思想前言

集成电路设计软件目前在世界上只有几家公司在做，普遍分布在欧美等国家，中国的集成电路设计软件长期依赖于盗版和进口。“工欲善其事，必先利其器”，中国要想大力发展集成电路产业，首先要有自主知识产权的集成电路设计工具。

Robei 是一款全新的拥有自主知识产权的集成电路设计工具，不仅具备传统的设计工具的代码编写、编译、仿真功能，更增加了可视化和模块化设计理念，具有模块设计透明化，方便模块重新利用，加快设计进度的特点。

传统的集成电路设计工具庞大、复杂、难用、很不容易入门。初学者将会经历感兴趣→迷茫→头疼→失去兴趣→悟道→加深兴趣→痴迷的过程，其中很大一批人在中间过程中就放弃了继续学习。Robei 可以让初学者平稳而快速的过渡到悟道阶段，并提出 7 天搞定 FPGA 的方案。Robei 为初学者扫平了学习集成电路的荆棘，将泥泞的土路打造成了高铁，为更多人在集成电路学习的道路上保驾护航。

传统的书籍都是由专家撰写，他们的知识体系已成系统，很多内容让初学者摸不清头脑，本书采用了初学者教初学者的方式，案例大部分是由初学者在学习过程中设计出来，然后把自己的学习心得编写成案例。因为了解初学者心理的一定是初学者，而非专家学者。本书将会随着初学者的增多，案例的增多，变得更加精炼，更加贴合读者的发展。

由于编写时间仓促，编者水平有限，书中难免出现纰漏和错误，敬请批评指正。本书仅供读者学习参考使用。

在本书付梓之际，谨向为本书付出辛勤劳动的青岛若贝电子有限公司的工作人员致以诚挚的谢意！参与本书编辑的人员有：

Copyright © Robei

简 介

软件：Robei 可视化芯片设计软件

Robei 是一款可视化的跨平台 EDA 设计工具，具有超级简化的设计流程，最新可视化的分层设计理念，透明开放的模型库以及非常友好的用户界面。Robei 软件将芯片设计高度抽象化，并精简到三个基本元素，掌握这三个基本元素，就能很快地掌握 Robei 的使用技巧。该软件将先进的图形化与代码设计相融合，让框图与代码设计优势互补，弱势相互抵消。Robei 软件是目前世界上最小的芯片设计仿真工具，也是唯一一个能在移动平台上设计仿真的 EDA 工具。它不依赖于任何芯片，在仿真后自动生成 Verilog 代码，可以与其它 EDA 工具无缝衔接。Robei 以易用（Easy to use）和易重用（Easy to reuse）为基础，是一款为芯片设计工程师量身定做的专用工具。

Robei

Copyright © Robei

目 录

第一章：认识工具，掌握基础.....VIII
1.1 为什么要选择 Robei.....IX
1.1.1. 背景介绍.....IX
1.1.2. EDA 的发展史.....X
1.1.3. Robei 的优势.....XIII
1.2 安装与注册.....XVI
1.2.1. 安装.....XVI
1.2.2. 注册.....XIX
1.3 如何使用 Robei.....XXII
1.3.1. 菜单和工具条.....XXII
1.3.2. 工具箱.....XXII
1.3.3. 属性栏.....XXIII
1.3.4. 工作空间.....XXIV
1.3.5. 输出.....XXV
1.4 Robei 三元素.....XXVI
1.4.1. 模块.....XXVI
1.4.2. 引脚.....XXVIII
1.4.3. 连接线.....XXIX
1.5 Verilog 基础.....XXXI
1.5.1. 数据.....XXXI
1.5.2. 运算符.....XXXI
1.5.3. 结构声明.....XXXII
1. 模块定义.....XXXII
2. 引脚定义.....XXXIII
3. 连接线.....XXXIII
4. 例化.....XXXIV

 Copyright © Robei

1.5.4. 代码撰写.....XXXV
1. 赋值语句.....XXXV
2. 分支语句.....XXXV
3. 循环语句.....XXXVI
4. 初始化与重复执行.....XXXVI
5. 阻塞式赋值与非阻塞式赋值.....XXXVII
1.5.5. 一个模块的总结.....XXXVIII
1.6. 第一天的总结.....XL
第二章：实例入手，体验若贝.....XLI
2.1 实例一 逻辑门设计.....XLII
2.1.1. 本章导读.....XLII
2.2.2. 设计流程.....XLII
1. 模型设计.....XLII
2. 测试文件设计.....XLV
2.1.3. 问题与思考.....XLIX
2.1.4. 常见问题.....L
2.2 实例二 计数器.....LI
2.2.1. 本章导读.....LI
2.2.2. 设计流程.....LI
1. 模型设计.....LI
2. 测试文件设计.....LIII
2.2.3. 问题与思考.....LVI
2.3 实例三 编译码器.....LVII
2.3.1. 本章导读.....LVII
2.3.2. 设计流程.....LVII
1. 编码器模型设计.....LVII
2. 译码器模型设计.....LVIII
3. 测试文件设计.....LIX
2.3.3. 问题与思考.....LXII

Copyright © Robei

2.4 实例四 ALU 设计.....LXIII
2.4.1. 本章导读.....LXIII
2.4.2. 设计流程.....LXIII
1. ALU 模型设计.....LXIII
2. 测试文件设计.....LXV
3. 16 位 ALU 设计.....LXVII
4. 32 位 ALU 设计.....LXXI
2.4.3. 问题与思考.....LXXIII
第三章：动手实战，板上点灯.....LXXIV
3.1 实例五 Robei 和 Vivado 的联合设计——流水灯设计.....LXXV
3.1.1. 本章导读.....LXXV
3.1.2. Robei 设计内容.....LXXV
1. light 模型设计.....LXXV
2. light_tb 测试文件的设计.....LXXVII
3. light_constrain 约束文件的设计.....LXXVIII
3.1.3. Vivado 设计内容.....LXXX
1. 工程创建.....LXXX
2. 使用 Vivado 综合工具来综合设计并且分析项目主要输出.....LXXXIV
3. 使用 Vivado 实现设计的分析以及项目摘要输出.....LXXXVI
4. 将设计在开发板上实现.....LXXXVIII
3.1.4. 总结.....XCI
3.2 实例六 自动售饮料机.....XCII
3.2.1. 本章导读.....XCII
3.2.2. 设计流程.....XCII
1. sell 模块的设计.....XCII
2. sell_test 测试文件设计.....XCIV
3. sell_constrain 约束文件设计.....XCVI
3.2.3. 板级验证.....XCVII
1. VIVADO 设计平台进行后端设计.....XCVII

Copyright © Robei

2. 开发板验证.....CII
3.2.4. 问题与思考.....CIII
第四章：复杂运算，板级体验.....CIV
4.1 实例七 8 位移位寄存器的设计.....CV
4.1.1. 本章导读.....CV
4.1.2. 设计流程.....CV
1. shift 模型设计.....CV
2. shift_test 测试文件设计.....CVI
3. shift_constrain 测试文件的设计.....CVIII
4.1.3. 板级验证.....CIX
1. VIVADO 设计平台进行后端设计.....CIX
2. 开发板验证.....CXV
4.1.4. 问题与思考.....CXV
4.2 实例八 带符号位小数的加法设计.....CXVI
4.2.1. 本章导读.....CXVI
4.2.2. 设计流程.....CXVII
1. qadd 模型设计.....CXVII
2. qadd_test 测试文件的设计.....CXVIII
3. 约束模块和约束文件设计.....CXX
4.2.3. 板级验证.....CXX
1. VIVADO 设计平台进行后端设计.....CXXI
2. 开发板验证.....CXXVI
4.2.4. 问题与思考.....CXXVII
4.3 实例九 除法器设计.....CXXVII
4.3.1. 本章导读.....CXXVII
4.3.2. 设计流程.....CXXVIII
1. divider 模型设计.....CXXVIII
2. divider_test 测试文件的设计.....CXXX
3. divider_constrain 约束文件的设计.....CXXXII

Copyright © Robei

4.3.3. 板级验证.....CXXXIII
1. VIVADO 设计平台进行后端设计.....CXXXIV
2. 开发板验证.....CXXXIX
4.3.4. 问题与思考.....CXL
第五章：认识协议，操作接口.....CXLI
5.1 实例十 FIFO.....CXLII
5.1.1. 本章导读.....CXLII
5.1.2. 设计流程.....CXLIII
1. 模型设计.....CXLIII
2. 测试模块设计.....CXLVI
3. 约束模块设计.....CXLVIII
5.1.3. 板级验证.....CL
1. VIVADO 设计平台进行后端设计.....CL
2. 开发板验证.....CLV
5.1.4. 问题与思考.....CLVI
5.2 实例十一 SPI 总线接口的 verilog 的实现.....CLVII
5.2.1. 本章导读.....CLVII
5.2.2. 设计流程.....CLVIII
1. spi_master 模型设计.....CLVIII
2. spi_master_tb 测试文件的设计.....CLXI
5.2.3. SPI 接口协议的板级验证.....CLXIII
5.2.4. 问题与思考.....CLXVI
第六章：串口通信，系统设计.....CLXVII
6.1 实例十二 UART 的发送与接收模块设计.....CLXIX
6.1.1. 本章导读.....CLXIX
6.1.2. 设计流程.....CLXX
1. 接收模块的设计.....CLXX
2. UARTTEST 测试文件的设计.....CLXXII
3. 发送模块设计.....CLXXIV

Copyright © Robei

4. UARTsendtest 测试文件的设计..... CLXXVI
6.1.3. 问题与思考.....CLXXVIII
6.2 实例十三 Natalius 8 位 RISC 处理器.....CLXXIX
6.2.1. 本章导读..... CLXXIX
6.2.2. 设计流程.....CLXXXII
1. ALU 模型设计.....CLXXXII
2. stack 模型设计..... CLXXXIV
3. data_supply 模型设计.....CLXXXVI
4. zc_control 模型设计.....CLXXXVIII
5. data path 模型设计.....CLXXXIX
6. instruction memory 模型设计.....CXC
7. control unit 模型设计.....CXC
8. Natalius processor 模型设计..... CCI
9. processor_test 测试文件的设计..... CCII
6.2.3. 问题与挑战.....CCIV
第七章：总结反思，项目挑战..... CCV
参考文献..... CCVI
鸣 谢.....CCVII

Robei

Copyright © Robei

第一章：认识工具，掌握基础

通过今天的学习，读者可以了解集成电路设计工具的历史背景情况，同时熟悉国内外的产业差距。今天的学习将为后面的操作打下基础，读者需要尽可能的熟悉软件和 Verilog 语法，了解 Robei 软件的结构和操作方式，并知道如何注册和寻找 Robei 资源。今天学习完成后，熟悉 Verilog 语言的读者可以加深记忆，刚刚开始学习 FPGA 设计的读者也可以轻松地掌握 Verilog 语言的结构和语法。

Robei

 Copyright © Robei

1.1 为什么要选择 Robei

1.1.1. 背景介绍

提供 EDA 设计工具的厂家有 Cadence，Synopsys，Mentor Graphics，Xilinx，Altera 等公司，这些公司都是欧美的公司，中国的 EDA 设计工具却少的可怜。中国 90%的芯片来自进口，已经引起了政府的重视和对集成电路产业的大力扶持，但是中国 99%的芯片设计工具来自进口和盗版，目前政府对于集成电路设计工具支持却是凤毛麟角。集成电路产业的发展依托于集成电路设计工具的发展，设计工具是和集成电路产业的发展同步的，就像两条腿走路，缺了哪条腿都是畸形的发展。“工欲善其事，必先利其器”，在中国大力发展集成电路的大环境和氛围下，唯独缺失的是对集成电路工具的发展。当一个国家的某个产业的快速膨胀和繁荣是依赖于其他国家的利器的时候，就等于把这个产业的咽喉拱手让别的国家给掐住。这个产业的利润将会慢慢通过知识产权的诉讼形式流失进入到其他工具所有国。这种畸形的发展会导致欧美等国家顺利完成产业淘汰和升级，中国变成了集成电路产业链的最低端，只能依赖于微薄的利润生存，并准备着时不时的被掌握设计工具的国家过来“剪羊毛”。在 EDA 设计工具上，中国有没有可能和欧美进行抗衡和抵制？

欧美国家的 EDA 设计工具也联合本国的 IP 供应商，打造了一个知识产权产业链，对于一些常用的 IP 进行知识产权封锁，要求国内的的生产和设计厂商进行购买，否则就无法完成整体设计。而且国外掌握着对软件和 IP 更新升级的主动权，即使购买了 IP，在一定时间之后，随着版本的升级，IP 在不同的版本中不兼容，而且 IP 升级需要继续付费，同时新 IP 的使用方法不同，这就要求国内的产业链随着软件和 IP 的升级不断的更改设计，甚至重新设计，导致大量的人力、物力和财力的浪费，同时又不得不出钱购买，因为产业的依附性已经形成。我们如何能打破这个局面？

国内的高校在培养集成电路设计人才的时候，也是不求甚解，很多课时在讲 IP 的使用，而不是如何设计 IP。如此以来，我们通过填鸭式教育培养了一批不会思考的集成电路设计工程师，这批不会思考的工程师在公司只会拿来主义， IP 的封闭导致了中国集成电路产业的惰性，建立在这种惰性之上的创新都是空谈。但是随着集成电路大基金的投入，集成电路制造厂的规模扩建，新生产线的上线和产能的增加，如何快速培养更多的集成电路人才来设计和流片，用来喂饱这些新增的产能和高端工艺线？

仔细分析一下集成电路设计的历史，从最早的逻辑门搭建到原理图设计（软件化的集成电路芯片模组），再到编写 Verilog 或者 VHDL 的代码（充分灵活的设计方式），每一步都是为了让设计更方便，更简单，更抽象。未来的 EDA 工具会怎么样？

现在的 EDA 工具设计相当灵活，全部由代码实现功能并仿真。在设计中牵扯到大量的模块重用，进行例化，这个过程需要使用者清楚要进行例化的每个引脚和位宽，用户就需要在声明和设计中来回切换，不时查证引脚信号的定义，浪费了大量的时间。如何才能更方便更快速的进行例化？

在大型的设计中，需要反复的对一些细节或者模块进行修改，一旦用的多得模块被修改，用户中其他的设计就要更新，将所有用到被修改了的模块的地方进行更新牵扯到大量的时间，甚至是重新设计。如果能有一键更新，分层调整，将可以大大节约工作量。如何在软件中进行快速的自上而下和自下而上的协同设计并能实现一键更新？

集成电路的学习是一个痛苦而抽象的过程，它不像 C 语言软件一样，可以快速编译，所见即所得。EDA 设计中语句是并行执行的，C 语言中是串行执行的，如何能减小软件和硬件设计的鸿沟，让有 C 语言基础的人快速转变成硬件设计工程师？大多数 EDA 软件都是庞然巨物，要想入门 EDA，首先必须把庞然巨物下载完成然后安装并熟悉使用，一旦运行

巨物，就要吞噬电脑的大量内存和计算性能，有没有办法用最小的工具完成初学者的学习任务，而让电脑运行流畅不妨碍我打游戏呢？

因为缺乏竞争对手，国外的 EDA 公司在国内的售价高的离谱，几十万上百万一套软件已经是司空见惯。上规模的集成电路公司为了避免被起诉不得不花费天价来购买软件，而这些费用都是几年或者十几年从微薄的利润中积累出来的。小微集成电路企业无力支付天价的软件费用，只能采用盗版，这也为以后成长为大公司埋下了法律隐患。如何让中国的 EDA 设计工程师用得起正版 EDA 软件？如何让他们不再因为使用盗版而天天提心吊胆？

若贝公司推出的 Robei 可视化芯片设计工具是在各种现有的 EDA 工具的最上层加了一层，进行可视化的所见即所得的设计，同时向下跨越到设计仿真和波形查看，基本涵盖了设计前端的所有功能，实现 RTL 级别的设计仿真，减少了中国大部分工程师和学生对于国外 EDA 设计工具的依赖和减少国外 EDA 设计工具的进口使用量。以前，国外的 EDA 设计工具需要人手一个，这样的成本不是个人和中小公司能够负担的起的，现在，由于 Robei 的出现，我们只需要购买少量的后端设计工具，大大减小了开支。同时 Robei 支持的结构化和可视化设计，方便了模块重用，节省了设计公司和工程师的时间，提升了设计效率和提升了设计质量。目前的集成电路工具更多是面向设计工艺的，极少从设计者和学习者的使用方便触发，而且软件动辄上 G 或者几十 G，对电脑性能要求也极高。Robei 软件是一个以轻量决胜的软件，只有不到 10M，可以减少学生学习的时间，提升学习的兴趣和效率，可以为集成电路行业培养更多的人才。

Robei 打造了一个完全透明的 IP 平台，在这个平台上，我们将容纳更多会思考、会设计的工程师来打造更多更好的设计，所有的设计完全透明，用户在使用的时候可以随意更改和裁剪，也可以学习 IP 设计的相关思路。这个平台打造的是学习与贡献一体，鼓励贡献优质 IP，定期公布设计项目，让更多有思想的工程师参与项目，打造 IP 共享社区，奖励优秀设计者。本来我们集成电路设计就落后很多，如果再采用知识自我封闭的理念，将会导致更加落后。Robei 崇尚的是一种学完就分享的模式，让更多的设计者受益，让更大的设计变得方便可控。这个平台是对知识产权垄断的抗争，是兴起集成电路设计产业的的中坚力量。

1.1.2. EDA 的发展史

最早的集成电路设计是依赖于逻辑门的，用 74LS 系列的芯片进行连接，实现一些基本的功能。随着技术的发展和演进，出现了 PCB 的设计，也就是板级的设计，可以让这些基本的逻辑单元器件布局到一个 PCB 板上实现。PCB 的设计演进也单独的发展成了一个分支，一直延续使用到现在。在集成电路历史上，第一个可编程逻辑器件诞生于 1985 年，由硅谷的 Xilinx 公司推出。既然是可编程逻辑器件，自然少不了可以用于编程的 EDA 工具。

最早的 EDA 设计工具是原理图设计方式，原理图主要基于市场上常见的元器件进行组合连线设计。这种设计基本上是将逻辑门设计中的基础电路模块软件虚拟化之后提供在设计库中，用户可以用来软搭建出自己需要的系统，再写入到 FPGA 中。原理图的优点是设计非常直观，但是灵活性不够。如果在原理图中新增加一个不存在的设备器件，往往需要写代码、打包、更新到器件库、从器件库查找并使用等过程，一旦元器件设计有误，就会需要重新回到代码修改并重新打包。已经在设计中存在的器件需要重新更替。流程繁复浪费了设计师大量的时间，尤其是源代码不慎丢失，将会直接导致模块无法修改，设计师需要重新来过。

后来由于原理图设计中每个模块要对应于实际的芯片，设计灵活性差，导致了后来以 VHDL 和 Verilog 为主流的代码编程设计，所有的模块实现均采用代码编写实现。代码设计是目前非常流行的设计方式，无论是元器件接口定义，模块例化还是功能实现，全部用

 Copyright © Robei

Verilog 或者 VHDL 等语言设计实现。代码设计的优点是灵活，想写什么器件就写什么器件，缺陷是不够直观。工程师需要读完大段代码才能了解其功能和结构信息，同时在撰写例化的时候，需要依据代码进行编写例化，要反复对照和查看接口名称和数据宽度等信息，防止出错。例化的代码也是手写为主。

界面设计方式和代码设计方式各成一体，相互之间交叉很少。模块化设计体现不够完美，很多模块不公开。基于界面的设计在生成代码的过程中，掺杂了很多冗余信号，在出错后返回查看代码，增加了很多冗余信号的障碍。

21 世纪之后，伴随着 FPGA 的越来越大，工艺越来越先进，逻辑资源使用量已经不是设计的瓶颈，这个时候出现了方便设计和使用的框图设计模式。最灵活的框图设计模式当属 Robei，它采用了框图设计结构，代码设计算法的方式，让软件自动生成结构层代码并与用户输入的代码组合成完整的代码。这种设计方式既拥有原理图设计的直观，又拥有代码设计的灵活性。

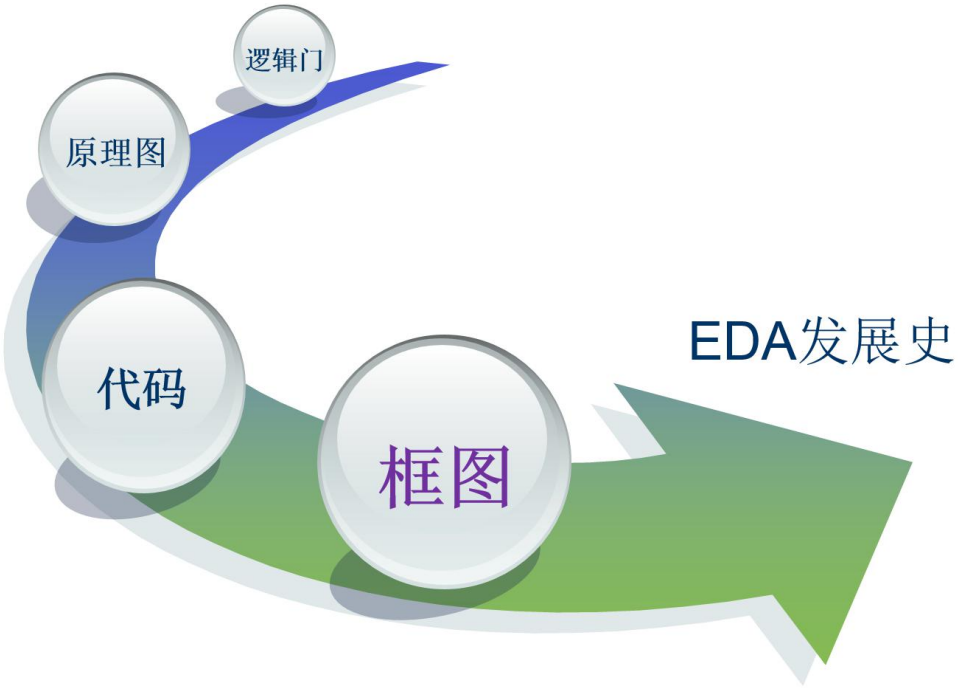


图 1-1-1 集成电路的发展史

自 Xilinx 公司 1984 年诞生到现在，其软件经历了 14 代以上的更新迭代。我们可以分析一下该公司的 ISE 设计工具的大小变化，来分析 EDA 软件的膨胀过程。Xilinx 的 ISE 从最早的几十兆到现在的 8 个 G，发展速度十分快，但是软件庞大毕竟不是好事，软件越大，问题越多，为了修复和维护这么庞大的软件，所需要的人力成本就越高，所以 Xilinx 公司有几千人做软件，远远超过做硬件设计的人数。在集成电路行业，软件的庞大是司空见惯，这是因为所有厂商都以芯片的结构为核心来设计软件，却忽略了用户的感受。从用户角度来说，软件的庞大就代表着下载耗费大量的时间，学习需要比较长的周期，运用的步骤复杂，每一步都要按部就班，稍有差错就要焦头烂额。Robei 是一款面向用户的芯片设计工具，从诞生之日起，Robei 开放软件让超过 35 万人试用，得到用户的体验和感受的问题，并在之后的版本中进行不断修改演进。这些意见来自美国、加拿大、中国、欧洲、韩国、日本等国家，能够让 Robei 快速匹配来自不同地域的用户体验感受。时至今日，Robei 也在鼓励用户反馈问题，在意见采纳后，为每个用户提供奖励。

