细阅读官方资料手册，以防设计错误。

在一些带收发器的FPGA器件中，还有以下几种电压标准：

* VCC\_CLKIN：支持1.2，1.5，1.8，2.5，3.0，3.3。主要为差分时钟输入管脚供电。
* VCCH\_GXB：2.5V，收发器输出(TX) 缓冲器电源。
* VCCA\_GXB：2.5V，收发器物理介质附加子层(PMA) 及辅助电源。
* VCCL\_GXB：1.2V，收发器PMA 及辅助电源。

关于FPGA电源精度要求

**FPGA对DC-DC精度的要求不断提升**

FPGA厂商不断采用更先进的工艺来降低器件功耗，提高性能，同时FPGA对供电电源的精度要求也越加苛刻，电压必须维持在非常严格的容限内，如果供电电压范围超出了规范的要求，就有会影响到FPGA的可靠性，甚至导致FPGA失效。

无论是Intel （Altera）FPGA还是Xilinx FPGA均在数据手册中明确提出了电源精度要求，其中要求最高的是内核和高速收发器的供电。举例来看，Intel公司的Cyclone V、Cyclone 10 GX、Arria10、Stratix 10的电源精度要求在±30mV以内。

新一代FPGA的供电精度都在±20-30mv左右，已经是单板中对电源精度要求最为苛刻的器件之一了。

由于输出精度都是理论计算值，并没有考虑单板PCB布线和其他外部设备引入的干扰和误差，因此实际设计产品时，电源输出精度不但必须符合数据手册中的要求，还必须预留一定的余量，通常设计中，我们还会保留50%-100%余量，以保证系统长期可靠工作。

**电源的稳态直流精度及计算方法**

供电电源的稳态直流精度主要取决于两个因素：电压调整精度和输出电压纹波。这里有一个误区，很多工程师只通过DC-DC数据手册上的电压输出精度来判断器件是否符合要求，其实这是不正确的。 首先很多DC-DC需要外部反馈电阻来决定最终的输出电压，数据手册上的电压调整精度是指芯片本身的输出精度，并没有计算反馈电路引入的偏差。其次，器件数据手册上的电压输出精度并不包含输出电压纹波，必须将两者叠加计算才能得到正确的直流稳态精度。

正确的电源稳态直流精度的计算公式如下：

电源直流稳态精度 =器件输出精度（这里要求全温度，全负载时的精度，很多器件手册只给出典型值，因此要小心)+ ½ 纹波 + 外部反馈电阻精度引入的误差。

**高精度电源对减低FPGA功耗的作用**

除了电源精度影响整个系统的稳定性和可靠性，更高精度的电源还可以帮助我们降低系统功耗。

重要的时钟树（??）

对于FPGA来说，要尽可能避免异步设计，尽可能采用同步设计。同步设计的第一个关键，也是关键中的关键，就是时钟树。一个糟糕的时钟树，对FPGA设计来说，是一场无法弥补的灾难，是一个没有打好地基的大楼，崩溃是必然的。

具体设计细则：

1）尽可能采用单一时钟；

2）如果有多个时钟域，一定要仔细划分，千万小心；

3）跨时钟域的信号一定要做同步处理。对于控制信号，可以采用双采样；对于数据信号，可以采用异步fifo。需要注意的是，异步fifo不是万能的，一个异步fifo也只能解决一定范围内的频差问题。

4）尽可能将FPGA内部的PLL、DLL利用起来，这会给你的设计带来大量的好处。

5）对于特殊的IO接口，需要仔细计算Tsu、Tco、Th，并利用PLL、DLL、DDIO、管脚可设置的delay等多种工具来实现。简单对管脚进行Tsu、Tco、Th的约束往往是不行的。

可能说的不是很确切。这里的时钟树实际上泛指时钟方案，主要是时钟域和PLL等的规划，一般情况下不牵扯到走线时延的详细计算（一般都走全局时钟网络和局部时钟网络，时延固定），和ASIC中的时钟树不一样。对于ASIC，就必须对时钟网络的设计、布线、时延计算进行仔细的分析计算才行。

关于锁存器，应该要知道(??)

锁存器latch，在数字电路中经常遇到，它和触发器FF有着本质的区别。

这里为读者介绍FPGA中避免锁存器的方法。

* 在组合逻辑进程中，if语句一定要有else。并且所有的信号都要在if的所有分支中被赋值。

always @（ \* ）

begin

  if （ sig\_a == 1‘b1 ）

   sig\_b = sig\_c;

end

这个是绝对会产生latch的。

这种情况要想避免锁存器的出现，应当为if语句增加else语句。

* 下面也会产生latch。也就是说在组合逻辑进程中不能出现自己赋值给自己或者间接出现自己赋值给自己的情况。

    always @（ \* ）

    begin

      if （ rst == 1‘b1 ）

         counter = 32’h00000000;

    else

         counter = counter + 1;

     end

但如果是时序逻辑进程，则不存在该问题。

* case语句的default一定不能少！

原因和if语句相同，这里不再多说了。

需要提醒的是，在时序逻辑进程中，default语句也一定要加上，这是一个很好的习惯。

* 组合逻辑进程敏感变量不能少也不能多。

verilog2001语法中可以直接用 \* 。latch有弊就一定有利。在FPGA的LE中，总存在一个latch和一个D触发器，在支持DDR的IOE（IOB）中也存在着一个latch来实现DDIO。在我们平时的设计中，对latch还是要尽可能的敬而远之。

