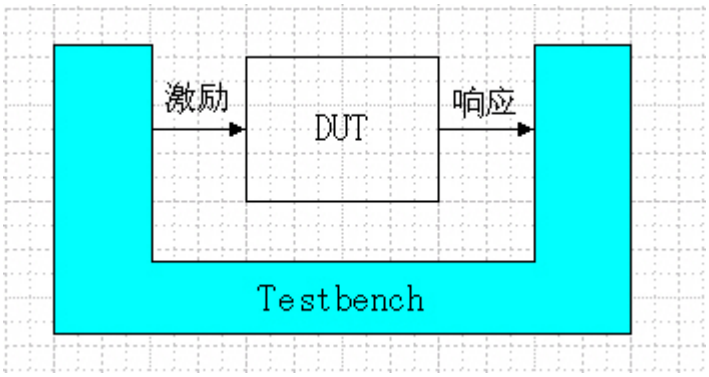
这篇文章有 EDA 论坛 deve 所作：



第一编　验证的重要性

验证，顾名思义就是通过仿真、时序分析、上板调试等手段检验设计正确性的过程，在 FPGA/IC 开发流程中，验证主要包括功能验证和时序验证两个部分。为了了解验证的重要性，我们先来回顾一下 FPGA 开发的整个流程。FPGA 开发流程和 IC 的开发流程相似，主要分为以下几个部分：

1）设计输入，利用 HDL 输入工具、原理图输入工具或状态机输入工具等把所要设计的电路描述出来；

2）功能验证，也就是前仿真，利用 Modelsim、VCS 等仿真工具对设计进行仿真，检验设计的功能是否正确；常用的仿真工具有 Model Tech 公司的 ModelSim，Synopsys 公司的 VCS，Cadence 公司的 NC-Verilog 和 NC-VHDL，Aldec 公司的 Active HDL VHDL/Verilog HDL 等。仿真过程能及时发现设计中的错误，加快了设计进度，提高了设计的可靠性。

3）综合，综合优化是把 HDL 语言翻译成最基本的与或非门的连接关系（网表），并根据要求（约束条件）优化所生成的门级逻辑连接，输出 edf 和 edn 等文件，导给 CPLD/FPGA 厂家的软件进行实现和布局布线。常用的专业综合优化工具有 Synplicity 公司的 synplify /Synplify Pro、Amplify 等综合工具，Synopsys 公司的 FPGA Compiler II 综合工具（Synopsys 公司将停止发展 FPGA Express 软件，而转到 FPGA Compiler II 平台），Exemplar Logic 公司出品的 LeonardoSpectrum 等综合工具。另外 FPGA/CPLD 厂商的集成开发环境也带有一些综合工具，如 Xilinx ISE 中的 XST 等。

4）布局布线，综合的结果只是通用的门级网表，只是一些门与或非的逻辑关系，与芯片实际的配置情况还有差距。此时应该使用 FPGA/CPLD 厂商提供的实现与布局布线工具，根据所选芯片的型号，进行芯片内部功能单元的实际连接与映射。这种实现与布局布线工具一般要选用所选器件的生产商开发的工具，因为只有生产者最了解器件内部的结构，如在 ISE 的集成环境中完成实现与布局布线的工具是 Flow Engine。

5）时序验证，其目的是保证设计满足时序要求，即 setup/hold time 符合要求，以便数据能被正确的采样。时序验证的主要方法包括 STA（Static Timing Analysis）和后仿真。在后仿真中将布局布线的时延反标到设计中去，使仿真既包含门延时，又包含线延时信息。这种后仿真是最准确的仿真，能较好地反映芯片的实际工作情况。仿真工具与综合前仿真工具相同。

6）生成并下载 BIT 或 PROM 文件，进行板级调试。

在以上几个主要开发步骤当中，属于验证的有功能仿真和时序验证两个步骤，由于前仿真和后仿真涉及验证环境的建立，需要耗费大量的时间，而在 STA 中对时序报告进行分析也是一个非常复杂的事情，因此验