Copyright © Robei

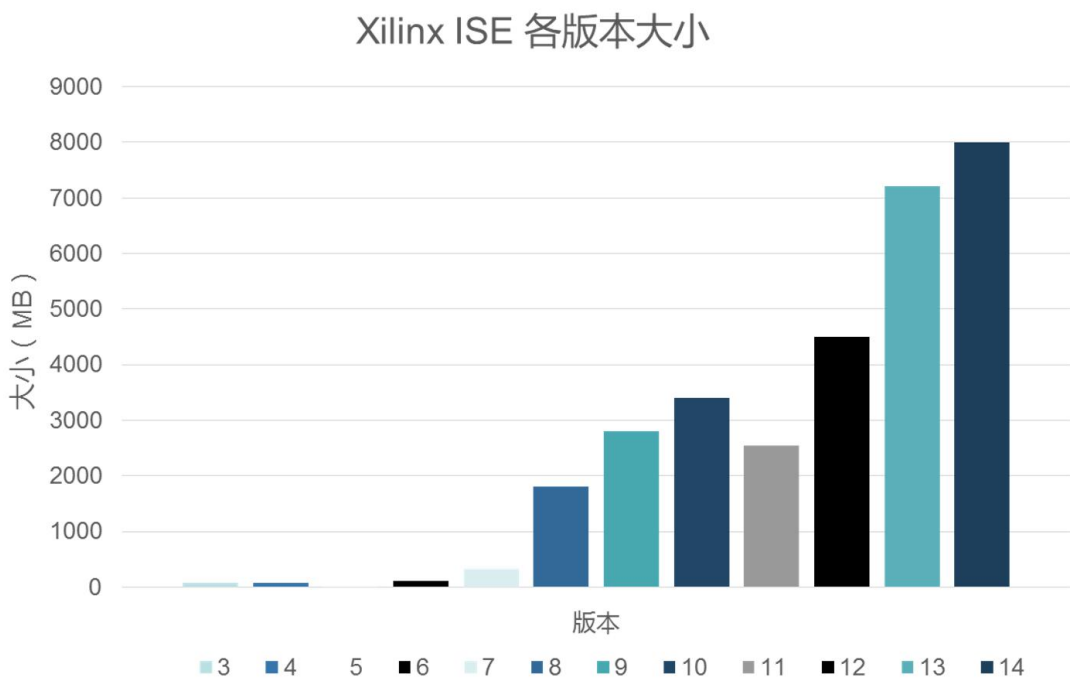


图 1-1-2 EDA 设计工具的演进（Xilinx ISE 为例）

Countries	Amount	Percentage
China	198499	55.97 %
United States	49324	13.91 %
Unknown	11789	3.32 %
India	9891	2.79 %
Germany	7649	2.16 %
Ukraine	6465	1.82 %
Russian Federation	5387	1.52 %
Japan	5302	1.49 %
European Union	5141	1.45 %
France	5039	1.42 %
Brazil	4760	1.34 %
United Kingdom	4318	1.22 %
Taiwan Province Of China	3595	1.01 %
Republic Of Korea	3277	0.92 %
Canada	2168	0.61 %
Romania	1987	0.56 %
Spain	1861	0.52 %
Israel	1663	0.47 %
Netherlands	1593	0.45 %
Italy	1517	0.43 %
Czech Republic	1431	0.4 %
Egypt	1377	0.39 %
Turkey	1304	0.37 %
Poland	1142	0.32 %
Mexico	1131	0.32 %
Sweden	1123	0.32 %
Malaysia	1113	0.31 %
Australia	1069	0.3 %

图 1-1-3 Robei 用户分布图

Copyright © Robei

1.1.3. Robei 的优势

Robei 是一款可视化的跨平台 EDA 设计工具，提供了超级简化的设计流程，最新可视化的分层设计理念，透明的模型库以及非常友好的用户界面。Robei 软件将芯片设计高度抽象化，并精简到三个基本元素，掌握这三个基本元素，就能很快地掌握 Robei 的使用技巧。该软件将先进的图形化与代码设计相融合，让框图与代码设计优势互补，弱势相互抵消。

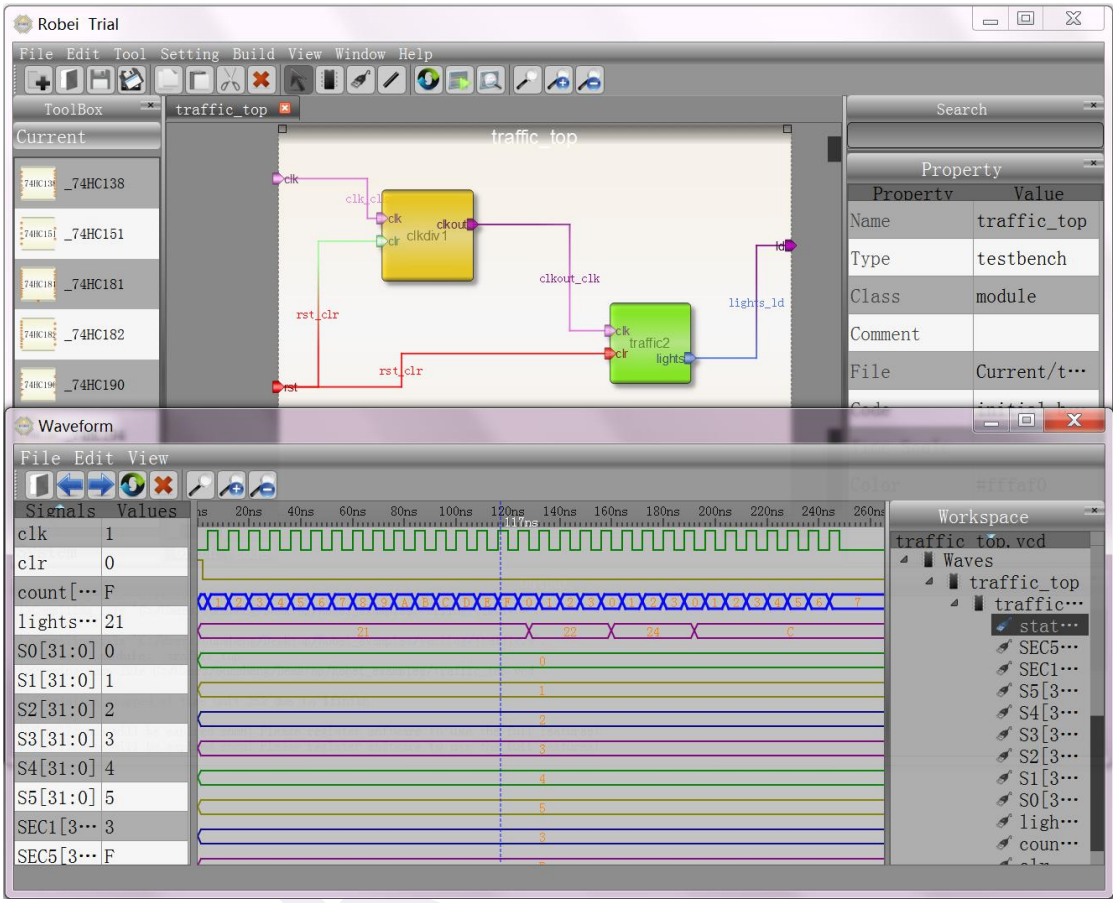


图 1-1-4 Robei 界面图

Robei 软件是在 Verilog 代码设计的基础上进一步抽象，让框图设计与代码设计实现完美融合。传统的原理图设计虽然看起来非常直观，但是灵活性差，用户要利用现成的模块来拼凑设计。而代码设计虽然相当灵活，但是密密麻麻的代码很不直观。Robei 软件通过一种结构层面上图形化设计，算法层面上代码输入的方式使设计更加直观灵活。

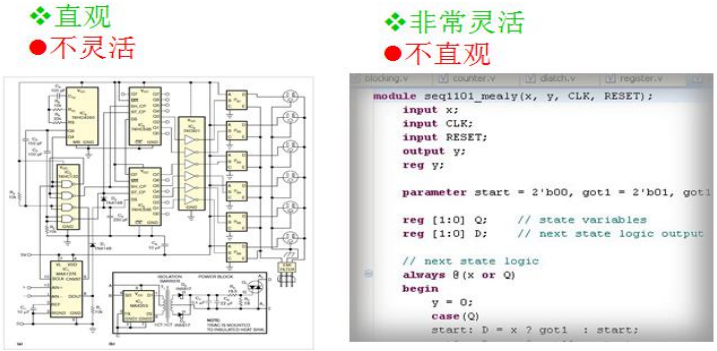


图 1-1-5 Robei 让框图设计与代码设计优势互补

图 1-1-6 Robei 自动生成代码

目前 EDA 的设计首先需要工程师在脑海中设计结构，再根据结构手写代码，容易出错。现在工程师利用 Robei 软件可以边构思边设计结构，结构完成后工程师可以专注于写核心算法，软件自动生成结构层的代码并与工程师输入的算法代码结合仿真。这种设计可以让工程师专注在设计算法上，而不用去记任何引脚名称和数据宽度。同时该软件将模型设计、测试文件和引脚分配集成在一个超级简化的设计流程中，可以进行快速设计仿真。

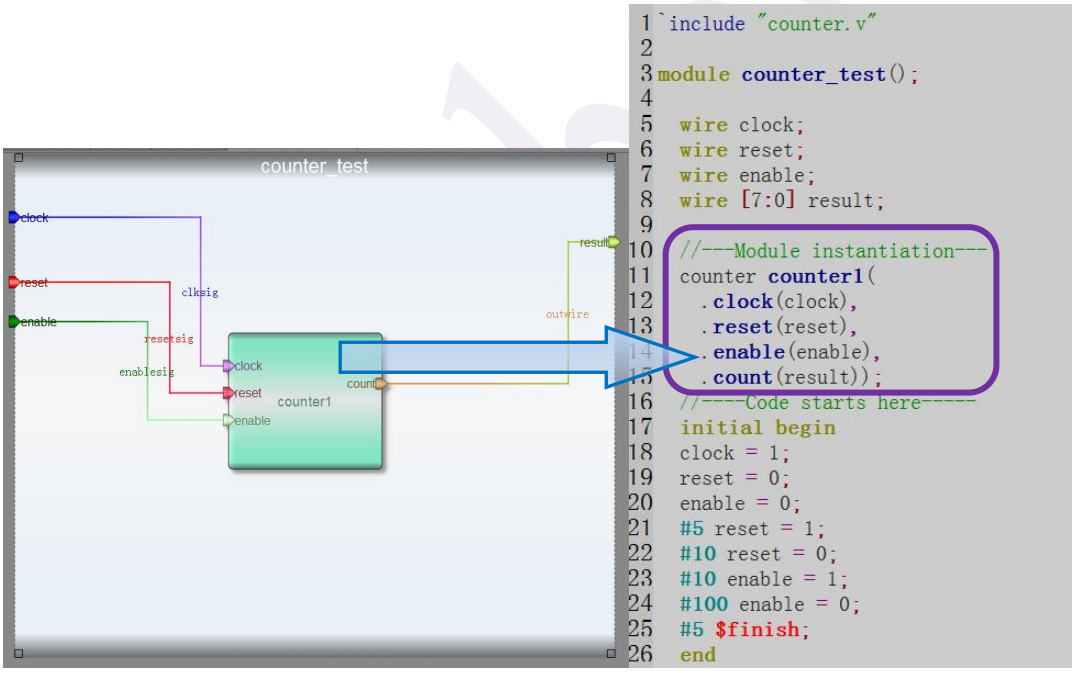


图 1-1-7 Robei 让框图设计与代码设计优势互补

Robei 集成了先进的图形化与代码设计的优势，同时具备 Verilog 编译仿真和波形分析，可以实现各种系统的快速设计、仿真和测试。软件生成标准的 Verilog 代码，可直接与各种 EDA 工具相融合。Robei 是最贴近用户的前端设计仿真软件，仿真后直接生成 Verilog 代码，可以直接在其他 EDA 设计工具中使用。

Copyright © Robei

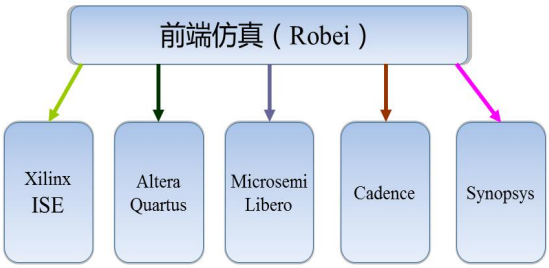


图 1-1-8　Robei 是一个通用的前端设计工具

Robei 也是一款跨平台的 EDA 仿真工具，不仅有 Windows，Linux，Mac OS 的版本，也有专门针对于 Android 平板的版本。到目前为止，Robei 也是唯一一个可以在移动平台上设计仿真的 EDA 工具。

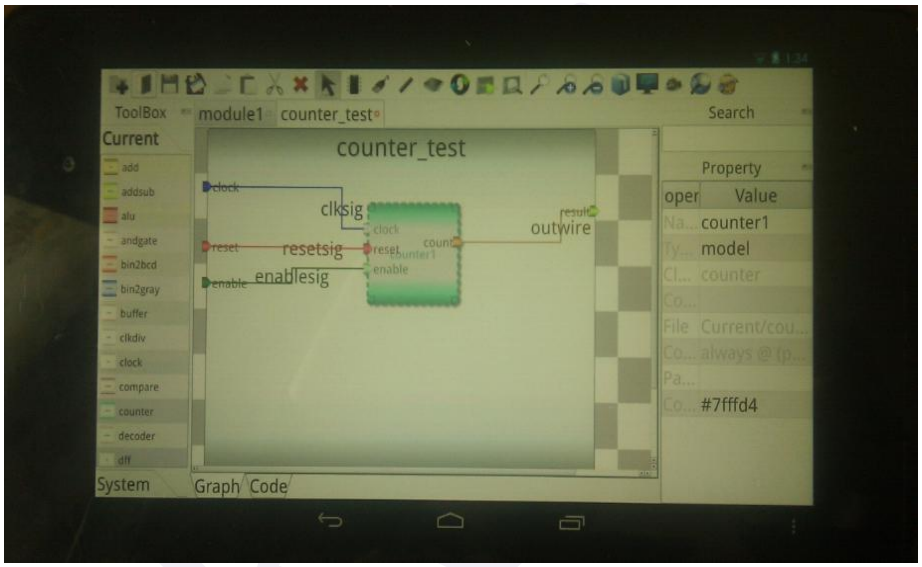


图 1-1-9 Robei 在安卓平板上运行

Copyright © Robei

1.2 安装与注册

1.2.1. 安装

从 Robei 官方网站(http://robei.com)上下载最新版 Robei 软件。解压 Robei.zip，然后双击 Robei-setup.exe，在弹出的安全警告中选择“是”，如图 1-2-1 所示。