FPGA GPU CPU比较

分类： [学习笔记](https://blog.csdn.net/m0_37666899/article/category/7145439)

CPU和GPU都属于冯·诺依曼结构，指令译码执行，共享内存。FPGA之所以比CPU、GPU更快，本质上是因为其无指令，无共享内存的体系结构所决定的。

冯氏结构中，由于执行单元可能执行任意指令，就需要有指令存储器、译码器、各种指令的运算器、分支跳转处理逻辑。而FPGA的每个逻辑单元的功能在重编程时就已经确定，不需要指令。

冯氏结构中使用内存有两种作用：①保存状态。②执行单元间的通信。

1）保存状态：FPGA中的寄存器和片上内存（BRAM）是属于各自的控制逻辑的，无需不必要的仲裁和缓存。

2）通信需求：FPGA每个逻辑单元与周围逻辑单元的连接在重编程时就已经确定了，并不需要通过共享内存来通信。

* 计算密集型任务中：

在数据中心，FPGA相比GPU的核心优势在于延迟。FPGA为什么比GPU的延迟低很多？本质上是体系结构的区别。FPGA同时拥有流水线并行和数据并行，而GPU几乎只有数据并行（流水线深度受限）。

Eg;处理一个数据包有10个步骤，FPGA可以搭建一个10级流水线，流水线的不同级在处理不同的数据包，每个数据包流经10级之后处理完成。每个处理完成的数据包可以马上输出。而GPU的数据并行方法是做10个计算单元，每个计算单元也在处理不同的数据包，但是所有的计算单元必须按照统一的步调，做相同的事情（SIMD）。这就要求10个数据包必须同进同出。当任务是逐个而非成批到达的时候，流水线并行比数据并行可实现更低的延迟。因此对流水式计算的任务，FPGA比GPU天生有延迟方面的优势。

ASIC在吞吐量、延迟、功耗单个方面都是最优秀的。但是其研发成本高，周期长。FPGA的灵活性可以保护资产。数据中心是租给不同租户使用的。有的机器上有神经网络加速卡，有的有bing搜索加速卡，有的有网络虚拟加速卡，任务的调度和运维会很麻烦。使用FPGA可以保持数据中心的同构性。

* 通信密集型任务中;

FPGA相比GPU、CPU的优势更大。

①吞吐量：FPGA可以直接接上40Gbps或者100Gbps的网线，以线速处理任意大小的数据包；而CPU则需要网卡把数据包接收过来；GPU也可以高性能处理数据包，但GPU没有网口，同样需要网卡，这样吞吐量受到网卡和（或）者CPU的限制。

②延迟:网卡把数据传给CPU，CPU处理后传给网卡，再加上系统中的时钟中断和任务调度增加了延迟的不稳定性。

在数据中心里 FPGA 的主要优势是稳定又极低的延迟，适用于流式的计算密集型任务和通信密集型任务。

FPGA 和 GPU 最大的区别在于体系结构，FPGA 更适合做需要低延迟的流式处理，GPU 更适合做大批量同构数据的处理。

成也萧何，败也萧何。缺少指令同时是 FPGA 的优势和软肋。每做一点不同的事情，就要占用一定的 FPGA 逻辑资源。如果要做的事情复杂、重复性不强，就会占用大量的逻辑资源，其中的大部分处于闲置状态。这时就不如用冯·诺依曼结构的处理器。

FPGA 和 CPU 协同工作，局部性和重复性强的归 FPGA，复杂的归 CPU。

FPGA建立时间和保持时间详解

时钟是FPGA设计中最重要的信号，FPGA系统内大部分器件的动作都是在时钟的上升沿或者下降沿进行。无论是在输入，输出或是寄存器与寄存器之间，只要设计到时钟上升沿或者下降沿的采样，就会提到建立时间（setup time） 和保持时间（hold time） 。

建立时间（Tsu：set up time）是指在时钟沿到来之前数据从不稳定到稳定所需的时间，如果建立的时间不满足要求那么数据将不能在这个时钟上升沿被稳定的打入触发器；