证在整个设计流程中占用了大量的时间，在复杂的 FPGA/IC 设计中，验证所占的时间估计在 60%～70%之间。相比较而言，FPGA 设计流程的其他环节由于需要人为干预的东西比较少，例如综合、布局布线等流程，基本所有的工作都由工具完成，设置好工具的参数之后，结果很快就可以出来，因此所花的时间精力要比验证少的多。

一般而言，在验证的几个内容中功能验证最受重视，研究讨论得最多，特别是现在 FPGA/IC 设计都朝向 SOC（System On Chip，片上系统）的方向发展，设计的复杂都大大提高，如何保证这些复杂系统的功能是正确的成了至关重要的问题。功能验证对所有功能进行充分的验证，尽早地暴露问题，保证所有功能完全正确，满足设计的需要。任何潜在的问题都会给后续工作作带来难以极大的困难，而且由于问题发现得越迟，付出的代价也越大，这个代价是几何级数增长的。这里将以功能验证为主说明验证方法、工具、验证环境的建立。

做功能验证时，需要建立验证环境，以便对设计（DUT/DUV，Design Under Test/ Verification）施加特定的输入，然后对 DUT 的输出进行检查，确实其是否正确。在实际验证工作中，一般采用由 TESTBENCH 和 DUT(design under test)组成的 Verification 体系，如图 1 所示。

这是验证系统普遍适用的模型，Testbench 为 DUT 提供输入，然后监视输出，从而判断 DUT 工作是否正确。注意到这是一个封闭的系统，没有输入也没有输出。验证工作的难度在于确定应该输入何种激励，相应的正确的输出应该是怎样的。下一篇我们看个具体的例子。

补充 sta：

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。

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.

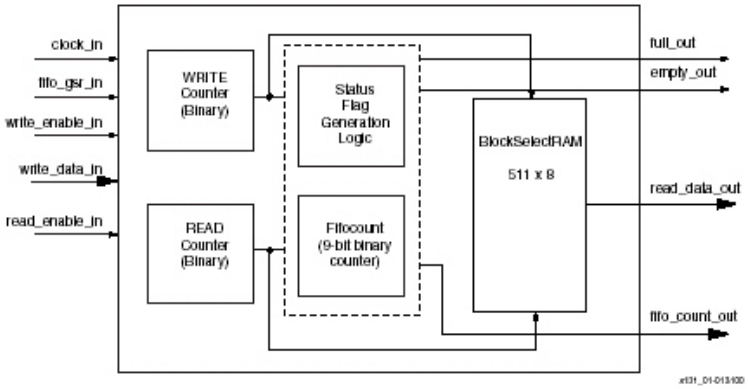


Figure 1: 511 x 8 Synchronous FIFO

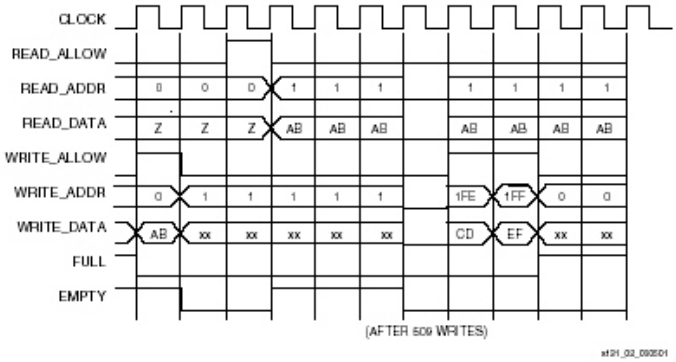


Figure 2: 511 x 8 Synchronous FIFO

读写 task 等内容，初始化部分主要完成复位信号、CLK 信号等的初始化工作，读写 task 把读写、delay 等操作模块化，方便使用。这里主要介绍一下验证 initial 块，也可以说是验证的主程序，如下所示。

```
initial begin
  delay;            //保证验证环境正确复位
  writeburst128;    //写入 512 个数，Full 信号应该在写入 511 个数后变高
  writeburst128;
  writeburst128;
  writeburst128;
  read_enable = 1; //读出一个数，Full 信号应该变低
  writeburst128;    //同时读写，检查 FIFO 操作是否正确
  read_enable = 0; //读操作结束
  endwriteburst;    //写操作结束
  delay;
  readburst128;     //连续读 512 次，Empty 信号应在读出 511 个数后变高
  readburst128;
  readburst128;
  readburst128;
```

```
  endreadburst;
end
```