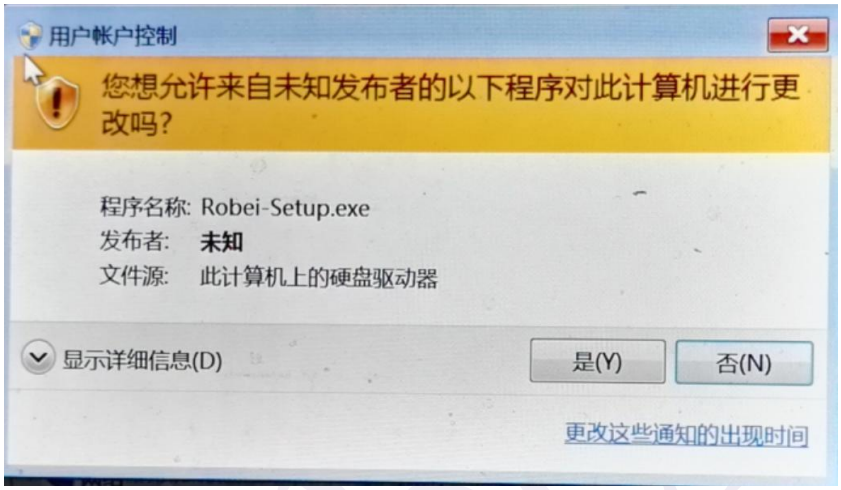


图 1-2-1 电脑提示

Robei 安装界面会出现，如图 1-2-2 所示。

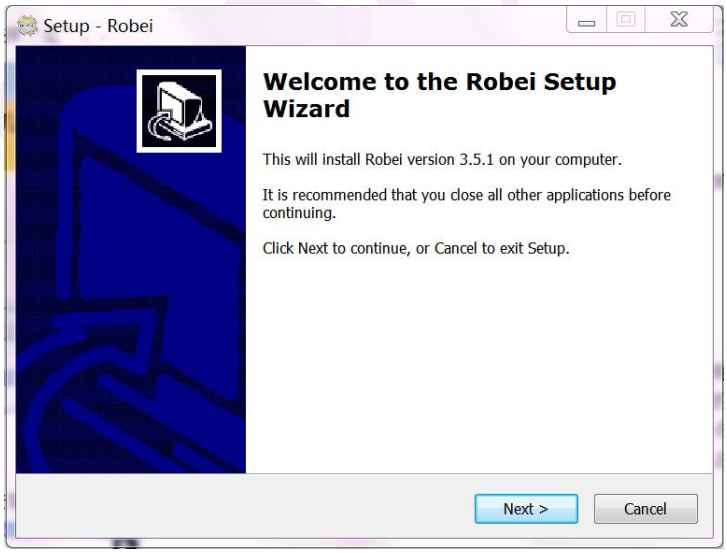


图 1-2-2 安装启动

在弹出的窗口中点“Next”。

 Copyright © Robei

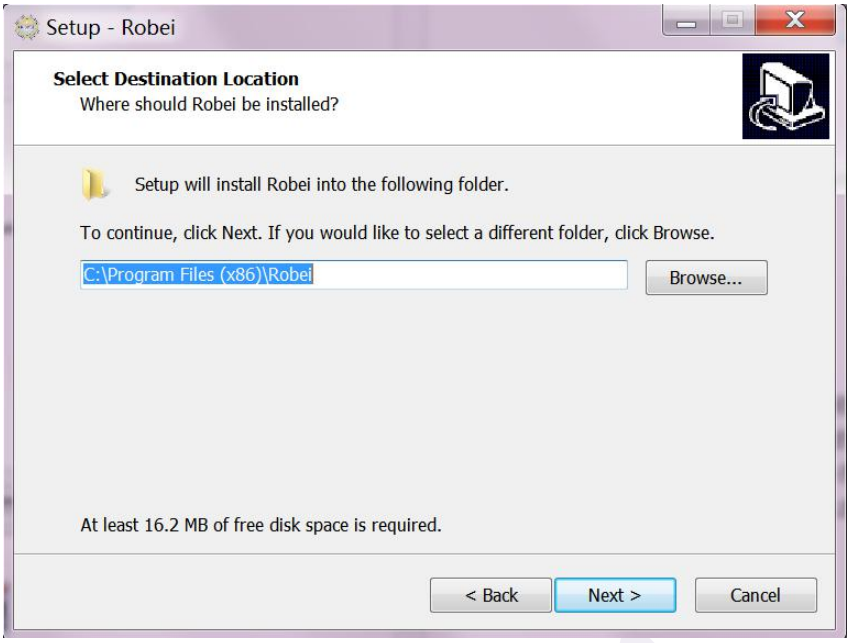


图 1-2-3 安装位置选择

如需要更换路径，可以点“Browse...”按钮重新选择路径，如果不需要更改，继续在弹出的窗口中点“Next”。

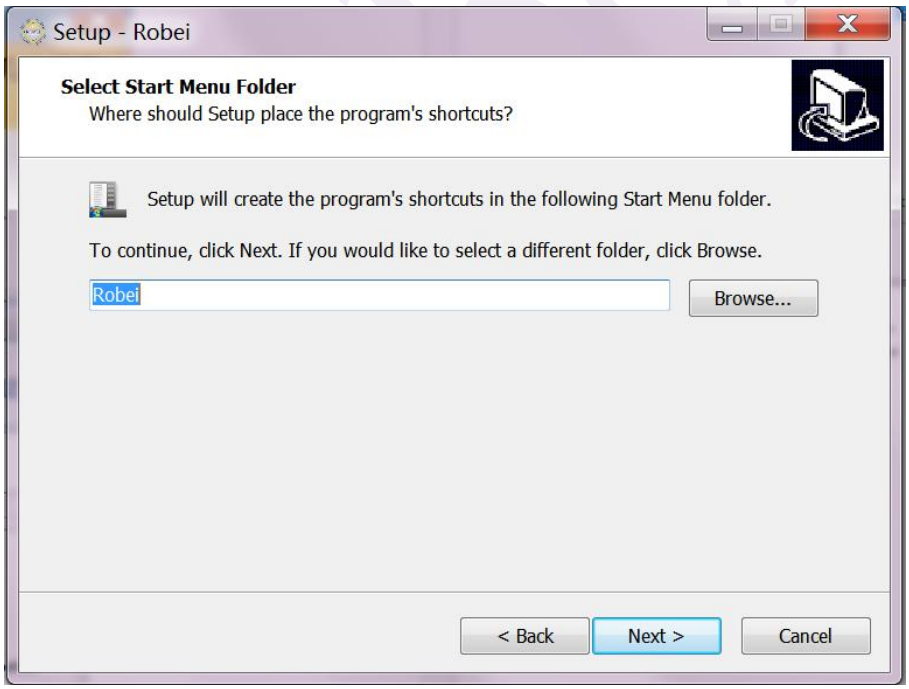


图 1-2-4 开始菜单添加 Robei

点击 Next。

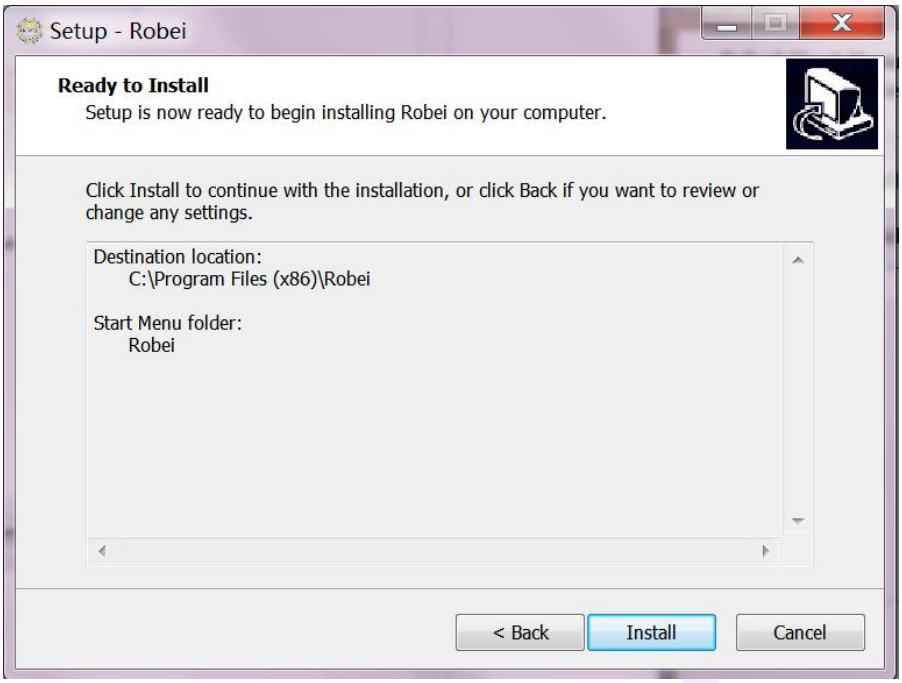


图 1-2-5 安装准备就绪

点击“Install”。

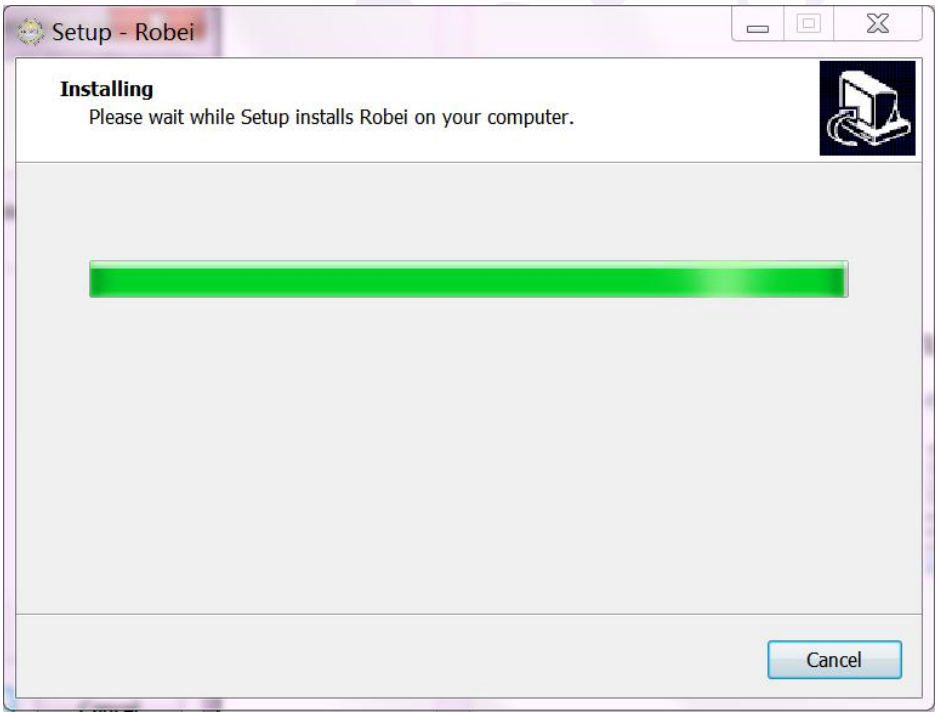


图 1-2-6 安装过程

等待执行完毕。

Copyright © Robei

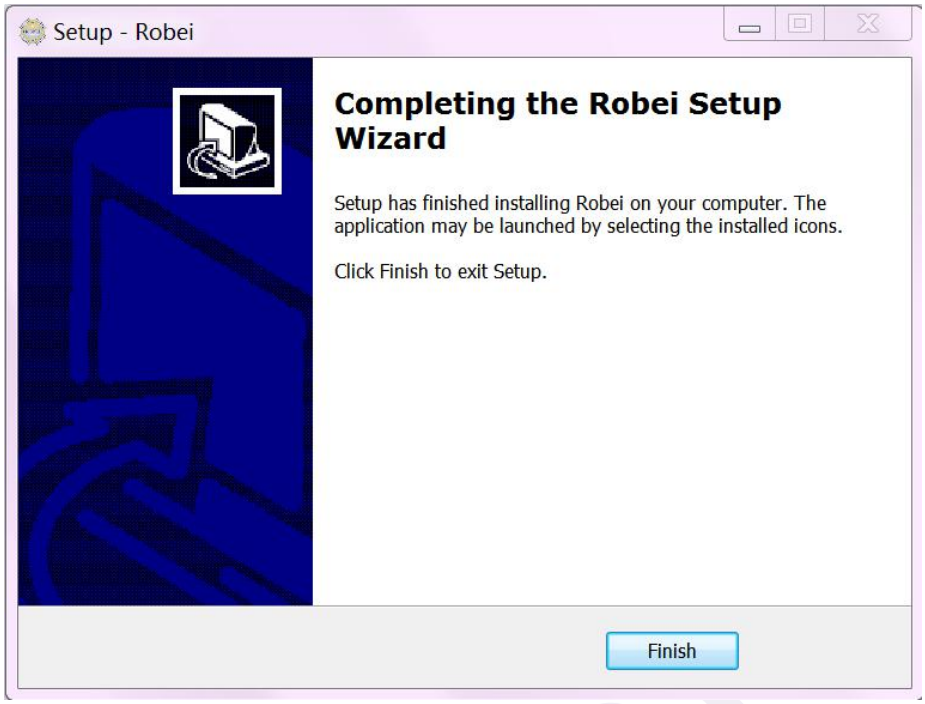


图 1-2-7 安装完毕

点击“Finish”，安装完毕。你可以从桌面上或者开始菜单栏中找到 Robei，启动 Robei 会看到如下图 1-2-8 的界面。

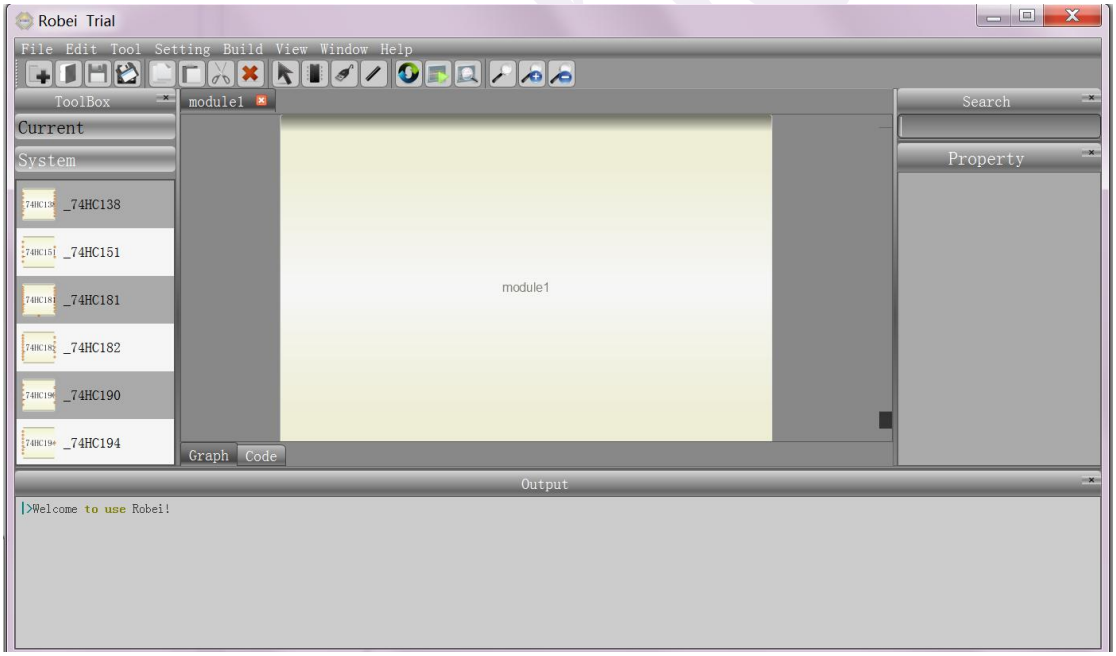


图 1-2-8 启动界面图

1.2.2. 注册

打开 Robei 官方网站：http://robei.cn/register.php，注册新用户，用户名称中不能含有中文和空格。注册完成后可以登录后台购买 Robei 的注册码。Robei 注册码分为四种不同的类型，学生版、个人版、教育版和专业版，请根据自身的需求选择购买。

获得注册码后，返回电脑上打开 Robei 软件，点击菜单“Help”，在下拉菜单里选择“Register”，如下图 1-2-9 所示。

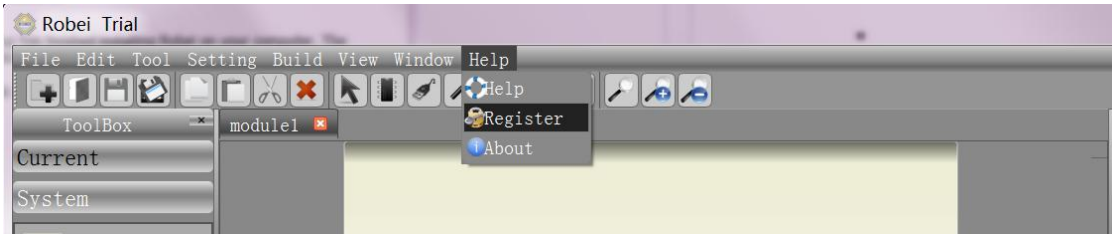


图 1-2-9 启动界面图

在弹出的 Register 对话框中输入之前注册的用户名和密码，点击按钮“Get License”，软件会弹出网页，在该网页中有相关的注册码信息。

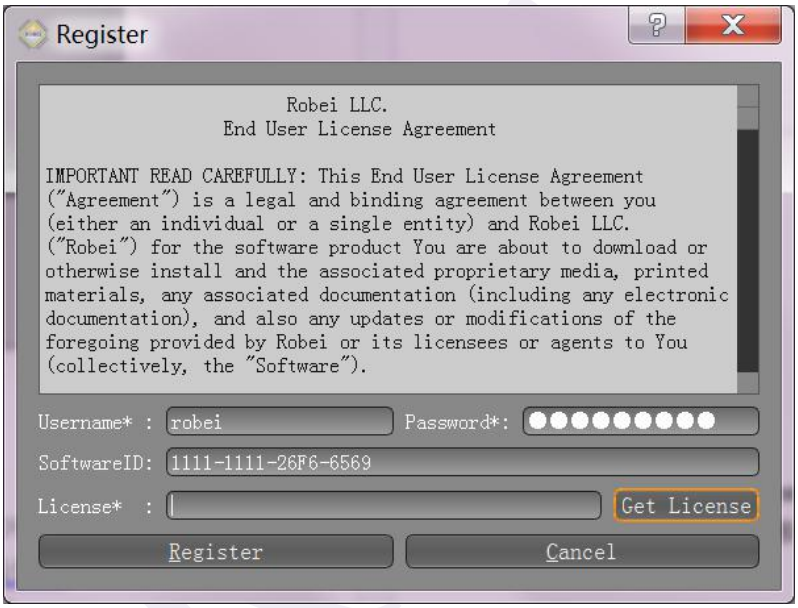
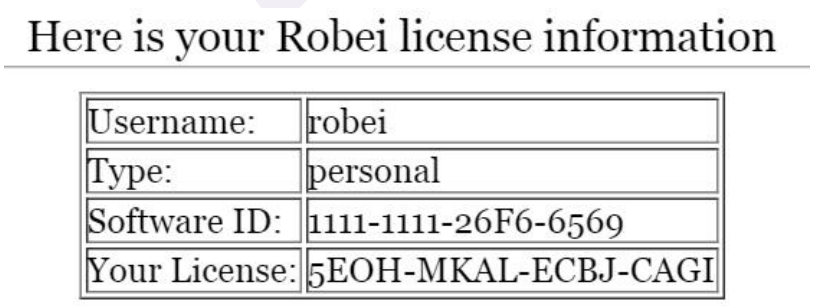


图 1-2-10 Register 对话框



Here is your Robei license information

Username:	robei
Type:	personal
Software ID:	1111-1111-26F6-6569
Your License:	5EOH-MKAL-ECBJ-CAGI

图 1-2-11 注册信息

复制 Your License：后面的一串编码，并输入到 Register 对话框的“License*：”中，点击“Register”按钮。

Copyright © Robei

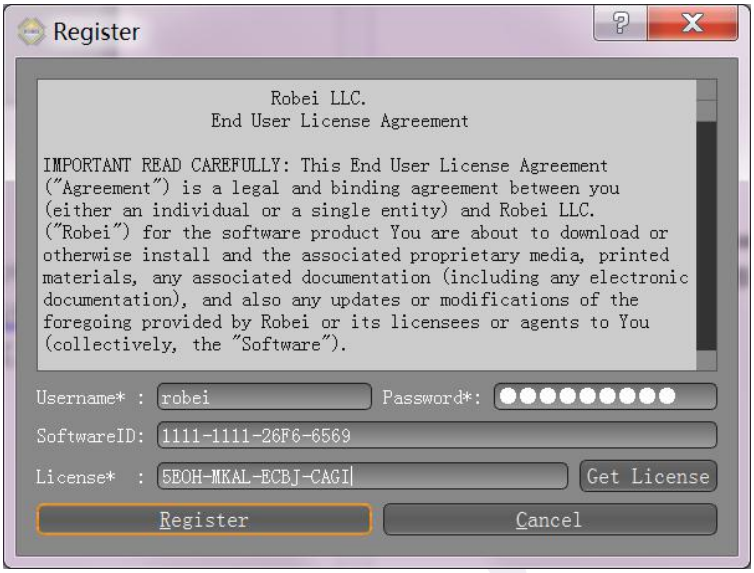


图 1-2-12 注册信息

如果看到下图 1-2-13 所示的对话框，恭喜你，注册成功。如果没有，请联系若贝公司：robei@robei.com。点击“Yes”按钮，退出注册。

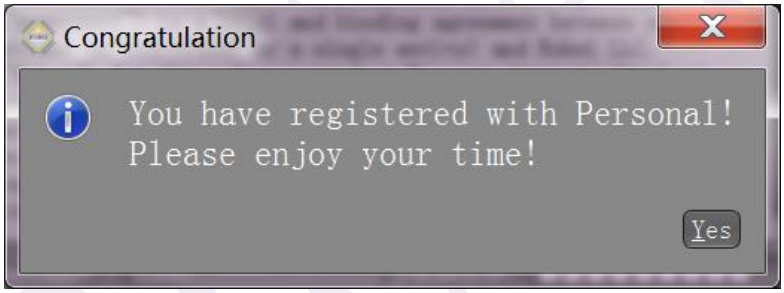


图 1-2-13 注册信息

一切准备就绪之后，关闭软件，重新启动，查看软件界面的最顶端，“Robei Trial”的字样已经消失，取而代之的是你的注册码的类型：学生版是“Robei Student”，个人版是“Robei Personal”，教育版是“Robei Education”，专业版是“Robei Professional”。无论何种版本都可以跟随本教程使用，只是能编译的设计个数受限。

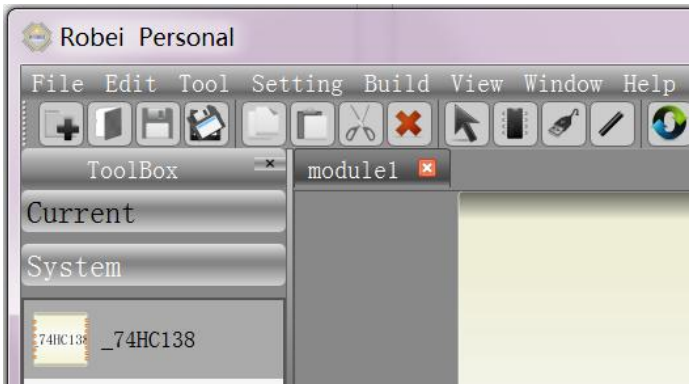


图 1-2-14 注册成功后注册码类型信息

1.3 如何使用 Robei

Robei 软件启动后，界面如图 1-3-1 所示，软件界面分为菜单，工具条，工具箱，属性栏，工作空间和输出窗口几个部分。菜单和工具条位于软件的顶部，与常见的软件工具一样，工具条上分布着一些常用的按钮。左侧的工具箱里包含设计好的模型，可以重复利用。用户可以在界面右侧的属性栏里修改当前设计模块的属性。中间的工作空间是主要设计区域，当前设计模块默认名为“module”。底部为输出窗口，用来显示错误以及警告信息。

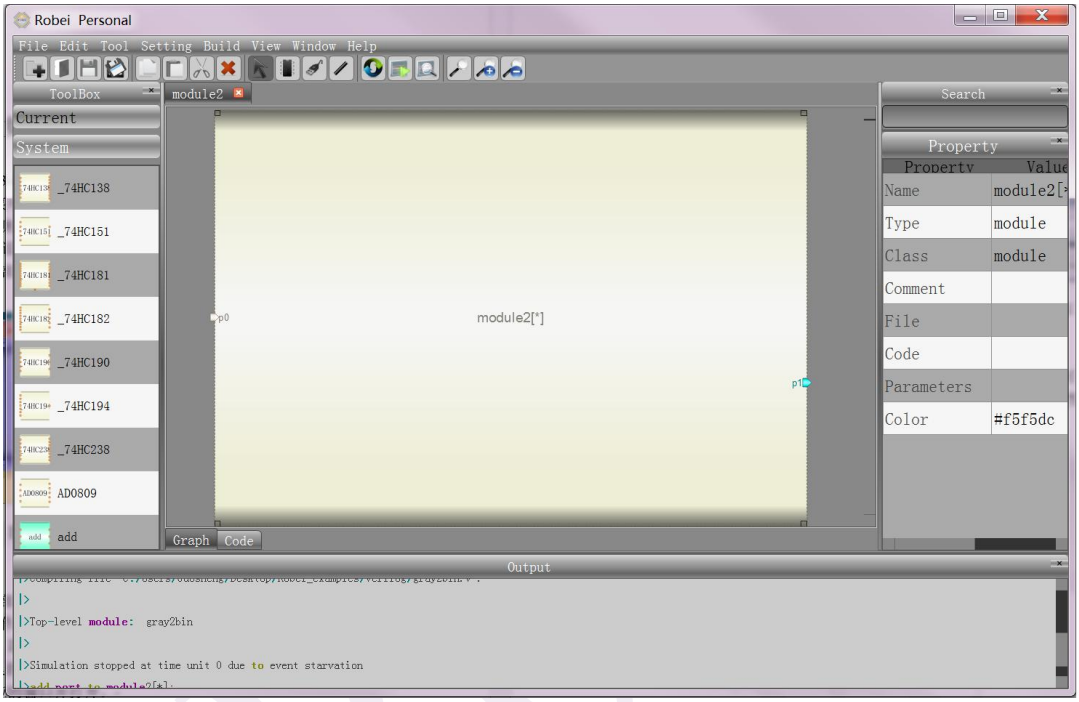


图 1-3-1 界面图

1.3.1. 菜单和工具条

Robei 为了方便用户使用，在顶部设有菜单和工具条。与文件相关的操作放在“File”菜单里，复制、粘贴、剪切和删除操作放置在“Edit”菜单里。与设计相关的操作比如添加模块、引脚、和连接线放在“Tools”菜单里。除了这些，还有在“Build”菜单里的执行仿真和查看波形操作，“View”菜单里的放大缩小操作。如果想恢复被错误关闭的窗口，用户可以到“Window”菜单下找出对应的窗口并打开。

图 1-3-2 菜单栏和工具条

1.3.2. 工具箱

工具箱分为两栏，第一栏“Current”对应的文件位置为当前工作文件夹，用来展示在当前目录下，用户开发的设计模型，其路径是用户当前模型所存储的文件夹。第二栏“System”是随 Robei 软件一起安装好的，里面的模型均由系统自带，其路径是“C:\ProgramData\Robei”。

 Copyright © Robei

用户也可自行添加新的栏目，并给出所对应的文件夹，Robei 会自动读取该文件夹里所有的 Robei 模型。添加方法是在“Toolbox”里点击右键，选择“Add”，系统会自动弹出添加库的对话框（如图 1-2-3 所示）。

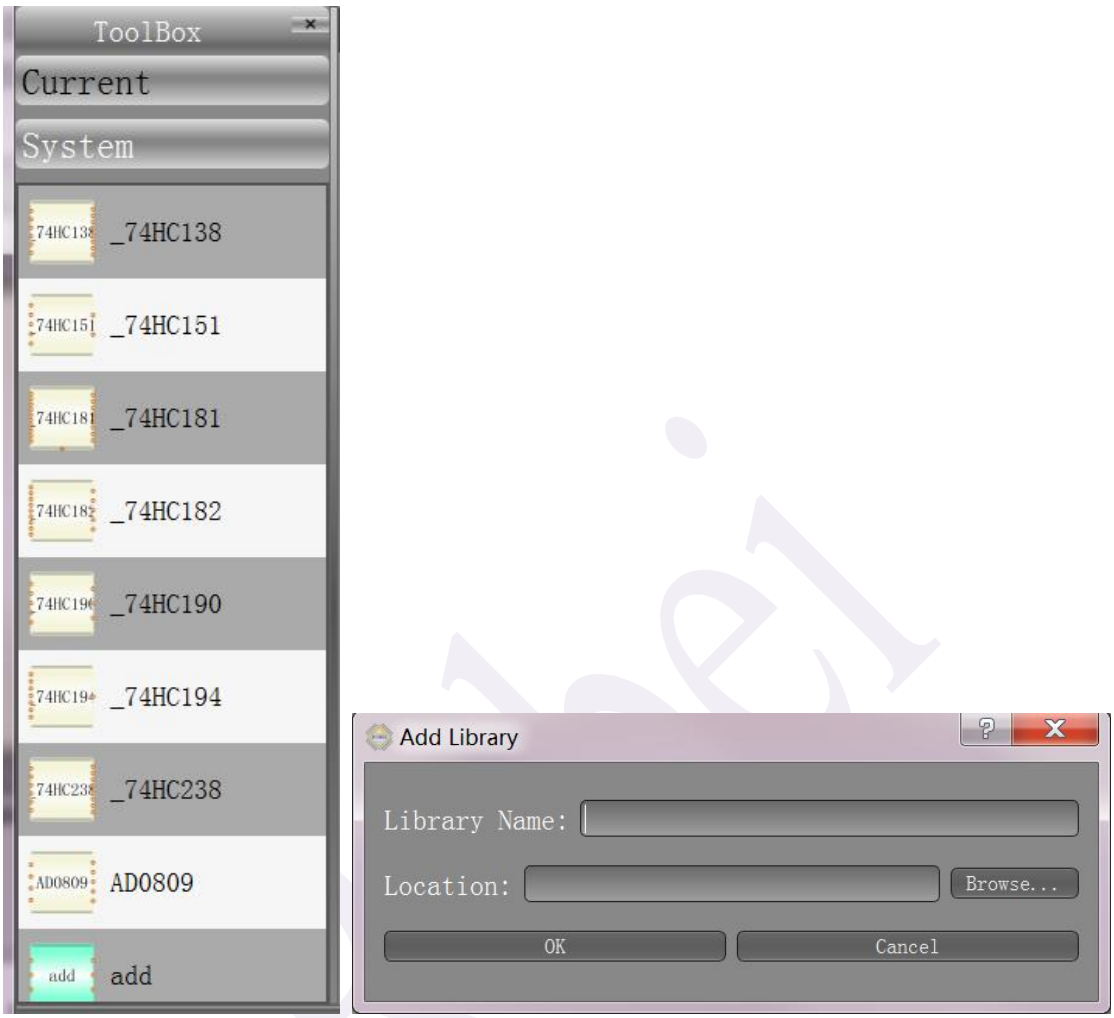


图 1-3-3 工具栏（左）和添加库对话框（右）

1.3.3. 属性栏

属性栏窗口用来展示工作区域中被选中物体的属性。用户可以修改对应的属性，并按下回车键，修改的属性会直接展示在工作区域中。模型中有些属性是受保护的，所以不能修改。

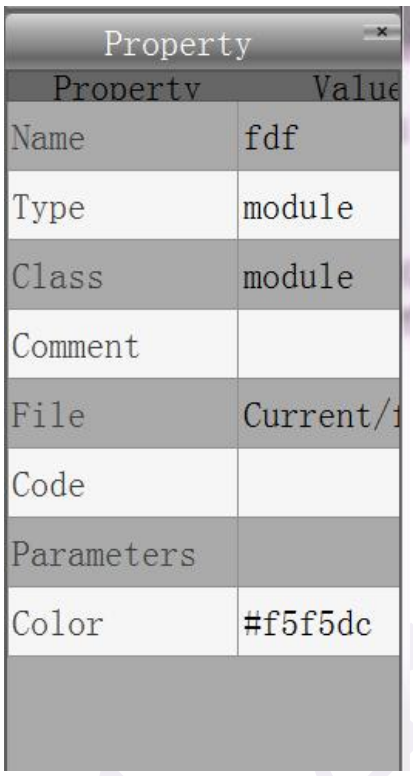
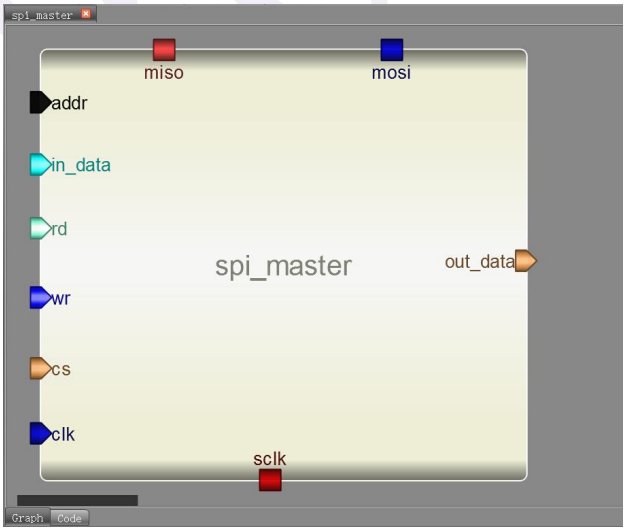


图 1-3-4 属性栏

1.3.4. 工作空间

工作空间是一个图形化设计区域，在这个空间，用户可以利用模块、模型、引脚和连接线来设计复杂的集成电路。工作空间由两个部分组成，一个是图形化设计视窗，一个是代码设计视窗，可以通过底端的“Graph”和“Code”选项卡进行切换。