保持时间（Th：hold time）是指数据稳定后保持的时间，如果保持时间不满足要求那么数据同样也不能被稳定的打入触发器。

建立时间和保持时间这两个指标说明器件本身不是理想的（有时延等），正是这个不理想的特性，限制了FPGA的时钟工作频率。

首先我们都知道setup time 和holdup time是由器件决定的，并不是说可以随着你FPGA设计的改变而改变。那么FPGA时钟频率是怎么计算的呢，在不考虑时钟延时抖动等条件下，一个信号从触发器的D端到Q端的延时假设是Tcd，从Q端出来之后会经过组合电路延时，这里注意即使没有组合电路，就单单经过导线也是有延时的，这个延时称作Tdelay，经过这个延时之后，信号将要去下一个触发器，而且必须要满足触发器的建立时间tsetup，不然时钟无法采样到稳定的数据。所以这三个时间加起来应该比时钟周期要小，否则数据无法打入下一个触发器，那就会进入亚稳态。Tcd+Tdelay+Tsetup<T，时钟频率f=1/T，周期越短频率越高，那么最短周期是什么呢就显而易见了。上面式子中Tcd 和Tsetup 都是由器件本身决定的，我们唯一能减小的就是Tdelay，在电路中Tdelay有无数条，有长有短，而那个最长的路径（关键路径）直接决定了FPGA时钟能跑多快，这就是为什么我们做时序优化总是要从关键路径下手。

至于FPGA时钟频率与holdup time的关系，具体做设计的时候还是需要满足Tcd+Tdelay+Tsetup<T，所以Tsetup决定了最长路径的上限。而Tcd+Tdelay>Tholdup，也就是说Tholdup 决定了最短路径的下限，也就是说组合逻辑是不能太大也不能太小的。这就是hold time 能起作用的地方吧。其实一般都能满足保持时间，一般只要考虑都是要满足建立时间。

建立时间与保持时间的简单示意图如下图1所示，在图1中我们看到clk\_r3的前后各有一条虚线，前一条虚线（最左边的虚线，左边代表出现时间早，与modelsim仿真时信号依次从左往右出现）到clk\_r3上升沿的这段时间即为建立时间，clk\_r3上升沿到后一条虚线（最右边的虚线）的这段时间即为保持时间。前面对建立时间和保持时间下定义时提到过，在这段时间内不能够有数据的变化，数据必须保持稳定。而在这个波形中，也确实没有看到在建立时间和保持时间内，reg3in的数据有任何的变化，因此我们可以稳定的将reg3in的数据锁存到reg3的输出reg3out中。

FPGA设计重点与难点经验总结（??）

做过FPGA设计的朋友肯定知道，特别是对于大的设计（无论软件还是硬件），不按照规范走几乎是不可实现的。逻辑设计也是这样：如果不按规范做的话，过一个月后调试时发现有错，回头再看自己写的代码，估计很多信号功能都忘了，更不要说检错了；如果一个项目做了一半一个人走了，接班的估计得从头开始设计；如果需要在原来的版本基础上增加新功能，很可能也得从头来过，很难做到设计的可重用性。在逻辑方面，我觉得比较重要的规范有这些：



1.设计必须文档化。要将设计思路，详细实现等写入文档，然后经过严格评审通过后才能进行下一步的工作。这样做乍看起来很花时间，但是从整个项目过程来看，绝对要比一上来就写代码要节约时间，且这种做法可以使项目处于可控、可实现的状态。

2.代码规范。如果在另一个设计中的时钟是40ns，复位周期不变，我们只需对CLK\_PERIOD进行重新例化就行了，从而使得代码更加易于重用。

3.信号命名要规范化。

a. 信号名一律小写，参数用大写。

b.对于低电平有效的信号结尾要用\_n标记，如rst\_n。

c.端口信号排列要统一，一个信号只占一行，最好按输入输出及从哪个模块来到哪个模块去的关系排列，这样在后期仿真验证找错时后方便很多。

d.一个模块尽量只用一个时钟，这里的一个模块是指一个module或者是一个entity。在多时钟域的设计中涉及到跨时钟域的设计中最好有专门一个模块做时钟域的隔离。这样做可以让综合器综合出更优的结果。

e.尽量在底层模块上做逻辑，在高层尽量做例化，顶层模块只能做例化，禁止出现任何胶连逻辑（glue logic），哪怕仅仅是对某个信号取反。理由同上。

f.在FPGA的设计上禁止用纯组合逻辑产生latch，带D触发器的latch的是允许的，比如配置寄存器就是这种类型。

g. 一般来说，进入FPGA的信号必须先同步，以提高系统工作频率（板级）。

h.所有模块的输出都要寄存器化，以提高工作频率，这对设计做到时序收敛也是极有好处的。

i.除非是低功耗设计，不然不要用门控时钟，这会增加设计的不稳定性，在要用到门控时钟的地方，也要将门控信号用时钟的下降沿打一拍再输出与时钟相与。

j.禁止用计数器分频后的信号做其它模块的时钟，而要用改成时钟使能的方式，否则这种时钟满天飞的方式对设计的可靠性极为不利，也大大增加了静态时序分析的复杂性。如FPGA的输入时钟是25M的，现在系统内部要通过RS232与PC通信，要以rs232\_1xclk的速率发送数据。