这段程序首先延迟 5 个时钟周期，等初始化完成之后再开始验证工作。验证时，首先写入 512 个数，使用波形观察器可以检查写入的过程是否正确，以及 Full 信号在写入 511 个数后是否变高；然后 read_enable = 1，读出一个数，Full 信号应该变低，这样写操作和 Full 信号的验证就基本完成了；程序接着也启动了写操作，由于此时 read_enable 仍然为高，即读写同时进行，这是对实际情况的模拟，可以对 FIFO 的功能进行更严格的验证；最后，连续读 FIFO 512 次，用波形观察器检查读操作是否正确，Empty 信号是否在读出 511 个数后变高，如果这些操作都是正确的，那么 FIFO 的功能就基本正确了。

需要注意的一点是，以上的程序是不可综合的，因为不是 RTL 级描述，而是行为级描述（Behavioral Description）。行为级描述的特点是直接描述对象的功能，具有比较高的抽象层次，开发、运行速度都比 RTL 代码要会，因此 testbench 都是用行为级描述写的。关于行为级描述的特点、写法以后将有专门的章节论述。

这个 testbench 的特点是，输入激励由 testbench 产生，输出响应的检查人工完成，这样的 testbench 编写相对容易，可以加快开发速度，作为开发人员自己验证是非常好的选择。有些 testbench 能完成输入激励和输出检查，不用观察波形也能完成验证工作，这样的 testbench 具有更高的自动化程度，使用方便，可重复性好，当设计比较复杂而且团队中有专门的验证工程师时，一般会有验证工程师建立一套这样的 testbench，用于验证开发工程师的 RTL 级代码，如果发现问题，开发工程师修改后在 testbench 再运行一次所花的时间非常少，开发复杂项目时这样做可以比用波形观察器节省很多时间。

3. 总结

验证一般要通过写 testbench 实现，从《FPGA 验证》第一篇我们知道，testbench 要完成向 DUT 施加激励和检查 DUT 相应是否正确的功能，这就要求我们非常清楚待验证模块（DUT）的功能，这样才知道需要验证什么、如何施加激励和如何检查响应是否正确。写 testbench 时， 首先要列出需要验证的功能，让后再编写 testbench，这样可以做到有的放矢，避免遗漏。

思考：

1. Testbench 中有“write_enable = #2 1”一行代码，为什么要 2ns 的延迟？

第三篇 验证工具介绍

我们做 FPGA/IC 开发会用到很多工具，包括代码输入、仿真、综合、布局布线、时序分析等各种各样工具，熟悉这些工具是成功完成设计的关键，因为我们的设计思想需要通过这些工具来实现，只有清楚的知道工具的用法、如何设置参数、如果检查工具的输出结果，才能使设计者的想法变为显示，对验证来说也是如此。

验证的工具很多，有些是验证必不可少的，例如仿真器，有些工具可以代替人完成最繁琐的工作，并能提高功能验证的可信度，例如 linting 和代码覆盖率工具。这里我们介绍常用验证工具的特点和用途，以便为工具的使用提供参考。

1）代码检查工具

常用的代码检查工具有 nlint 等，nlint 根据设计的 RTL 描述代码结构做静态分析，推断描述代码存在的逻辑错误，但无法决定描述代码是否能够现实设计要求的功能。代码检查工具可用于强制代码遵从编写规

范，由于代码检查工具工具是静态验证工具，因此运行速度快，可以节省时间。由于 Verilog 不是强类型语言，使用代码检查工具非常必要，可以检测 race conditions 及数据宽度不匹配，可保证 Verilog 正确描述数据处理过程，避免造成数据的弃位及增位现象，这种错误通过仿真并不一定发现。因为 verilog 语言的特点， 对 Verilog 描述的设计，Linting tool 是一种有益的验证工具。因为 VHDL 语言的特点，对 VHDL 使用 Linting tool 的作用不如对 Verilog 语言那么明显，但 Linting tool 还是能发现一些潜在的问题。