 Copyright © Robei

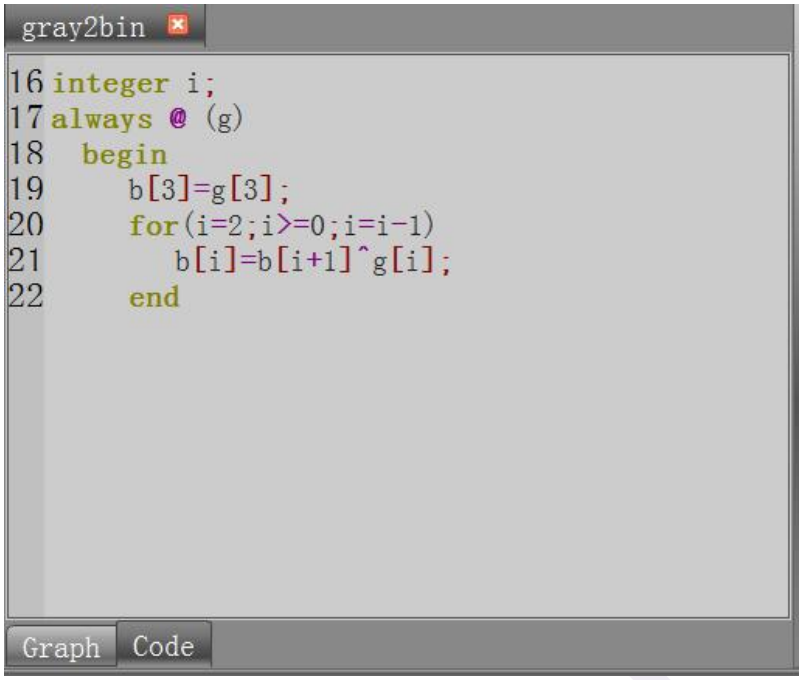


图1-3-5 图形视窗和代码视窗

1.3.5. 输出

输出窗口用来显示输出信息，包括错误信息和警告信息。

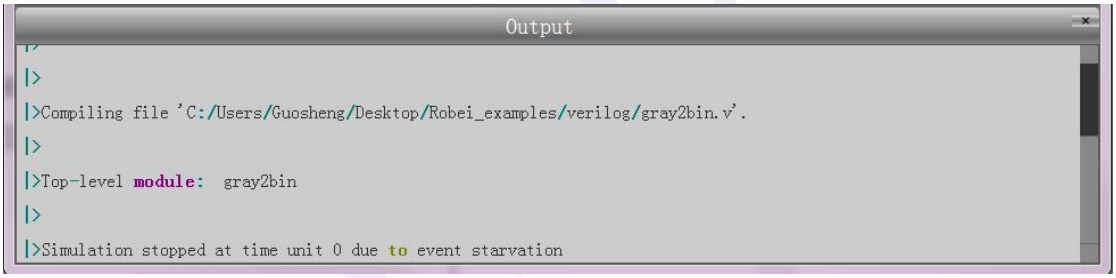


图 1-3-6 输出窗口

1.4 Robei 三元素

Robei 用三个基本元素来代表 Verilog 里面的组成部分：模块、引脚和连接线。在 Verilog 语言里，电路用一系列模块（module）来描述。一个模块可以代表一个逻辑门，一个寄存器，一个 ALU 或者一个 SOC 系统。模块是一个抽象的芯片，在这个抽象的芯片上，又存在着抽象的引脚。引脚是模块中用来与外界通信的门户，每个引脚与其它的模块进行通信，都需要一个抽象的连接线。这个连接线可以是一根线，也可以是一个总线。

1.4.1. 模块

模块是设计流程中的基本元素，可以看作是一个黑盒子。在这个黑盒子里，设计者可以放置引脚用来做通信的门户，可以放置设计好的模型和输入实现该模块功能的算法代码。根据设计阶段分类，一个模块可以细分为不同的类型。正在开发的当前模块都是“Module”类型，但是一旦设计完毕，便会保存成“Model”，这个模型可以在其它的模块中使用，而且部分属性进行了写保护，不能随意修改。模块中还有其它的类型，包括含有激励代码，用来实现仿真的“Testbench”类型，实现 FPGA 引脚分配的“Constrain”类型。

图 1-4-1 Module 可以看作一个黑盒子

（1）Module：Robei 的基本类型，当前正在设计的集成电路模块，保存后自动变成“Model”，用户可以修改任何属性。

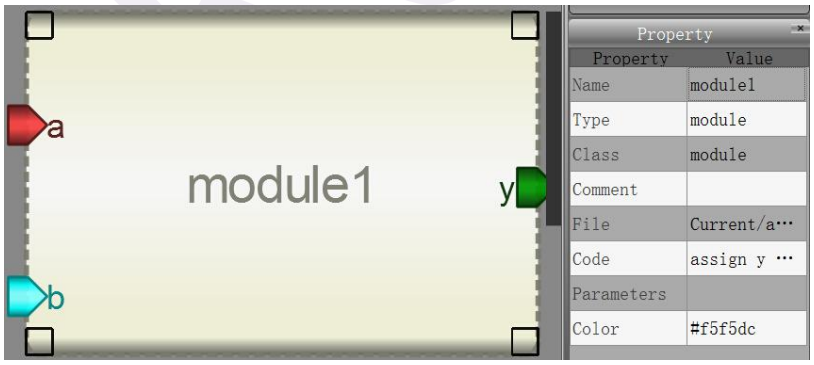


图 1-4-2 Module

（2）Model：设计好的模块在其它模块中使用，自动变成“Model”类型。部分属性进行了写保护，但是用户还是可以修改其它的属性，如颜色、名称和参数等。

 Copyright © Robei

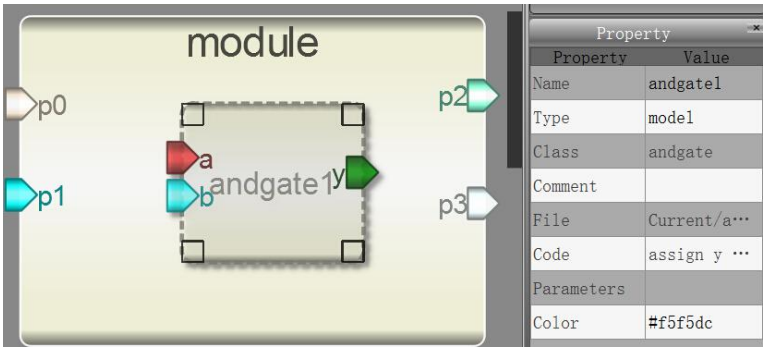


图 1-4-3 Model

（3）Testbench：测试模块对于验证设计非常重要，测试模块用来给予激励，调用设计好的模块，用来验证设计结果。如果要看到仿真结果，请确保顶层激励模块的属性设置成“testbench”，否则看不到波形分析。

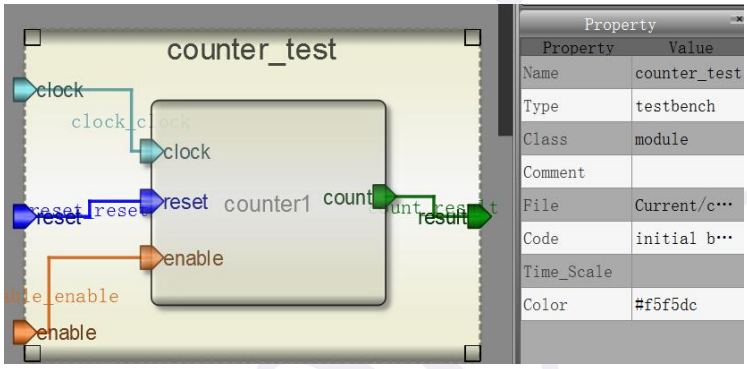


图 1-4-4 Testbench

（4）Constrain：约束文件是用来对 FPGA 引脚进行分配的文件。因为每家 FPGA 厂商的引脚分配有不同的模式，所以 Robei 会根据不同的厂商自动生成对应于该厂商的引脚约束文件，但是在 Robei 软件上进行分配的方式是统一的。比如 Xilinx ISE 的引脚约束文件是 UCF 文件，Vivado 的引脚约束文件是 XDC 文件，Altera 的引脚约束文件是 QSF 文件。如果要分配的引脚是一个总线，用户可以在连接线上声明要分配的引脚在总线中的编号。

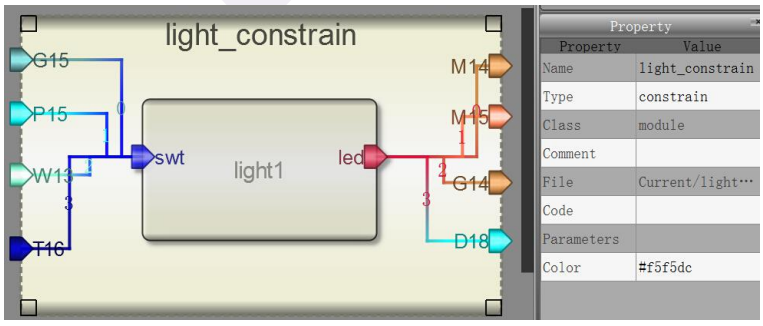


图 1-4-5 Constrain

在引脚分配之前，用户需要在菜单“Settings”里面选择“FPGA”，用来选择正确的 FPGA 厂家。额外的约束可以用代码形式写在工作空间中的代码视窗中。

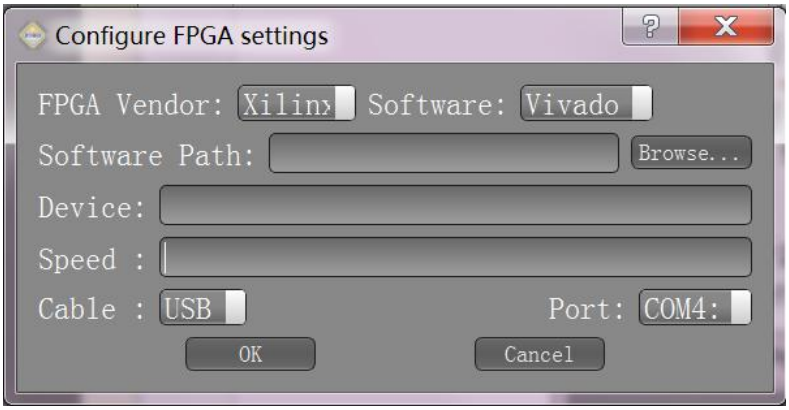


图 1-4-6 选择 FPGA 厂商

在菜单“View”的下拉菜单中选择“CodeView”， 就可以看到自动生成的引脚约束代码。

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN G15 [get_ports swt[0]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[0]]
set_property PACKAGE_PIN P15 [get_ports swt[1]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[1]]
set_property PACKAGE_PIN W13 [get_ports swt[2]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[2]]
set_property PACKAGE_PIN T16 [get_ports swt[3]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[3]]
```

```
#This file is generated by Robei!
#Pin Assignment for Altera FPGA with Software Quartus.
set_location_assignment PIN_G15 -to swt[0]
set_location_assignment PIN_P15 -to swt[1]
set_location_assignment PIN_W13 -to swt[2]
set_location_assignment PIN_T16 -to swt[3]
set_location_assignment PIN_M14 -to led[0]
set_location_assignment PIN_M15 -to led[1]
set_location_assignment PIN_G14 -to led[2]
set_location_assignment PIN_D18 -to led[3]
```

图 1-4-7 针对 Xilinx 和 Altera 公司的约束文件

1.4.2. 引脚

引脚是一个抽象的概念，可以对应于物理芯片上的一根针，或者开发板上的一个连接口，或者芯片上的一个总线。引脚是模块的门户，是模块与外界通信的接口。引脚就如同在黑盒子上打孔，从外向内钻孔的时候，孔是外面大，里面小，这是输入。从内向外钻孔的时候，孔是里面大外面小，这就是输出。还有一种既可以输入也可以输出的，就是两个方向上同时钻孔，内外大小一样。

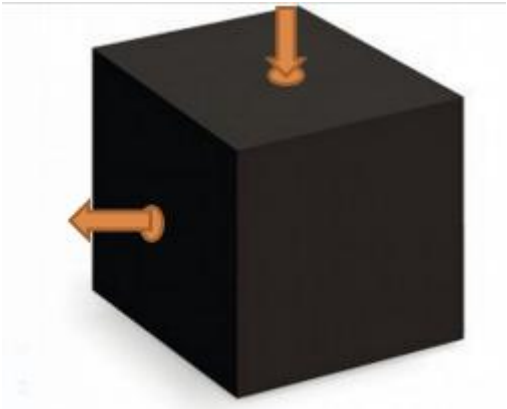


图 1-4-8 引脚对应于在黑盒子上打孔

Copyright © Robei

每个引脚的属性如图 1-4-9 所示。引脚的属性也有多种，如“reg”，“wire”，“supply”等。“Datasize”属性用来描述该引脚是单针还是总线。引脚只能在模块的边缘游走，不能进入模块也不能离开模块。当模块移动时，引脚也跟随移动。

注意：模型上的引脚的一些属性是写保护的，不能修改，但是位置和颜色信息可以随意调整。

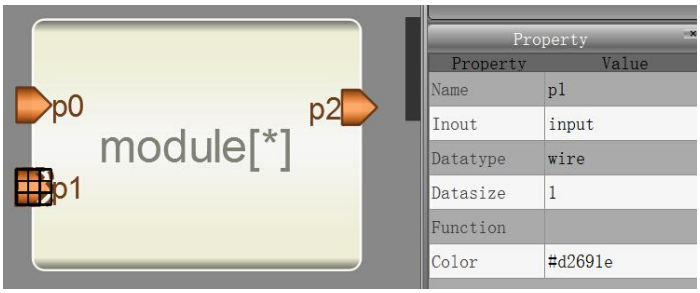


图 1-4-9 引脚

1.4.3. 连接线

连接线用来连接两个引脚，并负责信号的传输。连接线就如同连接黑盒子上的两个孔的管子，两个孔之间要想水不洒出来，就需要密封的管子进行连接。集成电路也是一样，只不过集成电路使用的是导线连接，而流入流出的不再是水，而是电。

图 1-4-10 连接线

通常情况下，连接线会继承起始引脚的颜色和数据宽度信息，然后颜色渐变为目标引脚的颜色。根据数据宽度的不同，连接线的粗细也不尽相同。

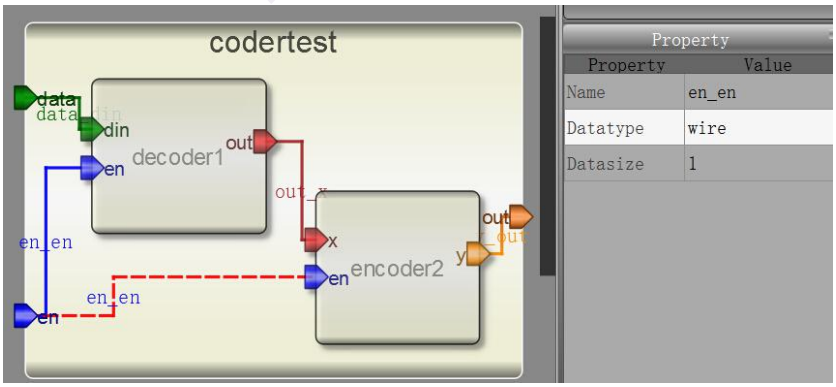


图 1-4-11 连接线

模块、引脚和连接线是 Robei 的三元素，通过这三元素，Robei 把复杂的集成电路设计简化到最简，进一步降低了对集成电路入门的要求，方便用户快速进入集成电路设计。

Robei

 Copyright © Robei

1.5 Verilog 基础

1.5.1. 数据

（1）电路四种状态

Verilog 用 4 个值来实现电平描述：0，1，Z 和 X。

0：数值 0，低电平，或者错误
1：数值 1，高电平，或者正确
Z 或者 z：高阻态
X 或者 x：未知或未初始化

（2）数值表示方法：位数+’+进制+值。

进制	示例	解释
二进制	3’b101	三位二进制数 101
八进制	9’o17	九位八进制数 17，相当于十进制的 15
十进制	12	不加任何符号代表十进制数，例子就是 12
	’d11	用 d 代表十进制，该数是十进制的 11
十六进制	64’hff01	64 位十六进制 FF01，十进制相当于 65281

表 1-5-1 数值进制表示方法

（3）数据类型

硬件的数据类型描述以驱动的方式来分类，常用的有两种：reg 和 wire。

reg：可以存储数据，如触发器
wire：连接两个引脚，不能存储数据

1.5.2. 运算符

（1）逻辑运算符

逻辑运算	运算符	例句
与	&	y=a&b;
或	|	y=a|b;
异或	^	y=a^b;
非	~	y=~a;
逻辑与	&&	y=a&&b;
逻辑或	||	y=a||b;
逻辑非	!	y=!a;

表 1-5-2 逻辑运算表

（2）算术运算符

Copyright © Robei

算术运算	运算符	例句
加	+	y=a+b;
减	-	y=a-b;
乘	*	y=a*b;
除	/	y=a/b;
取余	%	y=a%b;
左移	<<	y=a<<b;
右移	>>	y=a>>b;

表 1-5-3 算术运算表

（3）比较运算符

比较运算	运算符	例句
大于	>	y=a>b;
小于	<	y=a<b;
等于	==	y=a==b;
大于等于	>=	y=a>=b;
小于等于	<=	y=a<=b;
不等于	!=	y=a!=b;

表 1-5-4 比较运算表

1.5.3. 结构声明

由于 Robei 的存在，以下部分代码可以不用输入，省去用户的大量时间，但是用户需要了解这部分代码的存在。所以被省去的代码会在描述中提出。

1. 模块定义

Robei 的每个框图代表一个模块，每个模块的声明都由“module”开始，然后是该模块的名称，之后的括号里面包含了输入和输出的引脚。最后要写上“endmodule”。

```
module dff(d,q,clk);
    //这里开始编程
endmodule
```

```
4 module dff(
5   d,
6   q,
7   clk);
8
9   //---Ports declearation---
10  input d;
11  output [7:0] q;
12  input clk;
13
14  wire d;
15  wire [7:0] q;
16  wire clk;
17
18  //----Code starts here-----
19  always @ ( posedge clk)
20        q<=d;
21 endmodule    //dff
22
```

图 1-5-1 模块的定义

Copyright © Robei

2. 引脚定义

引脚的名称将会出现在模块定义的括号里面。模块定义完成后，在 module 和 endmodule 中间声明引脚的走向。箭头从外向内的是输入引脚，从内向外的是输出引脚，无箭头的是既可以做输入也可以做输出。有数据宽度的用中括号给出。紧接着，声明每个引脚信号的类型，一般是 wire 或者 reg。

```
input p0;
inout p1;
output [3:0] p2;
wire p0;
wire p1;
reg [3:0] p2;
```

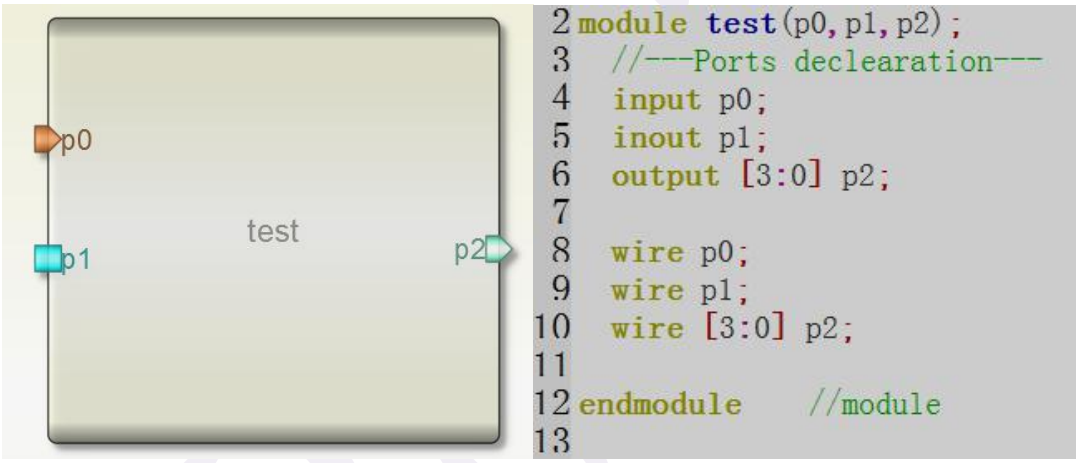


图 1-5-2 引脚的定义

3. 连接线

连接线的定义是用连接线类型加上位宽和名称形成的，与引脚的类型定义类似，但是在 Robei 中，顶层模块与子模块的连接线可以不声明，直接连接引脚，所以部分连接线并不存在于代码中。

```
wire clkout_clk;
```

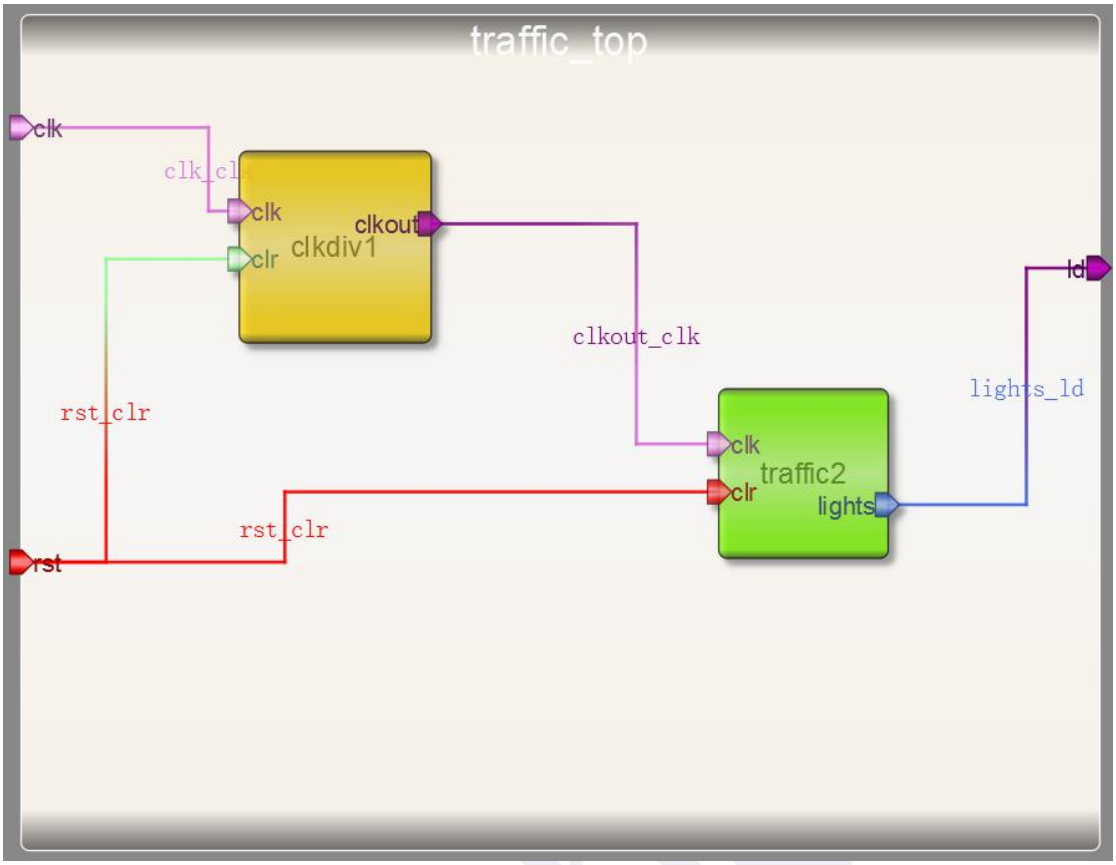


图 1-5-3 连接线的定义，只有 clkout_clk 声明了

4. 例化

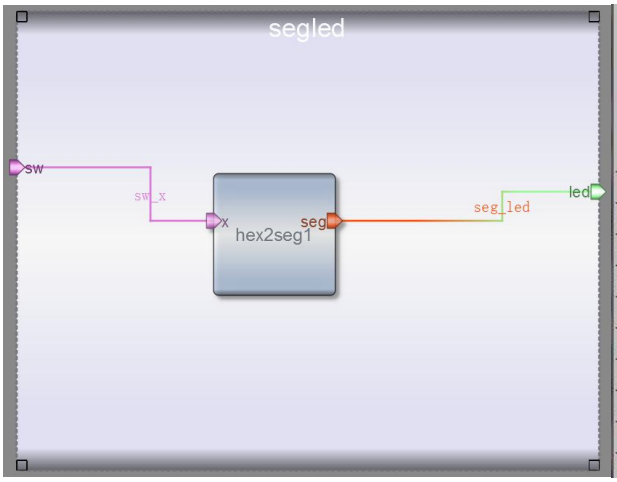
例化的时候根据模块的连接方式，确定每个引脚相连的引脚或者连接线，通过类似模块声明的方式进行例化。在例化的时候，有些时候需要空接一些信号，输入管脚悬空，该管脚输入为高阻 Z，输出管脚悬空，该管脚废弃不用。

```
Design u_2(   .(端口 1(u_1 的端口 1),
              .(端口 2(u_1 的端口 2),
              .(端口 3(u_1 的端口 3),
                        …… );
```

实际举例如 DFF 的例化：

```
DFF d1 (.Q(QS),
.Qbar ( ),
.Data (D),
.Preset ( ), // 该管脚悬空
.Clock (CK) );
```


Copyright © Robei




```
5 module segled(
6   sw,
7   led);
8
9   //---Ports declearation---
10  input [3:0] sw;
11  output [6:0] led;
12
13  wire [3:0] sw;
14  wire [6:0] led;
15
16  //---Module instantiation---
17  hex2seg hex2seg1(
18    .x(sw),
19    .seg(led));
```

图 1-5-4 例化声明

1.5.4. 代码撰写

1. 赋值语句

常用的赋值语句如 assign，assign 起的作用是将一个信号与另外一个信号直接进行相连，任何信号的变化都是同步的。Assign 中等号左边信号必须是 wire 型。