时序是设计出来的

时序是设计出来的，不是仿出来的，更不是湊出来的。

如何提高电路工作频率

对于设计者来说，当然希望我们设计的电路的工作频率（在这里如无特别说明，工作频率指FPGA片内的工作频率）尽量高。我们也经常听说用资源换速度，用流水的方式可以提高工作频率，这确实是一个很重要的方法，今天我想进一步去分析该如何提高电路的工作频率。

先来分析下是什么影响了电路的工作频率。

电路的工作频率主要与寄存器到寄存器之间的信号传播时延及clock skew有关。在FPGA内部如果时钟走长线的话，clockskew很小，基本上可以忽略， 在这里为了简单起见，只考虑信号的传播时延的因素。信号的传播时延包括寄存器的开关时延、走线时延、经过组合逻辑的时延（这样划分或许不是很准确，不过对分析问题来说应该是没有问题的），要提高电路的工作频率，就要在这三个时延中做文章，使其尽可能的小。先来看开关时延，这个时延是由器件物理特性决定的，没有办法去改变，所以只能通过改变走线方式和减少组合逻辑的方法来提高工作频率。

1.通过改变走线的方式减少时延。

以 Altera的器件为例，在quartus里面的timing closure floorplan 可以看到有很多条条块块，我们可以将条条块块按行和按列分，每一个条块代表1个LAB，每个LAB里有8个或者是10个LE。它们的走线时延的关系如下：同一个LAB中（最快） 同列或者同行 不同行且不同列。

通过给综合器加适当的约束（不可贪心，一般以加5%裕量较为合适，比如电路工作在100Mhz，则加约束加到105Mhz就可以了，贪心效果反而不好，且极大增加综合时间）可以将相关的逻辑在布线时尽量布的靠近一点，从而减少走线的时延。（注：约束的实现不完全是通过改进布局布线方式去提高工作频率，还有其它的改进措施）

2.通过减少组合逻辑的减少时延。

上面讲了可以通过加约束来提高工作频率，但是在做设计之初可万万不可将提高工作频率的美好愿望寄托在加约束上，我们要通过合理的设计去避免出现大的组合逻辑，从而提高电路的工作频率，这才能增强设计的可移植性，才可以使得设计在移植到另一同等速度级别的芯片时还能使用。

我们知道，目前大部分FPGA都基于4输入LUT的，如果一个输出对应的判断条件大于四输入的话就要由多个LUT级联才能完成，这样就引入一级组合逻辑时延，我们要减少组合逻辑，无非就是要输入条件尽可能的少，这样就可以级联的LUT更少，从而减少了组合逻辑引起的时延。

平时听说的流水就是一种通过切割大的组合逻辑（在其中插入一级或多级D触发器，从而使寄存器与寄存器之间的组合逻辑减少）来提高工作频率的方法。比如一个32位的计数器，该计数器的进位链很长，必然会降低工作频率，我们可以将其分割成4位和8位的计数，每当4位的计数器计到15后触发一次8位的计数器，这样就实现了计数器的切割，也提高了工作频率。

在状态机中，一般也要将大的计数器移到状态机外，因为计数器这东西一般是经常是大于4输入的，如果再和其它条件一起做为状态的跳变判据的话，必然会增加LUT的级联，从而增大组合逻辑。以一个6输入的计数器为例，我们原希望当计数器计到111100后状态跳变，现在我们将计数器放到状态机外，当计数器计到111011后产生个enable信号去触发状态跳变，这样就将组合逻辑减少了。

上面说的都是可以通过流水的方式切割组合逻辑的情况，但是有些情况下我们是很难去切割组合逻辑的，在这些情况下又该怎么做呢？

状态机就是这么一个例子，我们不能通过往状态译码组合逻辑中加入流水。如果我们的设计中有一个几十个状态的状态机，它的状态译码逻辑将非常之巨大，毫无疑问，这极有可能是设计中的关键路径。那该怎么做呢？还是老思路，减少组合逻辑。我们可以对状态的输出进行分析，对它们进行重新分类，并根据这个重新定义成一组组小状态机，通过对输入进行选择（case语句）并去触发相应的小状态机，从而实现了将大的状态机切割成小的状态机。在ATA6的规范中（硬盘的标准），输入的命令大概有20十种，每一个命令又对应很多种状态，如果用一个大的状态机（状态套状态）去做那是不可想象的，可以通过case语句去对命令进行译码，并触发相应的状态机，这样做下来这一个模块的频率就可以跑得比较高了。

总结：提高工作频率的本质就是要减少寄存器到寄存器的时延，最有效的方法就是避免出现大的组合逻辑，也就是要尽量去满足四输入的条件，减少LUT级联的数量。我们可以通过加约束、流水、切割状态的方法提高工作频率。