2）仿真器

仿真器是常用的验证工具，它通过忽略及简化设计的物理特性，对设计的实现进行模拟。仿真器通过执行 RTL 级的设计描述，模拟设计的物理实现，它无法确定设计真实的物理实现与设计描述之间的区别。仿真的结果取决于设计描述是否准确反映了设计的物理实现。仿真器不是一个静态工具，需要编写激励和检查输出响应。激励由模拟设计工作环境的 testbench 产生，响应为仿真的输出，由设计者确定输出的有效性。

仿真器的类型分为 3 种类型，Event-driven Simulator（事件驱动仿真器）、Cycle-Based Simulator（基于周期的仿真器）、Co-Simulator（联合仿真器），分别介绍如下：

1. Event-driven Simulator

事件驱动仿真器是最常用的仿真器，例如 modelsim/VCS 等都是事件驱动仿真器，它将信号的变化定义为一个事件，该事件驱动仿真执行，事件驱动仿真器能准确地模拟设计的时序特征，可模拟异步设计。

2. Cycle-based simulator

Cycle-based simulator 仿真器的特点是忽略设计的时序，假定所有 flip_flop 的 setup 和 hold 时间都满足要求，在一个时钟周期，信号仅更新一次，从而信号必须与时钟同步。仿真速度比事件驱动仿真器高。基于周期的仿真器的工作过程步骤是，首先编译电路，将组合逻辑压缩成单独的表达式，根据该表达式可确定 flop 的输入，然后执行仿真，遇到时钟的有效沿， flip_flop 的值被更新。基于周期的仿真器的缺点是不能仿真异步电路，不能进行验证设计的时序。

3. Co-Simulators

联合仿真器对同一设计各个部分，分别用不同的仿真器仿真，如即含有同步设计又含有异步设计的电路，可用 Event-driven Simulator 对异步设计仿真，用 Cycle-based Simulator 对异步设计仿真。联合仿真器中各个 Simulator 的操作是 locked-step 的，类似于电路的 pipeline 操作。其缺点是由于不同仿真器之间需要同步和相互通讯，Co-Simulators 的仿真速度受到最慢 Simulator 的限制，因而影响仿真器的性能，而且在各仿真器传送的信息会产生多义性。

4. Hardware modeler

硬件模拟器创建一个物理芯片的逻辑模型，向仿真器提供该芯片的行为信息，芯片和仿真器的通信过是首先将物理芯片插入硬件仿真器，然后格式化来自仿真器的数据，作为该芯片的输入，最后将该芯片输出的数据，包含时序信息，送往仿真器。硬件模拟器可以提供很高的仿真速度，但是设备价格高昂。需要注意的是，硬件模拟器做的仍然是功能仿真，而不是时序仿真，因为芯片是降频运行的。

3）波形观察器

仿真调试的过程中波形观察器是必不可少的工具，它能提供信号状态和变化的详细信息，但是波形观察器不能用来判断一个设计是否通过验证，因为波形是不可重复的且无法用于递归仿真。

波形观察器的优点是可以观察仿真的整个过程，有利于设计及 testbench 的诊断，缺点是由于要输出波形，影响了仿真的速度，因此应尽可能限制在波形图中显示的信号数量及时间长度。波形观察器的另一个作用

是波形比较，主要用于 redesign，保证设计具有 cycle-accurate 的后向兼容性。在波形比较中，不能仅看表象，需仔细分析，确认波形之间存在的差别是有意义的。例如，有时我们仅关心波形 transitions 之间的相对位置，而不关心它的绝对位置。

以上是比较常用的验证工具，另外可能用到的验证工具有：形式验证工具、静态时序分析工具以及 Vera、SpecmanE、SystemC 等高级语言验证工具，这些工具在复杂的 IC/FPGA 设计中用得比较多。