```
wire a, b, y;
assign y = a & b;
```

图 1-5-5 always 语句

2. 分支语句

if else

类似于 C 语言中的 if else 的写法。如果 if else 写全，就会生成寄存器，但是只有 if 没有 else 的语句，将会生成锁存器。在硬件设计中应该尽量避免锁存器的产生。当有多条语句存在于 if else 中间的时候，需要用 begin end 将多条语句进行包含，此时 begin end 相当于 C 语言中的{}。

```
always @ (posedge clk)
 if (reset) begin
    dff <= 0;
 end else begin
    dff <= din;
 end
```

```
always @ (posedge clk)
 if (enable) begin
   latch <= din;
 end
```

图 1-5-6 if else 语句

Case

类似于 C 语言中的 case 的写法。如果 case 写全并且 default 值设置好，就会生成寄存器，否则将会生成锁存器。

```
always @ (a or b or c or d or sel)
 case (sel)
   0: y = a;
   1: y = b;
   2: y = c;
   3: y = d;
   defualt: y = 0;
endcase
```

图 1-5-7 Case 语句

3. 循环语句

For

类似于 C 语言中的 For 的写法。但是 Verilog 中，没有自加的语句，所以没有 i++，只能用 i=i+1。

```
for (i = 0; i < 256; i = i + 1) begin
   ram[i] <= 0;
end
```

图 1-5-8 For 语句

While

类似于 C 语言中的 While 的写法。While 语句在执行时，首先判断循环执行条件表达式是否为真，如果真，执行后面的语句块， 然后再重新判断循环执行条件表达式是否为真，为真的话， 再执行一遍后面的语句块，如此不断，直到条件表达式不为真。

```
while (data[0] == 0) begin
  loc = loc + 1;
  data = data >>1;
end
```

图 1-5-9 While 语句

4. 初始化与重复执行

Initial

 Copyright © Robei

Verilog 中用 initial 进行初始化，initial 只执行一次，在 0 时刻执行。主要用在仿真测试模块中。

```
module initial_example();
    reg clk,reset,enable,data;
    initial begin
      clk = 0;
      reset = 0;
      enable = 0;
      data = 0;
    end
endmodule
```

图 1-5-10 initial 语句

Always

Verilog 中 always 一直重复执行到程序结束。Always 有自己的敏感信号列表，用 always@（敏感信号 1 or 敏感信号 2 or ...）来表示，当敏感信号发生改变的时候更新状态。

```
module always_example();
    reg clk,reset,enable,q_in,data;
   always @ (posedge clk)
      if (reset) begin
         data <= 0;
      end else if (enable) begin
          data <= q_in;
      end
endmodule
```

图 1-5-11 always 语句

只有寄存器类型的信号才可以在 always 和 initial 语句中进行赋值，类型定义通过 reg 语句实现。

5. 阻塞式赋值与非阻塞式赋值

阻塞赋值（ = ）

在串行语句块中，阻塞赋值语句按照它们在块中的排列顺序依次执行，即前一条语句没有完成赋值之前，后面的语句不可能被执行，换言之，后面的语句被阻塞了。在 always 中 begin...end 语句块中所有语句是顺序执行的，阻塞赋值是在上一条语句完全完成之后，才开始执行下一条语句的。

```
always @ (posedge clk)
begin
  oQA = iD;
  oQB = oQA;
end
```

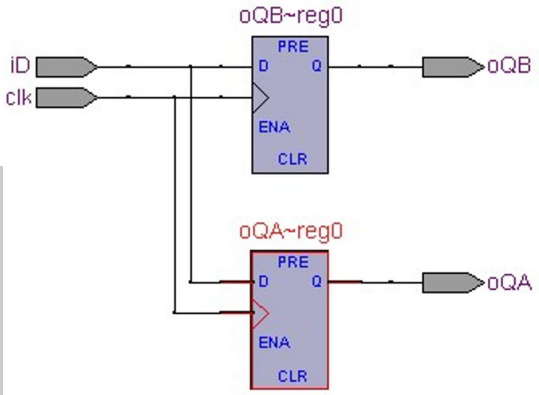


图 1-5-12 阻塞式赋值语句

非阻塞赋值（ <= ）

先计算右边表达式的值并暂存在一个暂存器中，iD 的值被保存在一个寄存器中，而 oQA 当前的值被保存在另一个寄存器中，在 begin 和 end 之间所有语句的右边表达式都被计算并存储完后，对左边的寄存器变量的赋值才会进行。这样 oQB 得到的是 oQA 的原始值而不是 iD。

```
always @ (posedge clk)
begin
  oQA <= iD;
  oQB <= oQA;
end
```

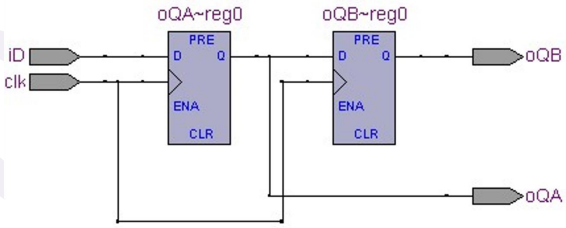


图 1-5-12 非阻塞式赋值语句

1.5.5. 一个模块的总结

module 模块名(引脚 1,引脚 2,……);
input [数据宽度] 输入引脚;
output[数据宽度] 输出引脚;
parameter 参数=默认值;
reg[数据宽度] 寄存性信号;
wire[数据宽度] 非寄存性信号;
//例化写法
模型名称 #(参数 1，参数 2，……) 例化名 （
.模型引脚 1（连接的信号 1），
.模型引脚 2（连接的信号 2），
……

 Copyright © Robei

```
                    );
    //其他描述算法
    //assign
    assign  信号 1=信号 2；

    //if else
    if (条件) begin
        执行语句；
    end else begin
        执行语句；
    end

    //case
    case (变量)：
        第一个值：执行语句；
        第二个值：执行语句；
        ……
        default：默认执行语句；
    endcase

    //while
    while（条件）begin
          执行语句；
  end

  //always
    always@(敏感信号 1 or  敏感信号 2 or …..)
        语句嵌套；

  //for
    for(i=0;i<10;i=i+1) begin
        语句嵌套；
    end

endmodule
```

以上只是简单的总结，学习的过程就是实践积累的过程，只有不断的练习，才能真正掌握集成电路设计。

Robei

1.6. 第一天的总结

第一天的学习任务非常的繁重也非常的基础，只有掌握好这一天的内容，才能够快速动手掌握其他的环节学习。第一天我们了解了 EDA 的发展历程，了解了 Robei 的特点和三元素，安装并注册了 Robei 软件和初步学习了 Verilog 语言，这对后续的学习打下了坚实的基础。后面的学习以动手为主，希望读者能亲自动手跟着学习。

Robei

 Copyright © Robei

第二天：实例入手，体验若贝

今天我们通过几个比较简单的实例，简要介绍一下 Robei 软件的模型建立、端口属性修改、代码输入以及仿真和波形查看的过程。同时也可以通过代码的设计巩固一下对 Verilog 语言应用的熟练程度。这些设计案例可能看起来比较基础，但是其实现的功能都是在 FPGA 设计中很常用很重要的。读者需要直接动手跟着操作，以便能顺利掌握 Robei 的设计流程和体验 Verilog 的编程方式。通过今天的学习，读者可以实现从零到一的跳跃。

Robei

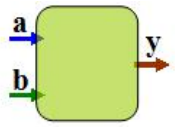
2.1 实例一 逻辑门设计

2.1.1. 本章导读

数字逻辑是芯片电路的基本组成部分。本次设计主要分析数字逻辑门在 Robei 软件中利用 Verilog 语言实现的方式，并通过该设计让参与者快速体验并掌握“图形化+代码”的新型设计模式。

理论分析

逻辑门是数字电路的基础，常见的数字电路逻辑门有与门，或门，非门，与非门，或非门和异或门等。本次设计重点讨论其中的几个逻辑门用 Verilog 在 Robei 软件中的设计和仿真。以常见的与门（图 2-1-1）为例，如图 2-1-1 所示，通过其真值表可以看出，只有当两个输入同时为 1 的时候，输出才是 1，其他情况下均为 0。与门的数学表达式为：y=a&b。



a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

图 2-1-1 与门示意图和真值表

软件准备

熟悉 Robei 软件。在 Robei 官方网站（http://www.robei.com）下载最新版 Robei 软件，并安装。打开 Robei 软件，熟悉 Robei 软件的结构和菜单。将鼠标放在工具栏的每个图标上查看图标所代表的内容。在下拉菜单中点击“Help”，查看 Robei 最新版用户使用说明书。

2.2.2. 设计流程

1. 模型设计

（1）新建一个模型。点击工具栏上的 图标，或者点击菜单“File”然后在下拉菜单中选择“New”，会有一个对话框弹出来（如图 2-1-2 所示）。在弹出的对话框中设置你所设计的模型。

图 2-1-2 所对应的每项分析如下：

Copyright © Robei

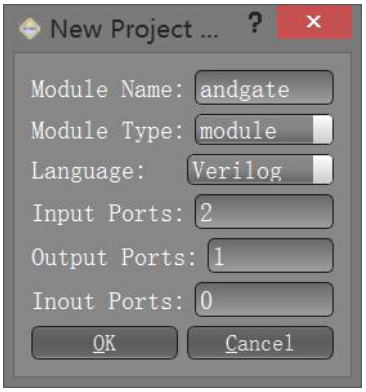


图 2-1-2 新建一个项目

Module Name：模块名称，这里我们想创建一个叫 andgate 的模块，输入 andgate。

Module Type：模块类型，Robei 目前支持 3 种类型，“module”，“testbench”和“constrain”。这里我们创建的是一个模块，选择“module”。

Language：设计语言，这里只有一种设计语言 Verilog。

Input Ports：输入引脚数目，我们设计的模块有 2 个输入引脚 a 和 b，所以输入 2。

Output Ports：输出引脚数目，我们设计的模块只有 1 个输出引脚 y，所以输入 1。

Inout Ports：既可以作为输入又可以作为输出引脚的数目，我们设计的模块没有用到该类型引脚，所以输入 0。

参数填写完成后点击“OK”按钮，Robei 就会生成一个新的模块，名字就是 andgate，如图 2-1-3 所示：

图 2-1-3 与门逻辑界面图

（2）修改模型。在自动生成的界面图上用鼠标选中输入引脚“p0”，右侧的属性编辑栏就会展示该引脚相对应的属性如图 2-1-4 所示。每条属性有其对应的名称。为了跟设计名称一致，我们把 p0 的名称改成 a，p1 的名称改成 b，p2 的名称改成 y。修改的方法是在属性编辑器 Name 栏里面修改并点回车。为了区分每个引脚，我们可以修改每个引脚的 Color 值，并点回车保存。修改完成后如图 2-1-5 所示：

Copyright © Robei

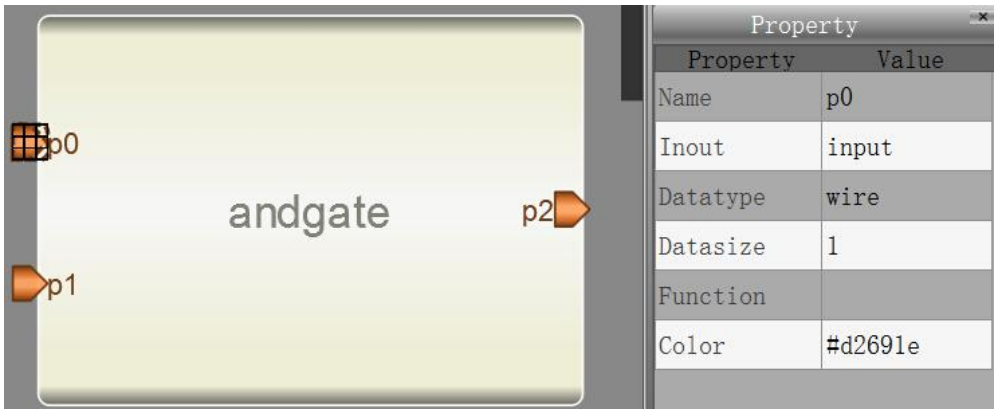


图 2-1-4 展示引脚“p0”的属性

图 2-1-5 引脚名称和颜色修改后的界面

（3）输入算法。点击模型下方的 Code（如图 2-1-6 所示）进入代码设计区。

图 2-1-6 点击 Code 输入算法

在代码设计区内输入以下 Verilog 代码：

```
assign y = a & b;
```

该代码实现的是与门逻辑运算。如图 2-1-7 所示：

 Copyright © Robei

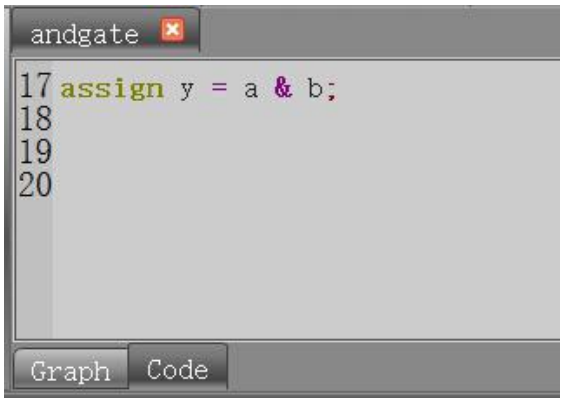


图 2-1-7 算法输入

（4）保存。点击工具栏 图标，或者点击菜单“File”中的下拉菜单“Saveas”，会出现如图 2-1-8 的界面，将模型另存到一个文件夹中。

注意：1. 保存的路径中不能含有中文和空格

2. 保存文件名不能以数字和特殊字符开头

3. 相关的文件要保存在同一路径下

4. 保存的文件名会显示成当前设计的模块名称

5. 命名时不能命名成 verilog 的关键字，如“module”，“if”等

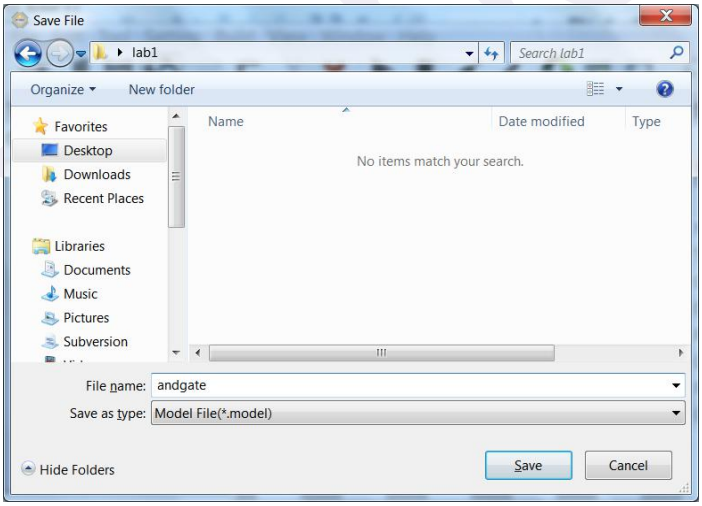


图 2-1-8 保存为模型

（5）运行。在工具栏点击 按钮或者点击菜单“Build”的下拉菜单“Run”，执行代码检查。如果有错误，会在输出窗口中显示，错误行数在 code 中显示的行中，可以通过修改该行或者上下行，错误行数不在 code 显示的范围中，需要修改界面。如果没有错误提示，恭喜你，模型 andgate 设计完成。

2. 测试文件设计

（1）新建一个文件。点击工具栏上的 按钮图标，在弹出的对话框中参照图 2-1-9 进行设计。

注意：如果 Module Type 不设置成 testbench，仿真将看不到波形。

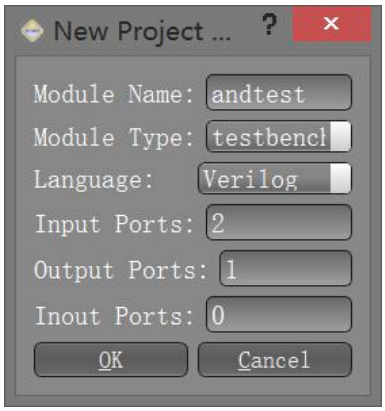


图 2-1-9 新建测试文件

（2）修改各个引脚的颜色。选中每个引脚，在属性栏中修改其颜色，方便区分不同的引脚信号，如图 2-1-20 所示。

图 2-1-10 修改引脚颜色

（3）另存为测试文件。点击工具栏 图标，出现如图 2-1-11 的界面，将测试文件保存到 andgate 模型所在的文件夹下。

注意：1.保存的路径中不能含有中文和空格

2. 保存文件名不能以数字和特殊字符开头

3. 必须保存到和 andgate 同一路径下，否则 Toolbox 中找不到 andgate 模块

4. 保存的文件名会显示成当前设计的模块名称

Copyright © Robei

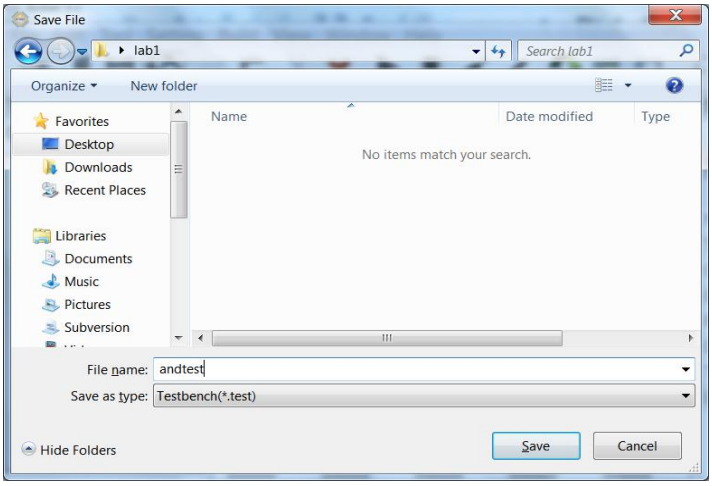


图 2-1-11 保存测试文件

（4）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现一个 andgate 模型，单击该模型并在 andtest 上添加。

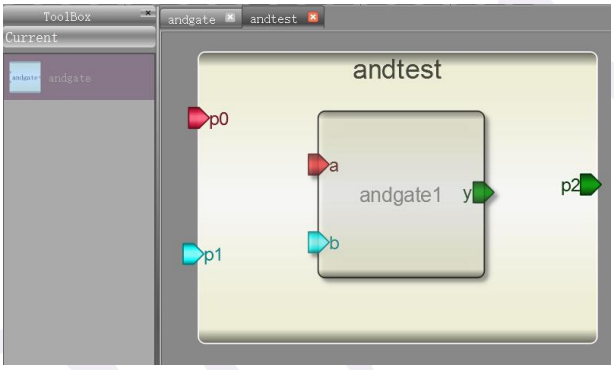


图 2-1-12 添加模型

（5）连接引脚。点击工具栏中的 图标，或者选择菜单“Tool”中的“Connect”，连接引脚 p0 到 a，p1 到 b 和 y 到 p2，如图 2-1-13 所示。这个时候，注意查看连接线的颜色。

如果鼠标要变回选择模式，点击图标 。

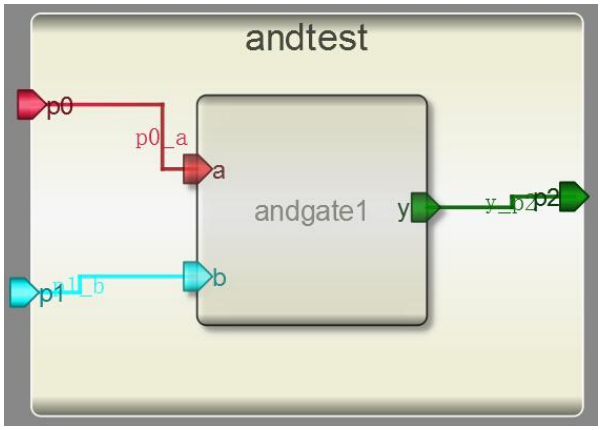


图 2-1-13 连接引脚

（6）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

```
initial begin
    p0 = 0;
    p1 = 0;
    #1
    p0 = 1;
    #1
    p1 = 1;
    #1
    p0 = 0;
    #1
    p1 = 0;
    #1
    $finish;
end
```

图 2-1-14 激励代码

（7）执行仿真并查看波形。点击工具栏 ，查看输出信息，检查没有错误之后点击 进行仿真，再点击 或者菜单“View”中的“Waveview”，波形查看器就会打开，如图 2-1-15 所示。如果顶层模块的“Module Type”不是 testbench，将不会看到波形。

Copyright © Robei

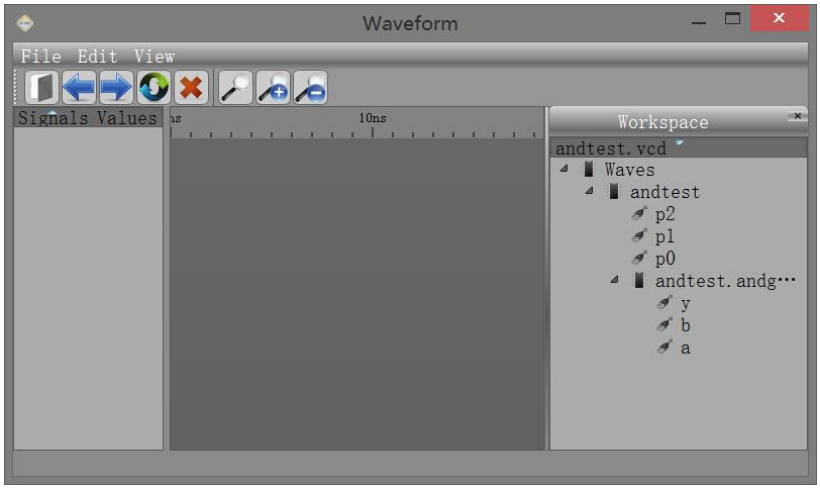


图 2-1-15 波形查看器

（8）点击右侧 Workspace 中的信号，进行添加并查看。点击波形查看器工具栏上的 图标进行自动缩放。分析仿真结果并对照真值表，查看设计正确与否。

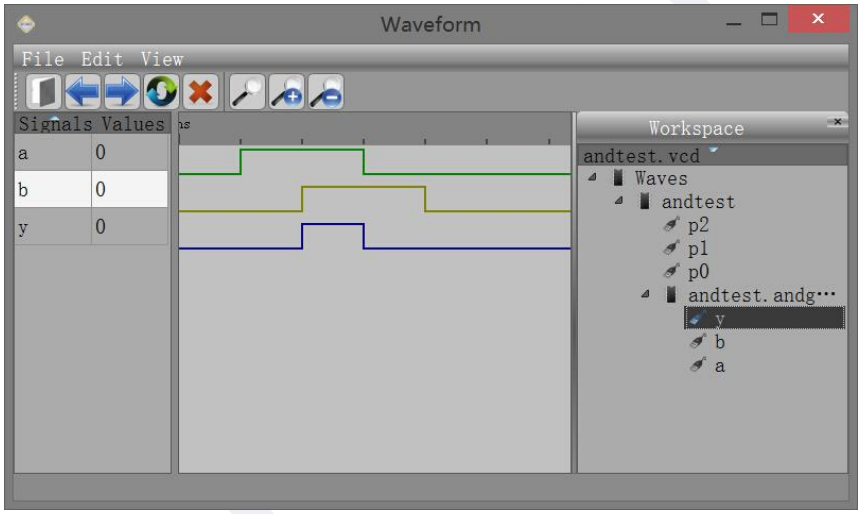


图 2-1-16 查看波形

2.1.3. 问题与思考

本次案例中以与门作为例子进行设计，你如何经过简单改动，按照同样的方式来设计或门，非门和以或门并进行仿真验证？

逻辑门	运算符	Verilog 算法代码	对应芯片
与门	&	assign y=a&b;	74LS08
或门	|	assign y=a|b;	74LS32
异或门	^	assign y=a^b;	74LS86
与非门	~&	assign y=~(a&b);	74LS00
或非门	~|	assign y=~(a|b);	74LS36

Copyright © Robei

2.1.4. 常见问题

（1）我为什么仿真之后看不到波形？

Robei 的模型有四种类型：“module”，“model”，“testbench” 和“constrain”。 如果你想仿真之后看波形应该将顶层的仿真模块类型设置成“testbench”。同时，testbench 的模块输入端口类型应为“reg”，输出类型应为“wire”。

（2）“model”和“module”有什么区别？

正在设计的模块叫做“module”，一旦设计完成，并把此模块应用到其它的设计模块的时候，该模块的类型自动变成“model”。“model”的一些属性不可更改，是被保护了的。

（3）怎么样看到模块的完整代码？

在“Code”中，你只能看到用户输入的代码部分，而且这些代码不是从第一行开始计数的。点击菜单“View”中的下拉菜单“CodeView”，你可以看到所有的代码，包括自动生成的。

（4）我没有注册能不能仿真看波形？

可以。但是仿真的模块数目有限制。

Robei

Copyright © Robei

2.2 实例二 计数器

2.2.1. 本章导读

计数器在数字逻辑设计中的应用十分广泛，可以对时钟信号进行计数，分频和产生序列信号，也可以用在计时器和串并转换等电路。这次我们就来学习一下如何用 Robei 和 Verilog 语言来设计一个 4 比特计数器。

设计要求

计数器对每个时钟脉冲进行计数，并将数值输出出来。现在我们来设计一个 4 比特的计数器，其范围在 0～F 之间，也就是计数到最大值 16。设计波形要求如图 2-2-1 所示：

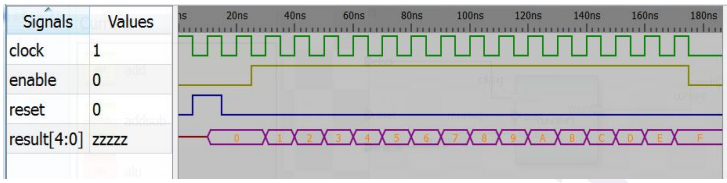


图 2-2-1 计数器输出波形要求

2.2.2. 设计流程

1. 模型设计

（1）新建一个模型。点击工具栏上的 图标，或者点击菜单“File”然后在下拉菜单中选择“New”，会有一个对话框弹出来（如图 2-2-2 所示）。在弹出的对话框中设置你所设计的模型。

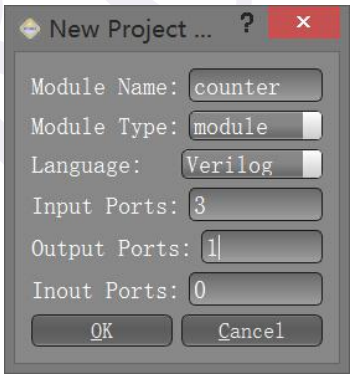


图 2-2-2 新建一个项目

参数填写完成后点击“OK”按钮，Robei 就会生成一个新的模块，名字就是 counter，如图 2-2-3 所示：

图 2-2-3 计数器界面图

（2）修改模型。在自动生成的界面图上进行名称地修改，输入引脚为 clock，enable 和 reset，输出引脚修改成 count。其中 count 引脚的“Datasize”为 4 比特，读者可以输入 4，也可以输入 3:0。为了区分每个引脚，我们可以修改每个引脚的 Color 值，并点回车保存。修改完成后如图 2-2-4 所示。如果选中模块，按“F1”键，就会自动生成一个 Datasheet，如图 2-2-5 所示：

图 2-2-4 修改引脚属性

Name	Inout	DataType	Datasize	Function
clock	input	wire	1	
reset	input	wire	1	
enable	input	wire	1	
count	output	reg	4	

图 2-2-5 “Datasheet”截图

（3）输入算法。点击模型下方的 Code（如图 2-2-6 所示）进入代码设计区。

在代码设计区内输入以下 Verilog 代码：