做逻辑的难点在于系统结构设计和仿真验证

做逻辑的难点不在于RTL级代码的设计，而在于系统结构设计和仿真验证方面。目前国内对可综合的设计强调的比较多，而对系统结构设计和仿真验证方面似乎还没有什么资料，这或许也从一个侧面反映了国内目前的设计水平还比较低下吧。以前在学校的时候，总是觉得将RTL级代码做好就行了，仿真验证只是形式而已，所以对HDL的行为描述方面的语法不屑一顾，对testbench也一直不愿意去学--因为觉得画波形图方便；对于系统结构设计更是一点都不懂了。到了公司接触了些东西才发现完全不是这样。

其实在国外，花在仿真验证上的时间和人力大概是花在RTL级代码上的两倍，现在仿真验证才是百万门级芯片设计的关键路径。

仿真验证的难点主要在于怎么建模才能完全和准确地去验证设计的正确性（主要是提高代码覆盖），在这过程中，验证速度也是很重要的。

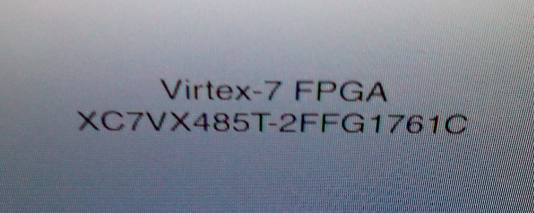
验证说白了也就是怎么产生足够覆盖率的激励源，然后怎么去检测错误。我个人认为，在仿真验证中，最基本就是要做到验证的自动化。这也是为什么我们要写testbench的原因。在我现在的一个设计中，每次跑仿真都要一个小时左右（这其实算小设计）由于画波形图无法做到验证自动化，如果用通过画波形图来仿真的话，一是画波形会画死（特别是对于算法复杂的、输入呈统计分布的设计），二是看波形图要看死，三是检错率几乎为零。那么怎么做到自动化呢？我个人的水平还很有限，只能简单地谈下BFM（bus function model，总线功能模型）。

以做一个MAC的core为例（背板是PCI总线），那么我们需要一个MAC\_BFM和PCI\_BFM及PCI\_BM（PCI behavior model）。MAC\_BFM的主要功能是产生以太网帧（激励源），随机的长度和帧头，内容也是随机的，在发送的同时也将其复制一份到PCI\_BM中；PCI\_BFM的功能则是仿PCI总线的行为，比如被测收到了一个正确帧后会向PCI总线发送一个请求，PCI\_BFM则会去响应它，并将数据收进来；PCI\_BM的主要功能是将MAC\_BFM发送出来的东西与PCI\_BFM接收到的东西做比较，由于它具有了MAC\_BFM的发送信息和PCI\_BFM的接收信息，只要设计合理，它总是可以自动地、完全地去测试被测是否工作正常，从而实现自动检测。 华为在仿真验证方面估计在国内来说是做的比较好的，他们已建立起了比较好的验证平台，大部分与通信有关的BFM都做好了，听我朋友说，现在他们只需要将被测放在测试平台中，并配置好参数，就可以自动地检测被测功能的正确与否。

在功能仿真做完后，由于我们做在是FPGA的设计，在设计时已经基本保证RTL级代码在综合结果和功能仿真结果的一致性，只要综合布局布线后的静态时序报告没有违反时序约束的警告，就可以下到板子上去调试了。事实上，在华为中兴，他们做FPGA的设计时也是不做时序仿真的，因为做时序仿真很花时间，且效果也不见得比看静态时序分析报告好。

当然了，如果是ASIC的设计话，它们的仿真验证的工作量要大一些，在涉及到多时钟域的设计时，一般还是做后仿的。不过在做后仿之前，也一般会先用形式验证工具和通过静态时序分序报告去查看有没有违反设计要求的地方，这样做了之后，后仿的工作量可以小很多。

FPGA芯片命名规则解释



XC7VX485T是芯片型号，表示属于Xilinx公司的V7系列的芯片，485T表示其有48.5万个逻辑单元。

-2表示速度等级，对于Xilinx FPGA 来说，一般有-1，-2，-3三个等级，值越大，速度越高。

FFG表示封装方式

1761表示引脚数

C代表的是温度等级Temperature grade ，这里是商用(Commercial)，如果是I 就是工业用。

同一款芯片可以有多个速度等级，不同的速度等级代表着不同的性能，不同的性能又导致芯片价格的巨大差异。芯片的速度等级不是设计出来的，而是在芯片生产出来之后，实际测试标定出来的。速度快的芯片在总产量中的比率低，价格也就相应地高。那么是什么因素导致了同一批芯片的性能差异，主要有下面两点：

芯片的速度等级决定于芯片内部的门延时和线延时，这两个因素又决定于晶体管的长度L和容值C，这两个数值的差异最终决定于芯片的生产工艺。

在芯片生产过程中，有一个阶段叫做speed binning。就是采用一定的方法，按照一组标准对生产出来的芯片进行筛选和分类，进而划分不同的速度等级。

2.对于Altera的FPGA芯片

以EP2C35F672C6N为例做一个说明：

EP：工艺；

2C：cyclone II（飓风）（S代表Stratix，A代表arria）；

35：逻辑单元数，35表示大约有35k的逻辑单元；

F：表示PCB封装类型，F是FBGA封装，E(EQFP)、Q(PQFP)、U(UBGA)、M(MBGA)；

Package Type：

E: Plastic Enhanced Quad Flat Pack (EQFP)

Q: Plastic Quad Flat Pack (PQFP)

F: FineLine Ball-Grid Array (FBGA)

U: Ultra FineLine Ball-Grid Array (UBGA)

M: Micro FineLine Ball-Grid Array (MBGA)

672：表示引脚数量；

C：工作温度，C表示可以工作在0°C到85°C，I表示可以工作在-40°到100°C，A表示可以工作在-40°C到125°C；

Operating Temperature：

C: Commercial temperature (TJ = 0°C to 85°C)

I: Industrial temperature (TJ = -40°C to 100°C)

A: Automotive temperature (TJ = -40°C to 125°C)

6：速度等级，6约最大是500Mhz，7约最大是430Mhz，8约最大是400Mhz；

N：后缀，N表示无铅，ES工程样片。

FPGA状态机详解（??）

什么是状态机：状态机通过不同的状态迁移来完成特定的逻辑操作

状态机的分类：Moore型状态机和Mealy型状态机

1. Moore型：状态机的变化只与当前的状态有关
2. Mealy型：状态机的变化不仅与当前的状态有关，还与输入有关

如何创建状态机：状态机的创建可以分为一段式，两段式和三段式

1. 一段式：主要是讲所有的状态变化以及导致的输出变化都写在了一个always快中。
2. 两段式：将一些复位信号，clk信号单独写在一个always快中，其他的状态变化，输出值得变化写在一个always快中。
3. 三段式：将一些复位信号，clk信号单独写在一个always快中，其他的状态迁移变化写在一个always快中，对应状态的输出值得变化写在一个always快中。

**举个例子**：从循环输入的字母中做连续检测，当连续检测到“hello”时，将led灯进行状态的翻转，继续进行下一次的检测。

一段式的编写方式:

module hello(

input clk,//系统时钟信号 50mHz

input rst\_n,//系统复位信号，低电平有效

input [7:0] data,//连续输入的字母

output reg led//led灯

);

//设置需要改变的状态

parameter checkh = 5'b0000\_1,

checke = 5'b0001\_0,

checkla = 5'b0010\_0,

checklb = 5'b0100\_0,

checko = 5'b1000\_0;

reg [4:0]state;

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

begin

led <= 1'b0;

state <= checkh;

end

else

begin

case (state)

checkh:

if(data == "h") state <= checke;

else state <= checkh;

checke:

if(data == "e") state <= checkla;

else state <= checkh;

checkla:

if(data == "l") state <= checklb;

else state <= checkh;

checklb:

if(data == "l") state <= checko;

else state <= checkh;

checko:

if(data == "o")

begin

led <= ~led;

state <= checkh;

end

else state <= checkh;

default:state <= checkh;

endcase

end

endmodule

两段式的编写方式:

module hello(

input clk,

input rst\_n,

input [7:0] data,

output reg led

);

parameter checkh = 5'b0000\_1,

checke = 5'b0001\_0,

checkla = 5'b0010\_0,

checklb = 5'b0100\_0,

checko = 5'b1000\_0;

reg [4:0] cstate;

reg [4:0] nstate;

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

begin

cstate <= checkh;

end

else

cstate <= nstate;

always @(cstate or data)

case (cstate)

checkh:

if(data == "h") nstate <= checke;

else nstate <= checkh;

checke:

if(data == "e") nstate <= checkla;

else nstate <= checkh;

checkla:

if(data == "l") nstate <= checklb;

else nstate <= checkh;

checklb:

if(data == "l") nstate <= checko;

else nstate <= checkh;

checko:

if(data == "o")

begin

led <= ~led;

nstate <= checkh;

end

else nstate <= checkh;

default:nstate <= checkh;

endcase

endmodule

三段式的编写方式:

module hello(

input clk,

input rst\_n,

input [7:0] data,

output reg led

);

parameter checkh = 5'b0000\_1,

checke = 5'b0001\_0,

checkla = 5'b0010\_0,

checklb = 5'b0100\_0,

checko = 5'b1000\_0;

reg [4:0] cstate;

reg [4:0] nstate;

//复位信号，clk的处理（主要是对初始状态进行赋值操作）

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

begin

cstate <= checkh;

end

else

cstate <= nstate;