```
always @ (posedge clock)
begin
    if (reset == 1)
    begin
        count<= 0;
    end
    else if (enable == 1)
    begin
```

 Copyright © Robei

```
            count <= count + 1;
        end
    end
```

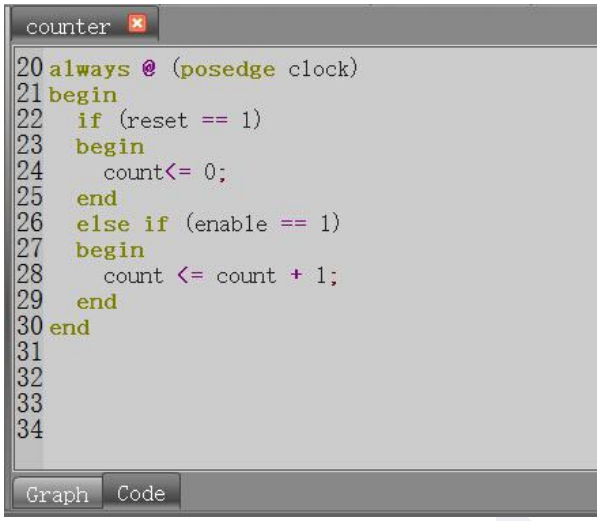


图 2-2-6 点击 Code 输入算法

（4）保存。点击工具栏 图标，或者点击菜单“File”中的下拉菜单“Saveas”，将模型另存到一个文件夹中。

（5）运行。在工具栏点击 或者点击菜单“Build”的下拉菜单“Run”，执行代码检查。如果有错误，会在输出窗口中显示。如果没有错误提示，恭喜，模型 counter 设计完成。

2. 测试文件设计

（1）新建一个文件。点击工具栏上的 图标，在弹出的对话框中参照图 2-2-7 进行设计。

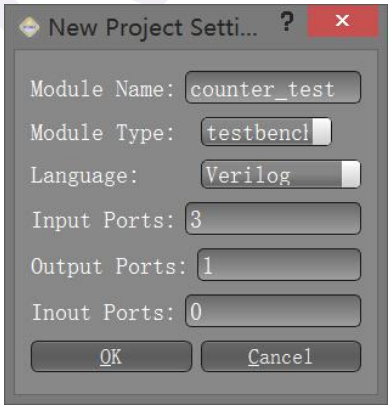


图 2-2-7 新建测试文件

（2）修改各个引脚的颜色。选中每个引脚，在属性栏中对照图 2-2-8 进行修改引脚属性，并修改其颜色，方便区分不同的引脚信号。

Copyright © Robei

Ports:

Name	Inout	DataType	Datasize
clock	input	reg	1
reset	input	reg	1
enable	input	reg	1
result	output	wire	4

图 2-2-8 引脚属性表

（3）另存为测试文件。点击工具栏 图标，将测试文件保存到 counter 模型所在的文件夹下。

（4）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现一个 counter 模型，单击该模型并在 counter_test 上添加，如图 2-2-9 所示。

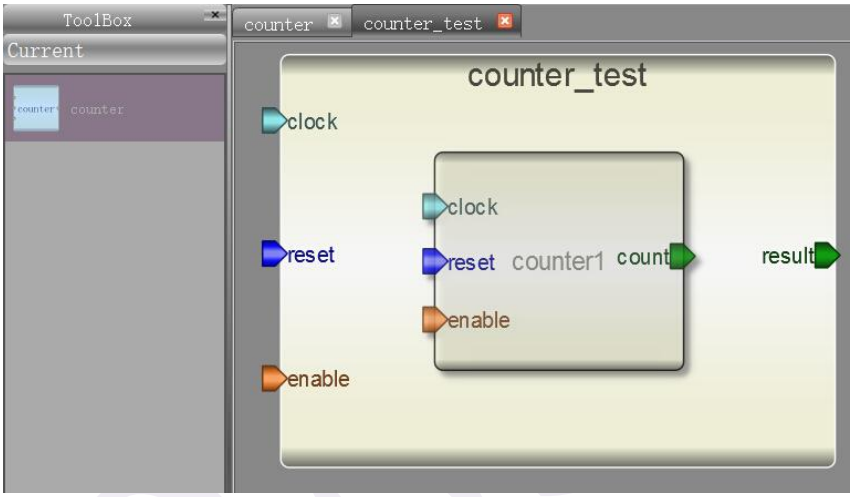


图 2-2-9 添加模型

（5）连接引脚。点击工具栏中的 图标，或者选择菜单“Tool”中的“Connect”，如图 2-2-10 所示，连接引脚。这个时候，注意查看连接线的颜色。如果鼠标要变回选择模式，点击图标 。

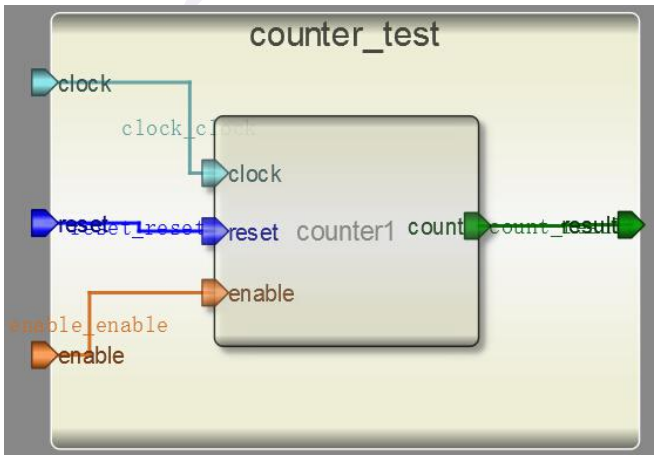


图 2-2-10 连接引脚

Copyright © Robei

（6）输入激励。点击测试模块下方的“Code”，输入激励算法，如图 2-2-11 所示。激励代码在结束的时候要用$finish 结束。

```
initial begin
    clock = 1;
    reset = 0;
    enable = 0;
    #5 reset = 1;
    #10 reset = 0;
    #10 enable = 1;
    #150 enable = 0;
    #5 $finish;
end
always begin
    #5 clock=~clock;
end
```

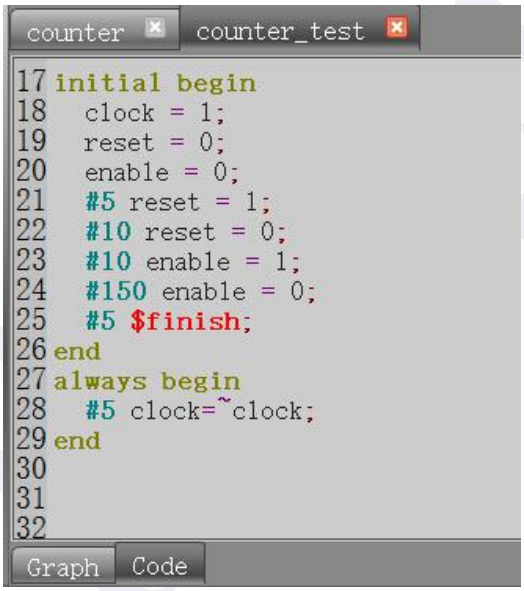


图 2-2-11 激励代码

（7）执行仿真并查看波形。点击工具栏 ，查看输出信息，检查没有错误之后点击 进行仿真，再点击 或者菜单“View”中的“Waveview”，波形查看器就会打开。点击右侧 Workspace 中的信号，进行添加并查看，如图 2-2-12 所示。点击波形查看器工具栏上的 图标进行自动缩放。分析仿真结果并对照真值表，查看设计波形与设计要求是否一致。

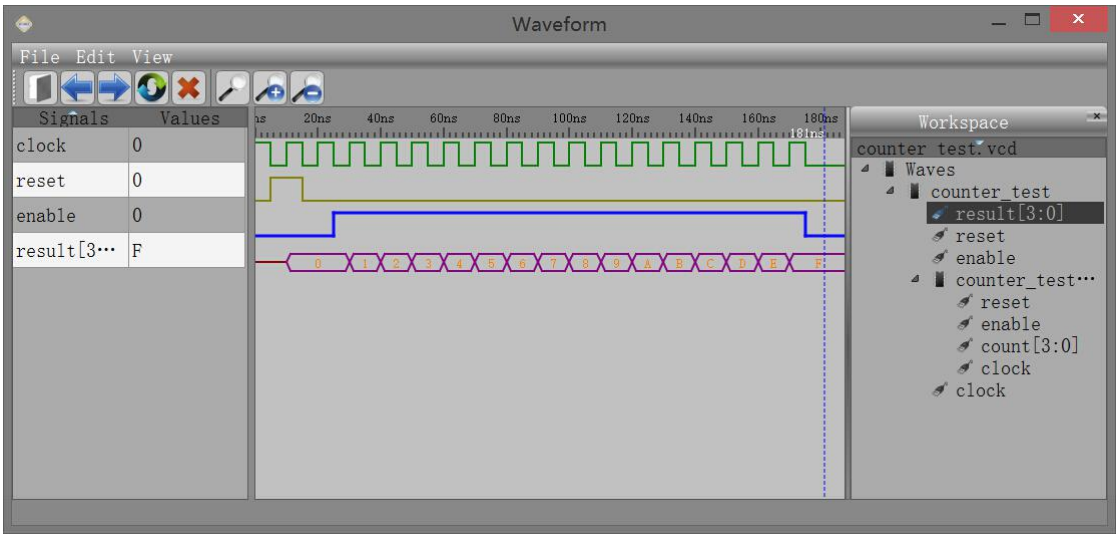


图 2-2-12 查看波形

2.2.3. 问题与思考

（1）如何利用 Robei 设计一个逆向计数器？计数开始时为 F，每个时钟信号到来计数器减一。测试你的逆向计数器。

（2）如何利用计数器实现占空比为 50% 的 2 分频，4 分频和 16 分频？提示：占空比：高电平持续时间在一个总周期所占的比率。

Copyright © Robei

2.3 实例三 编译码器

2.3.1. 本章导读

通过设计简单的编译码器实现对数据的转换。常见的编码方式有格雷码，BCD 码和 8-3 线编码器，16-4 线编码器等。本次设计以 8-3 线优先编码和 3-8 线译码器作为例子，进行数据的编译码设计。

设计原理

优先编码器是将多个二进制输入压缩成更少数目输出的电路算法。优先编码器常用于处理最高优先级请求时控制中断请求。8-3 线编码器是将输入为 8 比特的数据以 3 比特的方式描述出来。8 根输入线路中每次只有一个线路为高电平，其余为低电平。相反，3-8 译码器是用 8 根线对输入的 3 根线数据进行电平转换。如表 2-3-1。

x[7]	x[6]	x[5]	x[4]	x[3]	x[2]	x[1]	x[0]		y[2]	y[1]	y[0]
0	0	0	0	0	0	0	1		0	0	0
0	0	0	0	0	0	1	0		0	0	1
0	0	0	0	0	1	0	0		0	1	0
0	0	0	0	1	0	0	0		0	1	1
0	0	0	1	0	0	0	0		1	0	0
0	0	1	0	0	0	0	0		1	0	1
0	1	0	0	0	0	0	0		1	1	0
1	0	0	0	0	0	0	0		1	1	1

表 2-3-1 8-3 编码器真值表

2.3.2. 设计流程

1. 编码器模型设计

（1）新建一个模型命名为 encoder，类型为 module，同时引脚设置为 2 输入 1 输出。每个引脚的属性和名称参照图 2-3-1 进行对应的修改。

Name	Inout	DataType	Datasize	Function
x	input	wire	7:0	input data
en	input	wire	1	enable
y	output	reg	2:0	output

图 2-3-1 引脚属性

图 2-3-2 编码器界面图

（2）添加代码。点击模型下方的 Code（如图 2-3-3 所示）添加代码。

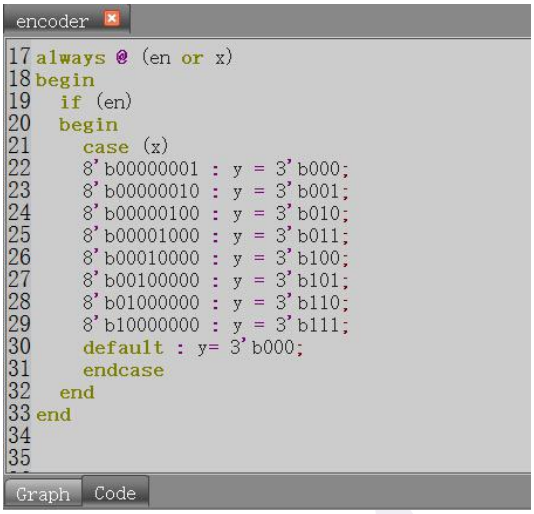


图 2-3-3 点击 Code 输入算法

在代码设计区内输入以下 Verilog 代码：

```
always @ (en or x)
begin
    if (en)
    begin
        case (x)
            8'b00000001 : y = 3'b000;
            8'b00000010 : y = 3'b001;
            8'b00000100 : y = 3'b010;
            8'b00001000 : y = 3'b011;
            8'b00010000 : y = 3'b100;
            8'b00100000 : y = 3'b101;
            8'b01000000 : y = 3'b110;
            8'b10000000 : y = 3'b111;
            default : y= 3'b000;
        endcase
    end
end
```

（3）保存。点击工具栏 图标，或者点击菜单“File”中的下拉菜单“Saveas”，将模型另存到一个文件夹中。

（4）运行。在工具栏点击 或者点击菜单“Build”的下拉菜单“Run”，执行代码检查。如果有错误，会在输出窗口中显示。

2. 译码器模型设计

Copyright © Robei

（1）新建一个模型命名为 decoder，类型为 module，同时引脚设置为 2 输入 1 输出。每个引脚的属性和名称参照图 2-3-4 进行对应的修改。

Name	Inout	DataType	Datasize	Function
din	input	wire	2:0	input data
en	input	wire	1	enable
out	output	wire	7:0	output

图 2-3-4 引脚属性

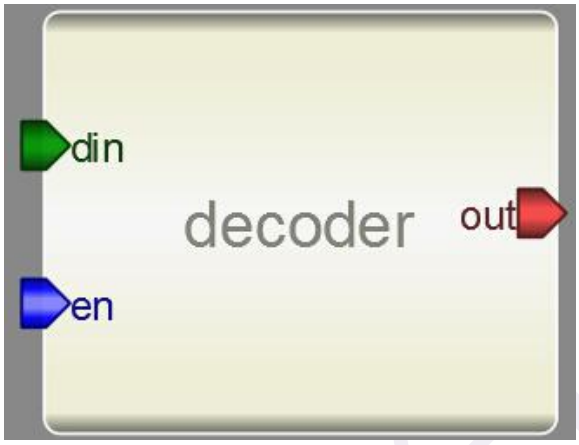


图 2-3-5 译码器界面图

（2）添加代码。点击模型下方的 Code（如图 2-3-6 所示）添加代码。

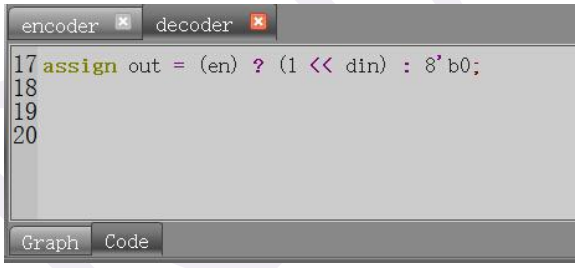


图 2-3-6 点击 Code 输入算法

在代码设计区内输入以下 Verilog 代码：

```
assign out = (en) ? (1 << din) : 8'b0;
```

（3）保存。点击工具栏 图标，将模型存到与 encoder 相同的文件夹中。

（4）运行。在工具栏点击 执行代码检查。如果有错误，会在输出窗口中显示。

3. 测试文件设计

（1）新建一个 2 输入 1 输出的测试文件。点击工具栏上的 图标，在弹出的对话框中参照图 2-3-7 进行设计。

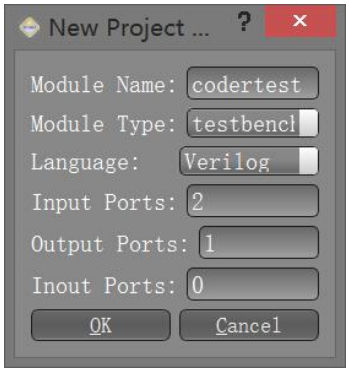


图 2-3-7 新建测试文件

（2）修改各个引脚的颜色。选中每个引脚，在属性栏中对照图 2-3-8 进行修改引脚属性，并修改其颜色，方便区分不同的引脚信号。

Name	Inout	DataType	Datasize	Function
data	input	reg	2:0	signal data
en	input	reg	1	enable
out	output	wire	2:0	output data

图 2-3-8 引脚属性表

（3）另存为测试文件。点击工具栏中的 图标，将测试文件保存到 decoder 和 encoder 模型所在的文件夹下。

（4）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现一个 decoder 和 encoder 模型，单击该模型并在 codertest 上添加。点击工具栏中的 图标，连接引脚，如图 2-3-9 所示。

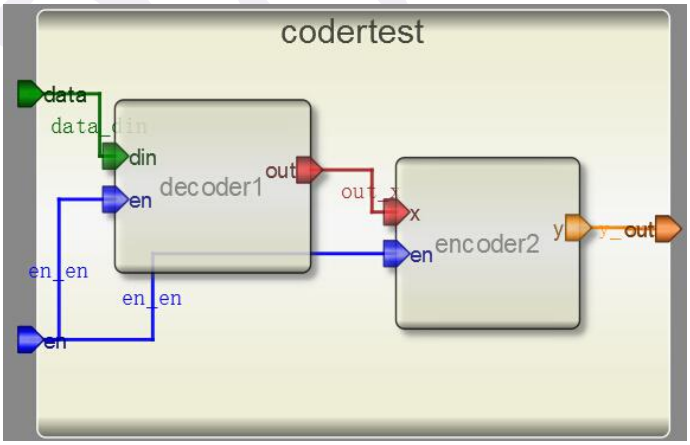


图 2-3-9 添加模型

（5）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

```
initial begin
    data=3'b000;
    en=0;
    #1 data=3'b000;
    en=1;
```

 Copyright © Robei