//状态迁移的处理

always @(cstate or data)

case (cstate)

checkh:

if(data == "h") nstate <= checke;

else nstate <= checkh;

checke:

if(data == "e") nstate <= checkla;

else nstate <= checkh;

checkla:

if(data == "l") nstate <= checklb;

else nstate <= checkh;

checklb:

if(data == "l") nstate <= checko;

else nstate <= checkh;

checko:

if(data == "o")

begin

nstate <= checkh;

end

else nstate <= checkh;

default:nstate <= checkh;

endcase

//输出数据的处理

always @(posedge clk or negedge rst\_n)

if(!rst\_n)

begin

led <= 1'b1;

end

else

case (cstate)

checko:

if(data == "o")

led <= ~led;

default;

endcase

endmodule

**注意**

1. 一般的状态机是使用Always语句和case语句组合来实现的
2. 不可以根据Always快的个数来判断他是属于几段的状态机
3. 一般情况下不建议使用一段式状态机，建议使用二和三段式状态机，二段式状态机使用时序逻辑处理状态变化，组合逻辑处理输入输出的变化，结构比较清晰，但容易产生毛刺
4. 三段式从输入到输出会比一、两段式状态机延时一个时钟周期

FPGA设计的十五条原则详细解析(??)

* 硬件设计基本原则

1. 速度与面积平衡和互换原则：

一个设计如果时序余量较大，所能跑的频率远高于设计要求，能可以通过模块复用来减少整个设计消耗的芯片面积，这就是用速度优势换面积的节约；反之，如果一个设计的时序要求很高，普通方法达不到设计频率，那么可以通过数据流串并转换，并行复制多个操作模块，对整个设计采用“乒乓操作”和“串并转换”的思想进行处理，在芯片输出模块处再对数据进行“并串转换”。从而实现了用面积复制换取速度的提高。

1. 硬件原则

理解HDL本质

1. 系统原则

整体把握

1. 同步设计原则

设计时序稳定的基本原则

* Verilog作为一种HDL语言，对系统行为的建模方式是分层次的。比较重要的层次有系统级（system）、算法级（Algorithm）、寄存器传输级（RTL）、逻辑级（Logic）、门级（Gate）、电路开关级（Switch）。
* 实际工作中，除了描述仿真测试激励（Testbench）时使用for循环语句外，极少在RTL级编码中使用for循环，这是因为for循环会被综合器展开为所有变量情况的执行语句，每个变量独立占用寄存器资源，不能有效的复用硬件逻辑资源，造成巨大的浪费。一般常用case语句代替。
* if…else…和case在嵌套描述时是有很大区别的，if…else…是有优先级的，一般来说，第一个if的优先级最高，最后一个else的优先级最低。而case语句是平行语句，它是没有优先级的，而建立优先级结构需要耗费大量的逻辑资源，所以能用case的地方就不要用if…else…语句。补充：1.也可以用if…; if…; if…;描述不带优先级的“平行”语句。
* FPGA一般触发器资源比较丰富，而CPLD组合逻辑资源更丰富。
* FPGA和CPLD的组成：FPGA基本有可编程I/O单元、基本可编程逻辑单元、嵌入式块RAM、丰富的布线资源、底层嵌入功能单元和内嵌专用硬核等6部分组成。CPLD的结构相对比较简单，主要由可编程I/O单元、基本逻辑单元、布线池和其他辅助功能模块组成。
* Block RAM:3种块RAM结构，M512 RAM（512bit）、M4K RAM（4Kbit）、M-RAM(64Kbit).M512 RAM:适合做一些小的Buffer、FIFO、DPRAM、SPRAM、ROM等；M4K RAM: 适用于一般的需求M-RAM: 适合做大块数据的缓冲区。Xlinx 和 Lattice FPGA的LUT可以灵活配置成小的RAM、ROM、FIFO等存储结构，这种技术被称为分布式RAM。补充：但是在一般的设计中，不提倡用FPGA/CPLD的片内资源配置成大量的存储器，这是处于成本的考虑。所以尽量采用外接存储器。
* 善用芯片内部的PLL或DLL资源完成时钟的分频、倍频率、移相等操作，不仅简化了设计，并且能有效地提高系统的精度和工作稳定性。
* 异步电路和同步时序电路的区别异步电路:电路核心逻辑有用组合电路实现；异步时序电路的最大缺点是容易产生毛刺；不利于器件移植；不利于静态时序分析（STA）、验证设计时序性能。

同步时序电路:电路核心逻辑是用各种触发器实现；电路主要信号、输出信号等都是在某个时钟沿驱动触发器产生的；同步时序电路可以很好的避免毛刺；利于器件移植；利于静态时序分析（STA）、验证设计时序性能。