```
        #1 data=3'b001;
        #1 data=3'b010;
        #1 data=3'b011;
        #1 data=3'b100;
        #1 data=3'b101;
        #1 data=3'b110;
        #1 data=3'b111;
        #1 data=3'b000;
        #1 data=3'b001;
        #1 $finish;
    end
```

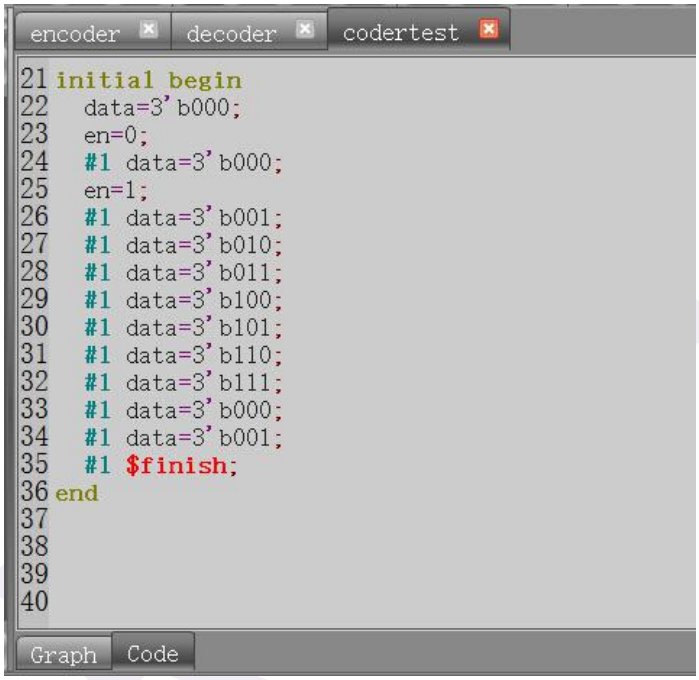


图 2-3-10 激励代码

（6）执行仿真并查看波形。点击工具栏 ，查看输出信息，检查没有错误之后点击 进行仿真，再点击 。在弹出的 Waveform 窗口上点击右侧 Workspace 中的信号，进行添加并查看，如图 2-3-11 所示。点击波形查看器工具栏上的 图标进行自动缩放。分析仿真结果并对照真值表，查看设计波形输入输出是否一致。

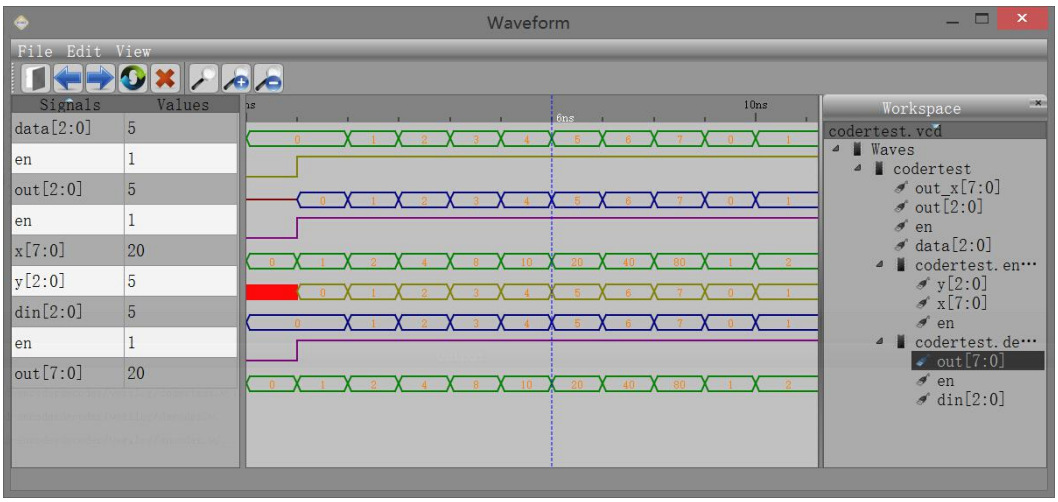


图 2-3-11 查看波形

2.3.3. 问题与思考

（1）使用 Robei 设计一个 BCD 编码器并进行仿真测试。

（2）使用 Robei 设计一个 16-4 线编码器和 4-16 线译码器，并仿真测试。

（3）使用 Robei 设计一个格雷编码器并测试结果。

 Copyright © Robei

2.4 实例四 ALU 设计

2.4.1. 本章导读

ALU（算数逻辑单元）是 CPU 的基本组成部分。设计要求掌握算术逻辑运算加、减操作原理，验证运算器的组合功能。

设计原理

ALU 的基本结构如图 2-4-1 所示。我们所设计的 ALU 要实现最基本的加减运算，与或非和异或等功能。

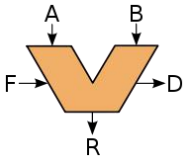


图 2-4-1 ALU 基本结构

（1）加法运算包含 2 种类型，一种是不带进位的加法器，另外一种是带进位的加法器。不带进位的加法器的公式：

$$\{D,R\}=A+B \quad (1)$$

带进位的可以进行加法器级联，实现更高位数的串行加法运算。带进位的加法器的公式：

$$\{D,R\}=A+B+F \quad (2)$$

（2）减法运算也包含 2 种类型。不带借位的减法运算：

$$\{D,R\}=A-B \quad (3)$$

带借位的减法运算：

$$\{D,R\}=A-B-F \quad (4)$$

设计要求

设计一个 8 位 ALU，并能实现数据与，或，非，异或，不带进位加法，带进位加法，不带借位减法和带借位减法运算。运算符采用 3 比特表示。A，B，R 均为 8 比特数据。用测试文件测试你的 ALU 功能，并用级联方式将 4 个 8 比特的 ALU 实现 32 比特的 ALU。

2.4.2. 设计流程

1. ALU 模型设计

（1）新建一个模型命名为 alu，类型为 module，同时具备 4 输入 2 输出。每个引脚的属性和名称参照图 2-4-2 进行对应的修改。

Name	Inout	DataType	Datasize	Function
A	input	wire	8	first input
B	input	wire	8	second input
op	input	wire	4	operation
F	input	wire	1	carry in
R	output	reg	8	result
D	output	reg	1	carry out

图 2-4-2 引脚属性

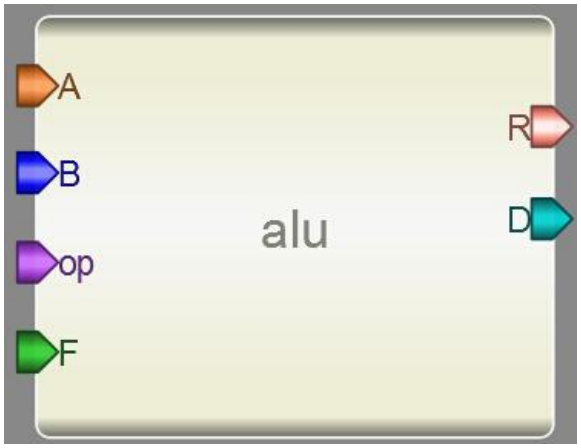


图 2-4-3 ALU 界面图

（2）添加代码。点击模型下方的 Code（如图 2-4-4 所示）添加代码。

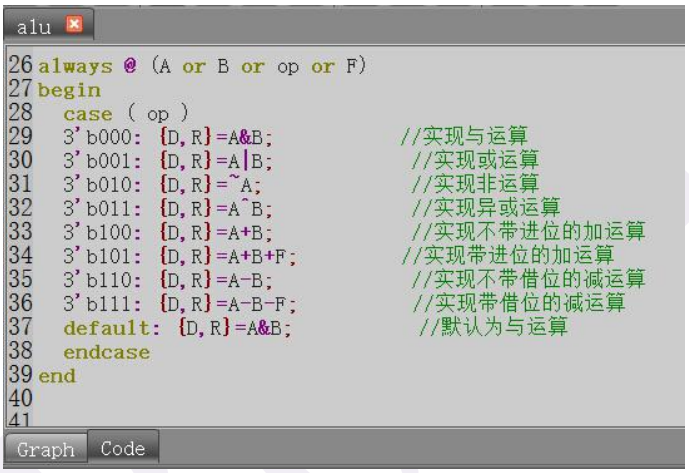


图 2-4-4 点击 Code 输入算法

在代码设计区内输入以下 Verilog 代码：

```
always @ (A or B or op or F)
begin
    case ( op )
    3'b000: {D,R}=A&B;          //实现与运算
    3'b001: {D,R}=A|B;          //实现或运算
    3'b010: {D,R}=~A;           //实现非运算
    3'b011: {D,R}=A^B;          //实现异或运算
    3'b100: {D,R}=A+B;          //实现不带进位的加运算
    3'b101: {D,R}=A+B+F;        //实现带进位的加运算
    3'b110: {D,R}=A-B;          //实现不带借位的减运算
    3'b111: {D,R}=A-B-F;        //实现带借位的减运算
    default: {D,R}=A&B;         //默认为与运算
    endcase
end
```

 Copyright © Robei

（3）保存模型到一个文件夹中，运行并检查有无错误输出。

2. 测试文件设计

（1）新建一个 4 输入 2 输出的测试文件，记得将 Module Type 设置为“testbench”各个引脚配置如图 2-4-5 所示。

Name	Inout	DataType	Datasize	Function
a	input	reg	8	first input
b	input	reg	8	second input
op	input	reg	4	operation
cin	input	reg	1	carry in
result	output	wire	8	result
cout	output	wire	1	carry out

图 2-4-5 新建测试文件

（2）另存为测试文件。将测试文件保存到 alu 模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现一个 alu 模型，单击该模型并在 alutest 上添加，并连接引脚，如图 2-4-6 所示。

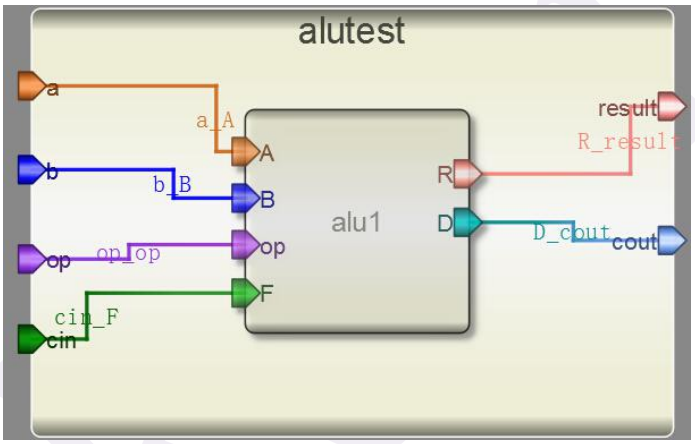


图 2-4-6 添加模型

（4）输入激励。点击测试模块下方的“Code”，输入激励算法，如图 2-4-7 所示。激励代码在结束的时候要用$finish 结束。

```
initial begin
    a=0;
    b=0;
    op=0;
    cin=0;
    #1
    a=3;
    b=1;
    op=0;
    #1
    a=2;
    b=1;
    op=1;
```

```
        #1
        a=255;
        b=0;
        op=2;
        #1
        a=5;
        b=6;
        op=3;
        #1
        a=128;
        b=128;
        op=4;
        #1
        a=4;
        b=5;
        cin=1;
        op=5;
        #1
        a=4;
        b=5;
        op=6;
        #1
        a=4;
        b=5;
        op=7;
        #1
        a=4;
        b=5;
        op=0;
        #1
        $finish;
    end
```

Robei

Copyright © Robei

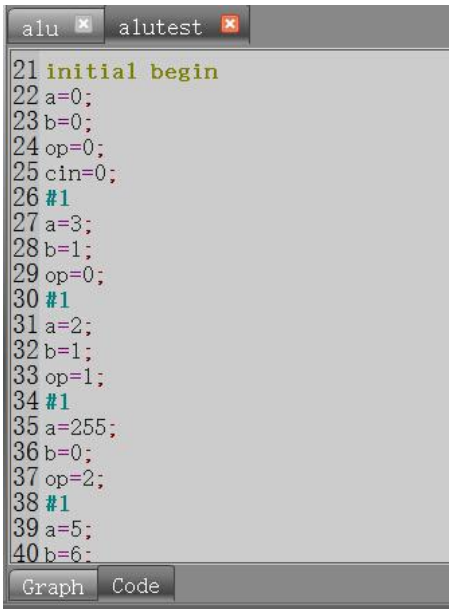


图 2-4-7 激励代码

（5）执行仿真并查看波形。查看输出信息。检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果，如图 2-4-8 所示。对照真值表，查看设计波形输入输出是否一致。

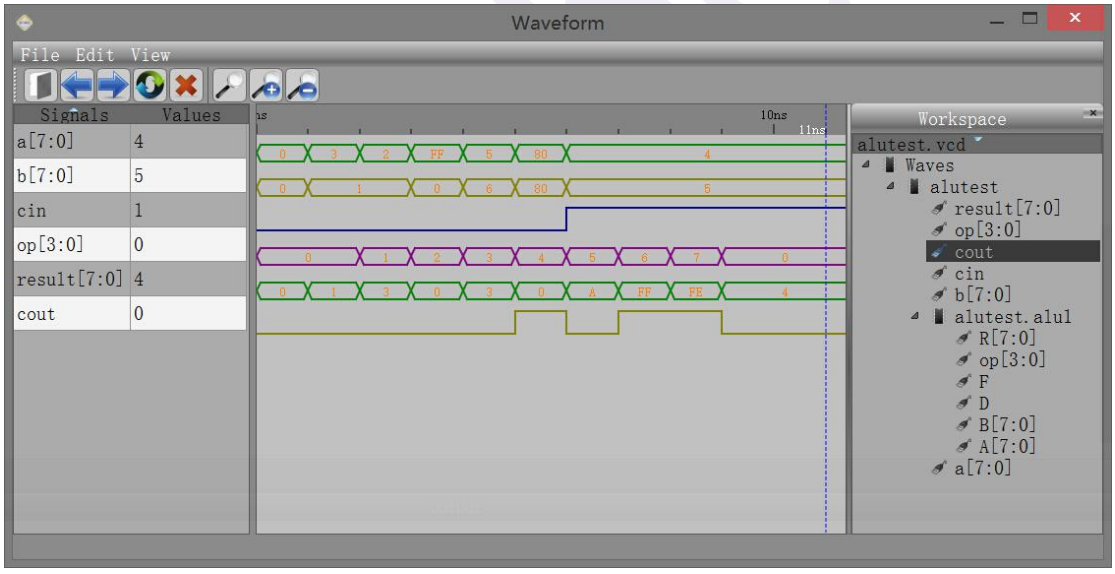


图 2-4-8 查看波形

3. 16 位 ALU 设计

（1）下面我们来设计一个 16 位的 ALU。这个设计中使用之前设计好的 8 位 ALU 模块，把输入的 16 位信号拆分为两个 8 位信号，经过 8 位 ALU 处理后再组合成最终的 16 位输出信号。

（2）split 模块设计：该模块的功能是把输入的 16 位数据分解为两个 8 位数据。模块引脚设计如图 2-4-9 所示。

Name	Inout	DataType	Datasize	Function
A	input	wire	16	Input Data
B	output	wire	8	Lower of Input
C	output	wire	8	Higher of Input

图 2-4-9 split 模块引脚

图 2-4-10 split 模块设计

模块设计好后，点击 code 标签，输入 split 模块的代码：

```
assign B=A[7:0];
assign C=A[15:8];
```

（3）merge 模块设计：该模块的功能是把两个 8 位数据组合为一个 16 位数据。模块引脚设计如图 2-4-11 所示。

Name	Inout	DataType	Datasize	Function
A	input	wire	7:0	
B	input	wire	7:0	
C	output	wire	15:0	

图 2-4-11 merge 模块引脚

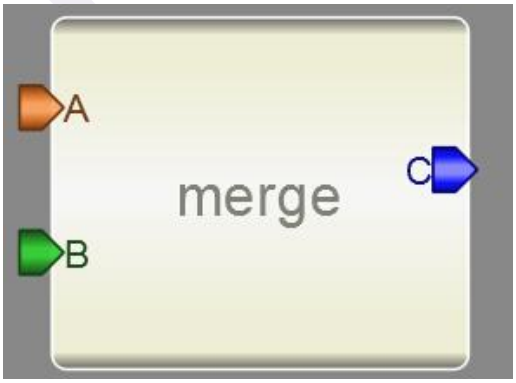


图 2-4-12 merge 模块设计

模块设计好后，点击 code 标签，输入 merge 模块的代码：

```
assign C={B, A};
```

（4）接下来建立一个模块，命名为 alu16，具有 4 输入和 2 输出，引脚设定如下图 2-4-13 所示：

Copyright © Robei

Ports:

Name	Inout	DataType	Datasize	Function
A	input	wire	16	
B	input	wire	16	
cin	input	wire	1	
op	input	wire	4	
result	output	wire	16	
cout	output	wire	1	

图 2-4-13 alu16 模块引脚设计

保存之后把之前设计好的 ALU，split 和 merge 模块添加进 alu16 模块，并进行连线。完成后的模块如图 2-4-14 所示：

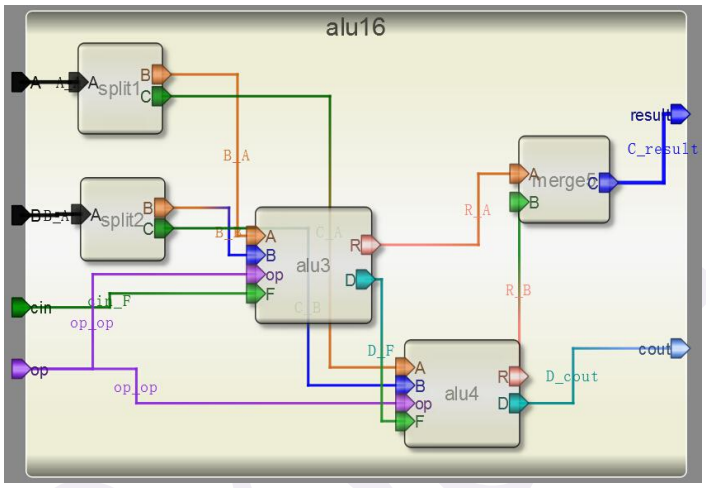


图 2-4-14 alu16 模块设计

（5）测试模块设计：新建一个模块，模块类型选择为 testbench，引脚设计如图 2-4-15 所示。

Name	Inout	DataType	Datasize
a	input	reg	16
b	input	reg	16
op	input	reg	4
cin	input	reg	1
result	output	wire	16
cout	output	wire	1

图 2-4-15 alu16 的测试模块引脚设计

保存后把之前设计的 alu16 模块添加进测试模块，并进行连线。连线后的测试模块如图 2-4-16 所示。

Copyright © Robei

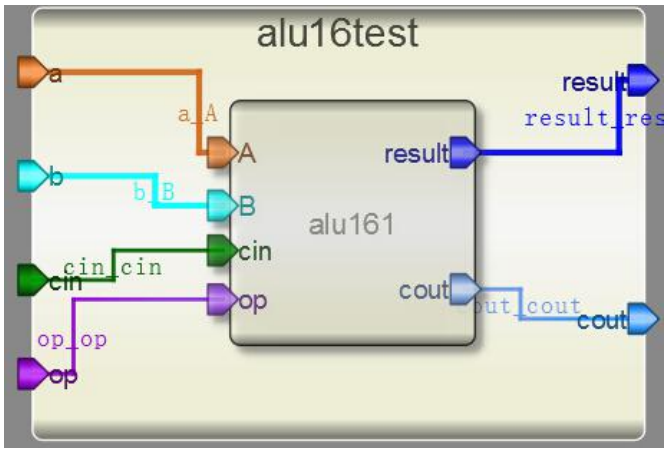


图 2-4-16 alu16 的测试模块设计

点击 code 标签，输入测试模块的激励代码：

```
    initial begin
        a=0;
        b=0;
        op=0;
        cin=0;
    #1
        a=24;
        b=35;
        op=0;
    #1
        a=56;
        b=18;
        op=1;
    #1
        a=96;
        b=80;
        op=2;
    #1
        a=51;
        b=26;
        op=3;
    #1
        a=128;
        b=128;
        op=4;
    #1
        a=64;
        b=15;
        cin=1;
        op=5;
```


Copyright © Robei

```
    #1
        a=74;
        b=35;
        op=6;
    #1
        a=24;
        b=75;
        op=7;
    #1
        a=24;
        b=55;
        op=0;
    #1
        $finish;
    end
```

（6）运行后，点击 Wave 查看波形，如图 2-4-17 所示，检查设计的正确性。

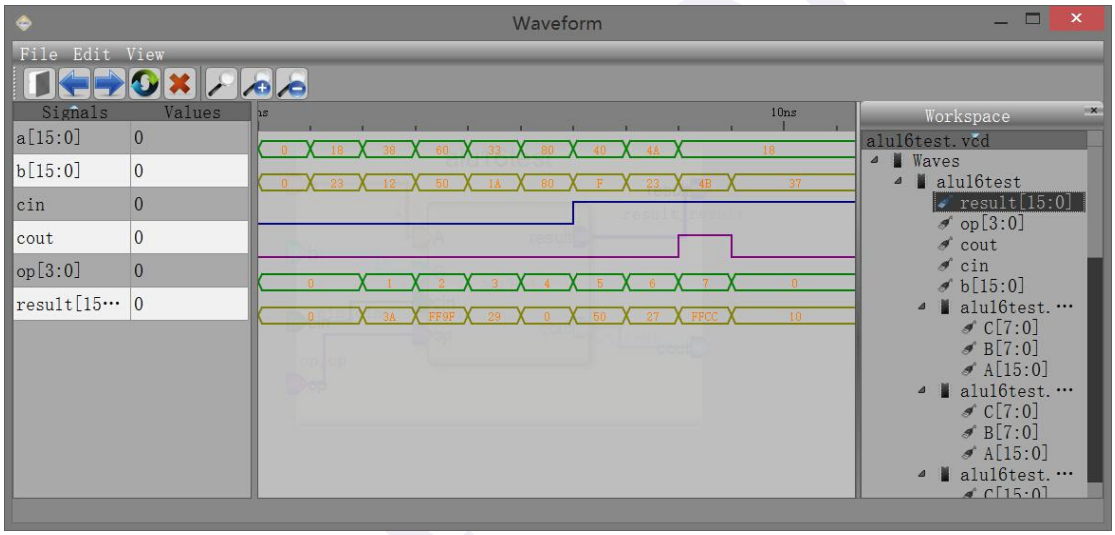


图 2-4-17 alu16 的测试模块仿真波形

4. 32 位 ALU 设计

（1）我们利用 8 位的 ALU 级联来设计一个 32 位的 ALU，这个设计需要先行注册 Robei 软件，否则不能进行仿真。

（2）创建一个新的模型，添加 10 个输入引脚，5 个输出引脚，各个引脚的配置如图 2-4-18 所示。保存到 alu 模型所在的文件夹。

Name	Inout	DataType	Datasize	Function
A0	input	wire	7:0	Bit 0-7 of A
B0	input	wire	7:0	Bit 0-7 of B
A1	input	wire	7:0	Bit 8-15 of A
B1	input	wire	7:0	Bit 8-15 of B
A2	input	wire	7:0	Bit 16-23 of A
B2	input	wire	7:0	Bit 16-23 of B
A3	input	wire	7:0	Bit 24-31 of A
B3	input	wire	7:0	Bit 24-31 of B
op	input	wire	4	operation
R0	output	reg	7:0	Bit 0-7 of R
R1	output	reg	7:0	Bit 8-15 of R
R2	output	reg	7:0	Bit 16-23 of R
R3	output	reg	7:0	Bit 24-31 of R
D	output	reg	1	carry out
F	input	wire	1	carry in

图 2-4-18 32 位 ALU 引脚

（3）添加 4 个 ALU 连接引脚。如图 2-4-19 所示。4 个 8 位的 ALU 进行级联，第一个输出的 D 连到下一级的 F，最终的 ALU 的 D 连接到顶层的 D 引脚。第一个 ALU 的 F 连接到顶层模块的 F。op 都连接到顶层的 op 引脚上，A，B 和 R 按照高低位进行连接。这样输入 $A_3A_2A_1A_0$，$B_3B_2B_1B_0$ 和 $R_3R_2R_1R_0$ 分别是 32 位 ALU 的输入和输出端。如图 2-4-19 所示。

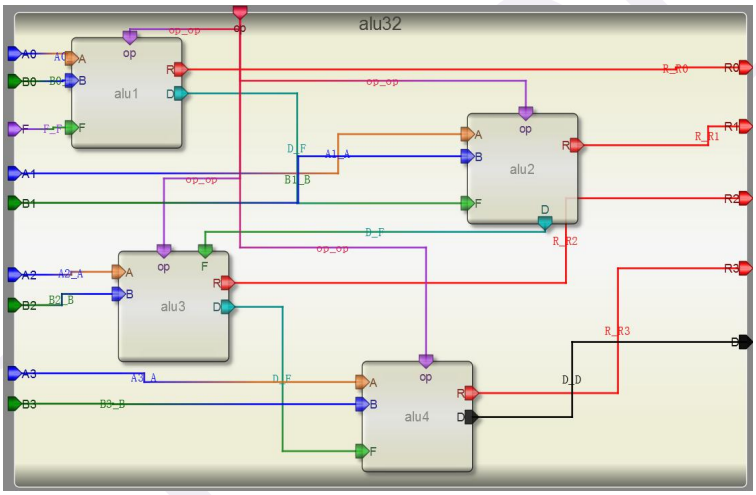


图 2-4-19 32 位 ALU 设计图

（4）创建一个测试文件，10 个输入引脚 5 个输出，按照图 2-4-20 进行引脚配置并保存到与 alu32bit 模型同一个文件下。

Name	Inout	DataType	Datasize	Function
A0	input	reg	7:0	Bit 0-7 of A
B0	input	reg	7:0	Bit 0-7 of B
A1	input	reg	7:0	Bit 8-15 of A
B1	input	reg	7:0	Bit 8-15 of B
A2	input	reg	7:0	Bit 16-23 of A
B2	input	reg	7:0	Bit 16-23 of B
A3	input	reg	7:0	Bit 24-31 of A
B3	input	reg	7:0	Bit 24-31 of B
op	input	reg	4	operation
R0	output	wire	7:0	Bit 0-7 of R
R1	output	wire	7:0	Bit 8-15 of R
R2	output	wire	7:0	Bit 16-23 of R
R3	output	wire	7:0	Bit 24-31 of R
D	output	wire	1	carry out
F	input	reg	1	carry in

图 2-4-20 32 位 ALU 测试文件引脚配置

Copyright © Robei

（5）从 Toolbox 里面的 Current 栏找 alu32bit 模型，并添加到测试模块上。对应引脚相连。如图 2-4-21 所示。

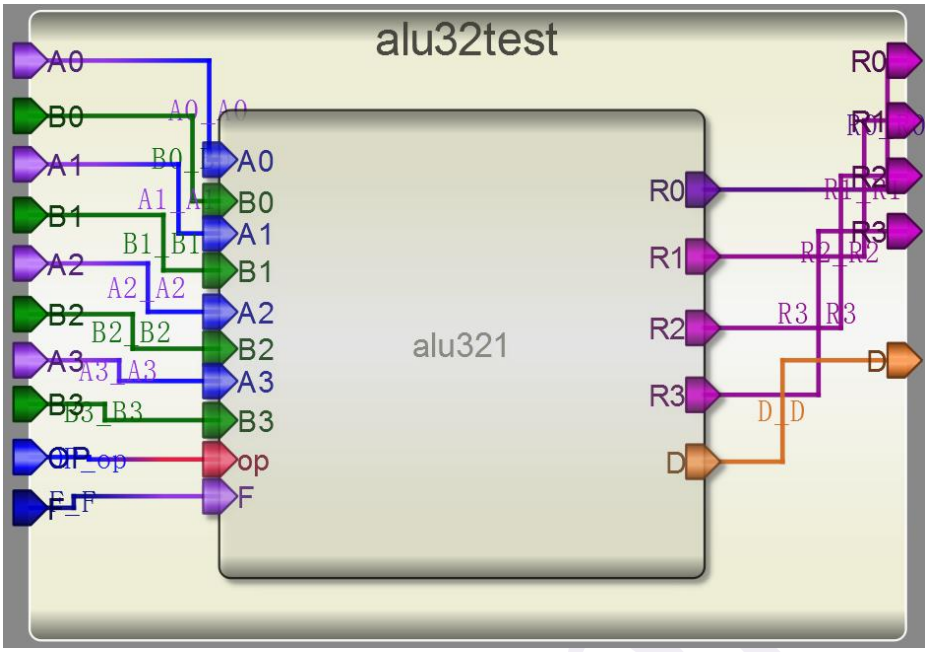


图 2-4-21 32 位 ALU 测试引脚连接

（6）自己设计测试激励代码，并仿真查看结果。

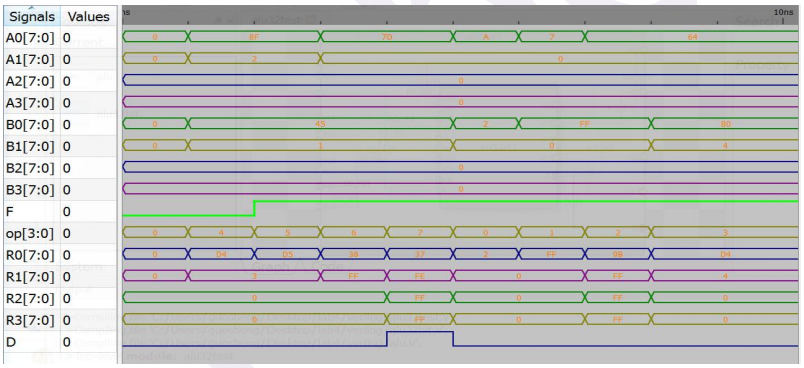


图 2-4-22 32 位 ALU 仿真波形

2.4.3. 问题与思考

1、不要使用 8 位 ALU 级联的方式，尝试直接用 Verilog 在 Robei 中实现一个 32 位或者 64 位 ALU。

2、挑战题：在 8 位 ALU 设计上添加乘法功能，输出结果变成 16 位输出。利用这个 ALU 实现一个 16 位的乘法器。提示：16 位乘法器分成低 8 位和高 8 位。如 A[15:0]拆分成 A[15:8]和 A[7:0]，同样拆分 B。之后用 4 个乘法器分别实现：

- A[7:0]×B[7:0]
- A[7:0]×B[15:8]
- A[15:8]×B[7:0]
- A[15:8]×B[15:8]

然后进行适当移位，再用加法器实现相加。

第三章：动手实战，板上点灯

今天我们进行的设计，不仅仅是要在 Robei 软件中完成前端的功能仿真，还要结合 Xilinx 公司的设计软件 Vivado，把我们的设计进行综合，下载到开发板上验证功能是否正确。读者们可能需要一些时间安装 Vivado 软件，读者需要自己去申请 Vivado 的免费 License。在学习数字电路的过程中，很多设计只要使用 Robei 进行功能仿真就可以完成，综合实现这一步可能并不是必要的，但是在整个 FPGA 设计流程中，使用 Vivado 这样的后端软件的步骤是必不可少的。通过今天的学习，读者可以直接体验从设计到板级应用的效果，把理论应用到实践当中来。

Robei

 Copyright © Robei

3.1 实例五 Robei 和 Vivado 的联合设计——流水灯设计

3.1.1. 本章导读

该设计将指导你在 Robei 中完成一个简单的 Verilog 设计并且通过波形仿真来验证你的设计的功能正确性，随后使用 Vivado IDE 综合和实现并生成比特流文件，最后，将生成的比特流下载到 Zybo 开发板，用实际电路验证设计是否正确。

设计目的

完成这个设计后，你将能够：

- 使用 Robei、Vivado 联合设计项目，使用 Zybo 开发板对项目进行验证，建立软硬件协同的设计平台；
- 使用 Robei 仿真模拟设计；
- 使用提供的 Xilinx 设计约束（XDC）文件来约束引脚位置；
- 合成并实现设计；
- 生成比特流；
- 使用生成的比特流配置 FPGA 和验证功能。

该设计通过四个开关对四个 LED 灯进行控制。具体的开关与 LED 灯的电路逻辑关系如图 3-1-1 所示。

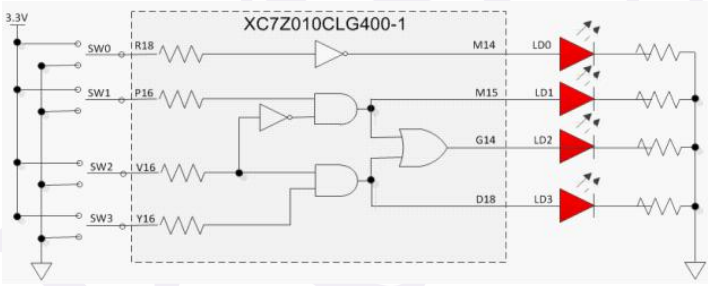


图 3-1-1 电路逻辑

3.1.2. Robei 设计内容

1. light 模型设计

（1）新建一个模型命名为 light，类型为 module，同时具备 1 输入 1 输出。每个引脚的属性和名称参照下图 3-1-2 进行对应的修改。

Name	Inout	DataType	Datasize
swt	input	wire	4
led	output	wire	4

图 3-1-2 light 模块引脚的属性

图 3-1-3 light 模块界面图

（2）添加代码。点击模型下方的 Code 添加代码。

代码如下：