* 同步设计中，稳定可靠的数据采样必须遵从以下两个基本原则：(1)在有效时钟沿到达前，数据输入至少已经稳定了采样寄存器的Setup时间之久，这条原则简称满足Setup时间原则；(2)在有效时钟沿到达后，数据输入至少还将稳定保持采样寄存器的Hold时钟之久，这条原则简称满足Hold时间原则。
* 同步时序设计注意事项：异步时钟域的数据转换。组合逻辑电路的设计方法。同步时序电路的时钟设计。同步时序电路的延迟。同步时序电路的延迟最常用的设计方法是用分频或者倍频的时钟或者同步计数器完成所需的延迟，对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，根据计数产生延迟；对于比较小的延迟，可以用D触发器打一下，这样不仅可以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。另外，还有用行为级方法描述延迟，如“#5 a<=4’0101;”这种常用于仿真测试激励，但是在电路综合时会被忽略，并不能起到延迟作用。Verilog 定义的reg型，不一定综合成寄存器。在Verilog代码中最常用的两种数据类型是wire和reg型，一般来说，wire型指定的数据和网线通过组合逻辑实现，而reg型指定的数据不一定就是用寄存器实现。
* 常用设计思想与技巧(1)乒乓操作；(2)串并转换；(3)流水线操作；(4)异步时钟域数据同步。是指如何在两个时钟不同步的数据域之间可靠地进行数据交换的问题。数据时钟域不同步主要有两种情况：两个域的时钟频率相同，但是相差不固定，或者相差固定但是不可测，简称为同频异相问题。两个时钟频率根本不同，简称异频问题。两种不推荐的异步时钟域操作方法：一种是通过增加Buffer或者其他门延时来调整采样；另一种是盲目使用时钟正负沿调整数据采样。
* 模块划分基本原则：(1)对每个同步时序设计的子模块的输出使用寄存器（用寄存器分割同步时序模块原则）。(2)将相关逻辑和可以复用的逻辑划分在同一模块内（呼应系统原则）。(3)将不同优化目标的逻辑分开。(4)将送约束的逻辑归到同一模块。(5)将存储逻辑独立划分成模块。(6)合适的模块规模。(7)顶层模块最好不进行逻辑设计。
* 组合逻辑的注意事项(1)避免组合逻辑反馈环路（容易毛刺、振荡、时序违规等）。解决：A、牢记任何反馈回路必须包含寄存器；B、检查综合、实现报告的warning信息，发现反馈回路（combinational loops）后进行相应修改。(2）替换延迟链。解决：用倍频、分频或者同步计数器完成。(3)替换异步脉冲产生单元（毛刺生成器）。解决：用同步时序设计脉冲电路。(4)慎用锁存器。解决：A、使用完备的if…else语句；B、检查设计中是否含有组合逻辑反馈环路；C、对每个输入条件，设计输出操作，对case语句设置default操作。特别是在状态机设计中，最好有一个default的状态转移，而且每个状态最好也有一个default的操作。D、如果使用case语句时，特别是在设计状态机时，尽量附加综合约束属性，综合为完全条件case语句。小技巧：仔细检查综合器的综合报告，目前大多数的综合器对所综合出的latch都会报“warning”，通过综合报告可以较为方便地找出无意中生成的latch。
* 时钟设计的注意事项(1)同步时序电路推荐的时钟设计方法：时钟经全局时钟输入引脚输入，通过FPGA内部专用的PLL或DLL进行分频/倍频、移相等调整与运算，然后经FPGA内部全局时钟布线资源驱动到达芯片内所有寄存器和其他模块的时钟输入端。FPGA设计者的5项基本功：仿真、综合、时序分析、调试、验证。对于FPGA设计者来说，练好这5项基本功，与用好相应的EDA工具是同一过程，对应关系如下：

1. 仿真：Modelsim, Quartus II(Simulator Tool)

2. 综合：Quartus II (Compiler Tool, RTL Viewer, Technology Map Viewer, Chip Planner)

3. 时序：Quartus II (TimeQuest Timing Analyzer, Technology Map Viewer, Chip Planner)

4. 调试：Quartus II (SignalTap II Logic Analyzer, Virtual JTAG, Assignment Editor)

5. 验证：Modelsim, Quartus II(Test Bench Template Writer)掌握HDL语言虽然不是FPGA设计的全部，但是HDL语言对FPGA设计的影响贯穿于整个FPGA设计流程中，与FPGA设计的5项基本功是相辅相成的。对于FPGA设计者来说，用好“HDL语言的可综合子集”可以完成FPGA设计50%的工作——设计编码。练好仿真、综合、时序分析这3项基本功，对于学习“HDL语言的可综合子集”有如下帮助：

1. 通过仿真，可以观察HDL语言在FPGA中的逻辑行为。

2. 通过综合，可以观察HDL语言在FPGA中的物理实现形式。

3. 通过时序分析，可以分析HDL语言在FPGA中的物理实现特性。对于FPGA设计者来说，用好“HDL语言的验证子集”，可以完成FPGA设计另外50%的工作——调试验证。

1. 搭建验证环境，通过仿真的手段可以检验FPGA设计的正确性。

2. 全面的仿真验证可以减少FPGA硬件调试的工作量。

3. 把硬件调试与仿真验证方法结合起来，用调试解决仿真未验证的问题，用仿真保证已经解决的问题不在调试中再现，可以建立一个回归验证流程，有助于FPGA设计项目的维护。FPGA设计者的这5项基本功不是孤立的，必须结合使用，才能完成一个完整的FPGA设计流程。