```
assign led[0] = ~swt[0];
assign led[1] = swt[1] & ~swt[2];
assign led[3] = swt[2] & swt[3];
assign led[2] = (swt[1] & ~swt[2]) | (swt[2] & swt[3]);
```

（3）保存并执行后，如果没有错误，单击 view->code view 可查看完整的代码。如图 3-1-4：

```
1
2  module lab1(
3    swt,
4    led);
5
6    //---Ports declearation---
7    input [3:0] swt;
8    output [3:0] led;
9
10   wire [3:0] swt;
11   wire [3:0] led;
12
13   //----Code starts here-----
14   assign led[0] = ~swt[0];
15   assign led[1] = swt[1] & ~swt[2];
16   assign led[3] = swt[2] & swt[3];
17   assign led[2] = (swt[1] & ~swt[2]) | (swt[2] & swt[3]);
18
19
20
21
22 endmodule    //lab1
23
```

图 3-1-4 code view 查看完整的设计代码

在 Verilog 代码第 2 行定义了该模块的开始（标有关键字 module），第 22 行定义了该模块（标有关键字 endmodule）的结束。7-8 行定义了输入和输出端口，10-11 行定义了数据类型，而行 13-17 定义了模块的实际功能。在这些代码中，只有 13-17 行模块的实际功能需要手动输入，其余代码 Robei 软件都可以直接自动生成。

 Copyright © Robei

（4）保存模型到一个指定的文件夹（文件夹路径不能有空格和中文）中，运行并检查有无错误输出。

2. light_tb 测试文件的设计

（1）新建一个 1 输入 1 输出的 light_tb 测试文件，记得将 Module Type 设置为 “testbench”，各个引脚配置如图 3-1-5 所示。

Name	Inout	DataType	Datasize
switches	input	reg	4
leds	output	wire	4

图 3-1-5 light_tb 测试文件引脚的属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现当前目录下的所有模型，单击该模型并在 light_tb 上添加，并连接对应的引脚，如图 3-1-6 所示。

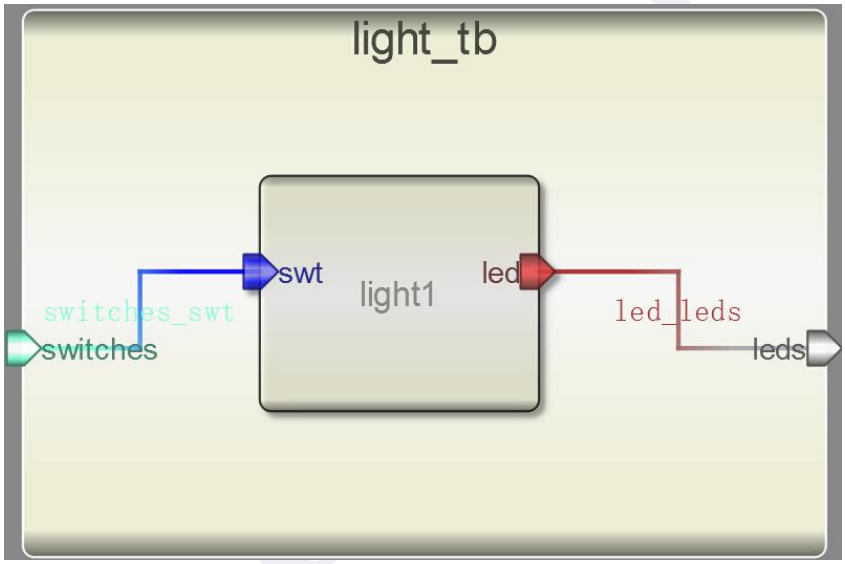


图 3-1-6 light_tb 的界面

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

激励代码：

```
initial
begin
    #5 switches=4'b1111;
    #5 switches=4'b1110;
    #5 switches=4'b1101;
    #5 switches=4'b1010;
    #5 switches=4'b1011;
    #5 switches=4'b0110;
    #5 switches=4'b0101;
```

```
    #5 switches=4'b0110;
    #10 $finish;
end
```

（5）执行仿真并查看波形。查看输出信息。检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 3-1-7：

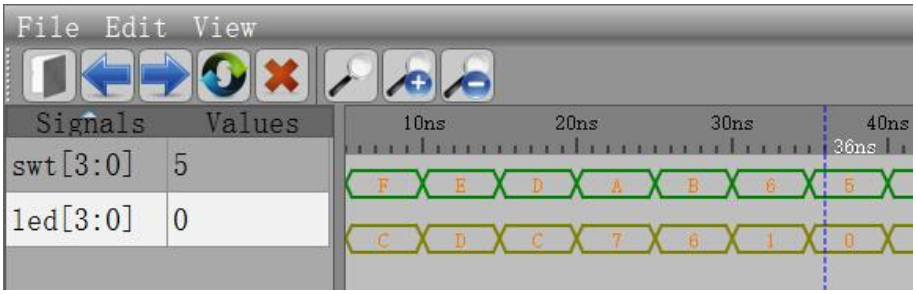


图 3-1-7 波形图

3. light_constrain 约束文件的设计

需要进行综合并下载到开发板验证的设计都要用约束文件来分配 FPGA 位于开发板上的开关和指示灯的物理 IO 地址。换句话说，约束文件的作用就是将设计中的各个输入输出端口和开发板上的实际物理 IO 对应起来。在一个完整的 FPGA 设计验证流程中，约束文件是必不可少的。

Robei 软件可以帮助我们完成约束文件的设计。

（1）新建一个模块，模块类型选择 constrain。如图 3-1-8 所示。

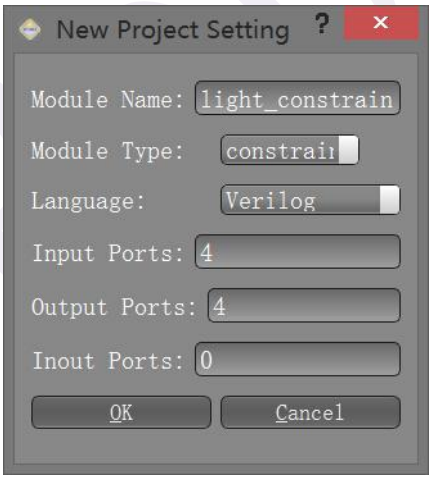


图 3-1-8 创建约束文件

虽然我们设计的流水灯模块只有一个输入 swt 和一个输出 led，但是这个输入和输出都是 4 位的。在约束文件中，一个输入或者输出只能对应 1 位，所以我们的约束文件需要 4 个输入和 4 个输出。

（2）把这个新建的模块和之前设计的模块保存在同一个目录下，接下来在左侧 Toolbox 中通过单击将 light 模块添加进约束模块。添加完成后，可以先将约束模块的 4 个输入和 light 模块的输入连接起来，将约束模块的 4 个输出和 light 模块的输出连接起来，如图 3-1-9 所示。

 Copyright © Robei

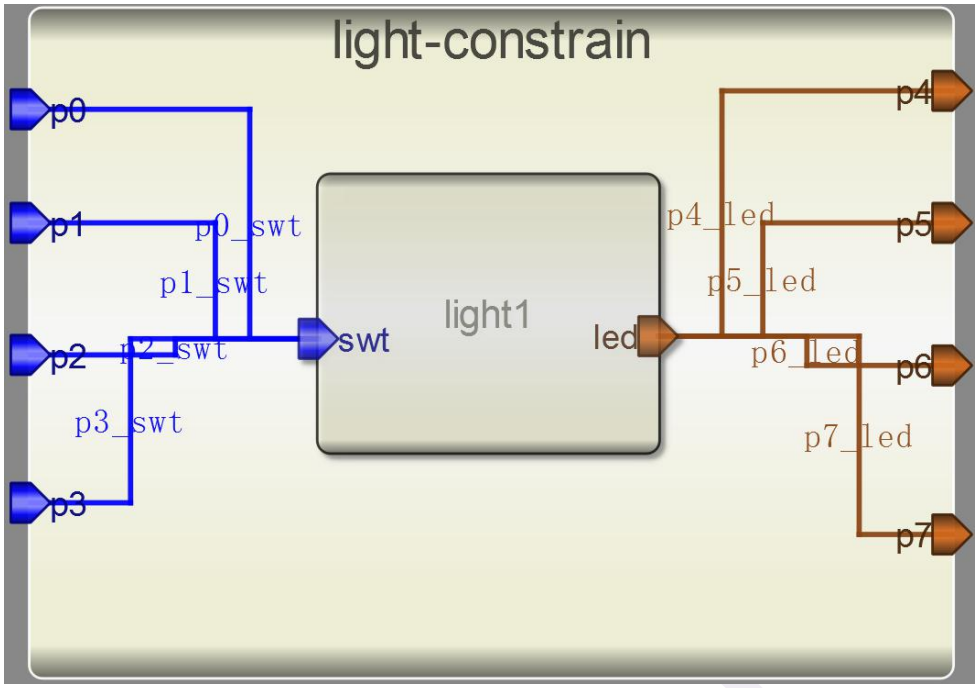


图 3-1-9 完成连线的约束模块

（3）修改端口和连线的名称。

约束文件端口的名称对应开发板上硬件 IO 的引脚名称。很多硬件引脚名称可以直接在开发板的 PCB 上读到，比如 4 个 LED 灯分别对应的是 M14,M15, G14,D18，4 个拨动开关分别对应的是 G15,P15,W13,T16 等。开发板的所有引脚 ID 可以在开发板相关文档资料里查到。

首先把连接输入 swt 的 4 个端口名称修改为 G15,P15,W13,T16，然后把连接输出 led 的 4 个端口名称修改为 M14,M15,G14,D18。

接下来，我们需要修改连线的名称。比如对于开关 G15，我们想让它对应我们设计中 swt 的最低位也就是 swt[0]，那么就要把端口 G15 和端口 swt 的连线名称修改为 0。类似地，连接 swt 的四条连线名称分别为 0,1,2,3，连接 led 的也是一样。修改后的模块如图 3-1-10。

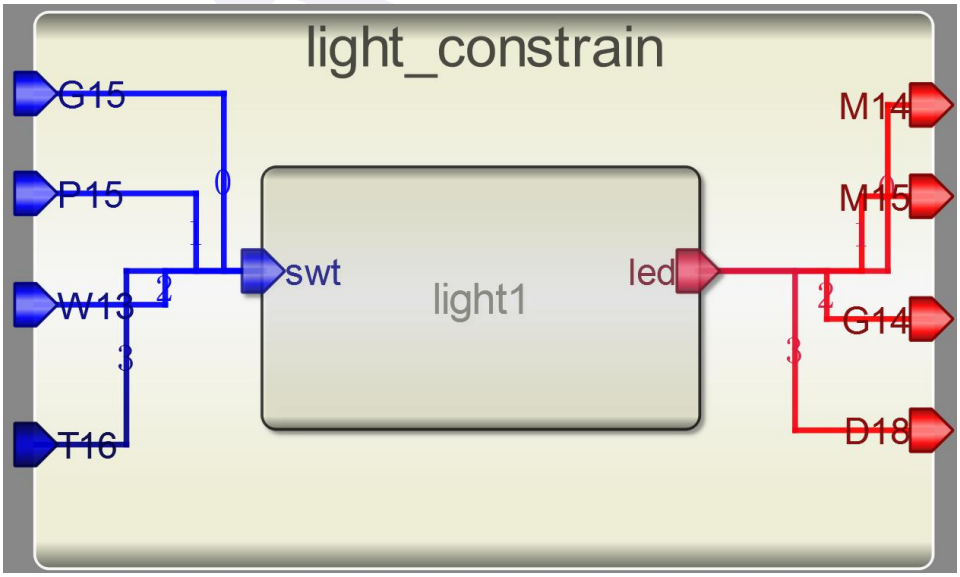


图 3-1-10 端口和连线名称修改后的模块图

（4）保存并运行。如果没有错误，Robei 软件就会在模块存储目录下创建一个名为 constrain 的文件夹，并在其中创建后缀名为 xdc 的约束文件。与查看设计时的代码一样，可以通过菜单栏的 View->CodeView 查看完整的约束代码，如图 3-1-11 所示。

Robei

```
1 #This file is generated by Robei!
2 #Pin Assignment for Xilinx FPGA with Software Vivado.
3 set_property PACKAGE_PIN G15 [get_ports swt[0]]
4 set_property IOSTANDARD LVCMOS33 [get_ports swt[0]]
5 set_property PACKAGE_PIN P15 [get_ports swt[1]]
6 set_property IOSTANDARD LVCMOS33 [get_ports swt[1]]
7 set_property PACKAGE_PIN W13 [get_ports swt[2]]
8 set_property IOSTANDARD LVCMOS33 [get_ports swt[2]]
9 set_property PACKAGE_PIN T16 [get_ports swt[3]]
10 set_property IOSTANDARD LVCMOS33 [get_ports swt[3]]
11 set_property PACKAGE_PIN M14 [get_ports led[0]]
12 set_property IOSTANDARD LVCMOS33 [get_ports led[0]]
13 set_property PACKAGE_PIN M15 [get_ports led[1]]
14 set_property IOSTANDARD LVCMOS33 [get_ports led[1]]
15 set_property PACKAGE_PIN G14 [get_ports led[2]]
16 set_property IOSTANDARD LVCMOS33 [get_ports led[2]]
17 set_property PACKAGE_PIN D18 [get_ports led[3]]
18 set_property IOSTANDARD LVCMOS33 [get_ports led[3]]
```

图 3-1-11 完整的约束文件代码

3.1.3. Vivado 设计内容

1. 工程创建

启动 Vivado 并选择 XC7Z010CLG400-1，并使用 Verilog HDL 语言。使用我们通过 Robei 设计的项目 light.v 和已经生成好的 light.xdc 文件。

（1）打开 Vivado 选择开始>所有程序>Xilinx 设计工具> Vivado2013.4> Vivado2013.4；

（2）单击创建新项目 Create New Project 启动向导。你将看到创建一个新的 Vivado 项目对话框，单击 Next；

（3）单击 New Project 里的浏览按钮，选择希望新项目保存的位置，比如可以选择路径 c:\xup\fpga_flow\labs，然后单击 Select；

（4）在工程名中输入 lab1。确保创建项目子目录复选框被选中。单击 Next；

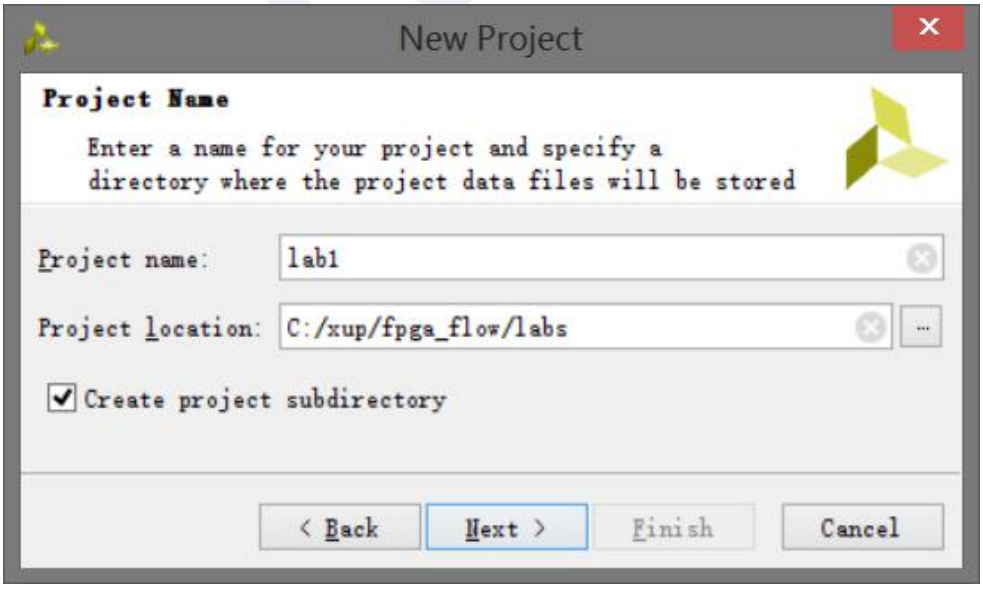


图 3-1-12 项目名称和项目位置

Copyright © Robei

（5）选择项目类型表单的 RTL Project 选项，不要在下面“Do not specify sources at this time”选项上打勾，以便我们可以接着添加源文件，然后单击 Next；

（6）使用下拉按钮，在添加源形式中选中 Verilog 作为目标语言和仿真语言；

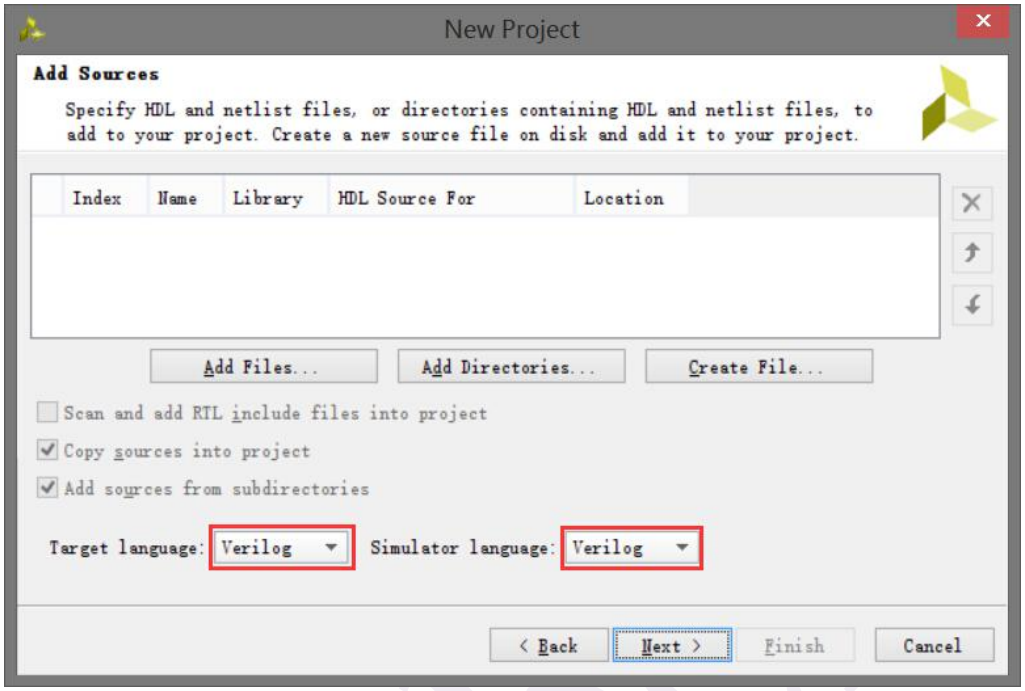


图 3-1-13 选择目标和仿真语言

（7）点击添加 Add Files…按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 light.v，单击 Open，然后单击 Next 去添加现有的 IP 形式；

（8）因为我们没有任何的 IP 添加，跳过这一步，直接单击 Next 去添加约束文件；

（9）点击添加 Add Files…按钮，浏览到之前设计存放的路径下的 constrain 文件夹，选择 light.xdc 并单击 OK，然后单击 Next；

Xilinx 的设计约束文件用于分配 FPGA 位于主板上的开关和指示灯的物理 IO 地址。这些信息可以通过主板的原理图或电路板的用户手册来获得。

（10）在默认窗口中，选择 Parts 选项，在 Filter 部分的各种下拉菜单中，选择 XC7Z010CLG400-1。过滤条件在图 3-1-14 中列出，分别是：在 Family 中选择 Zynq-7000，在 Sub-Family 中选择 Zynq-7000，在 Package 中选择 CLG400，如果需要的话，继续在 Speed grade 中选择-1。选择完毕后单击 Next；

Copyright © Robei

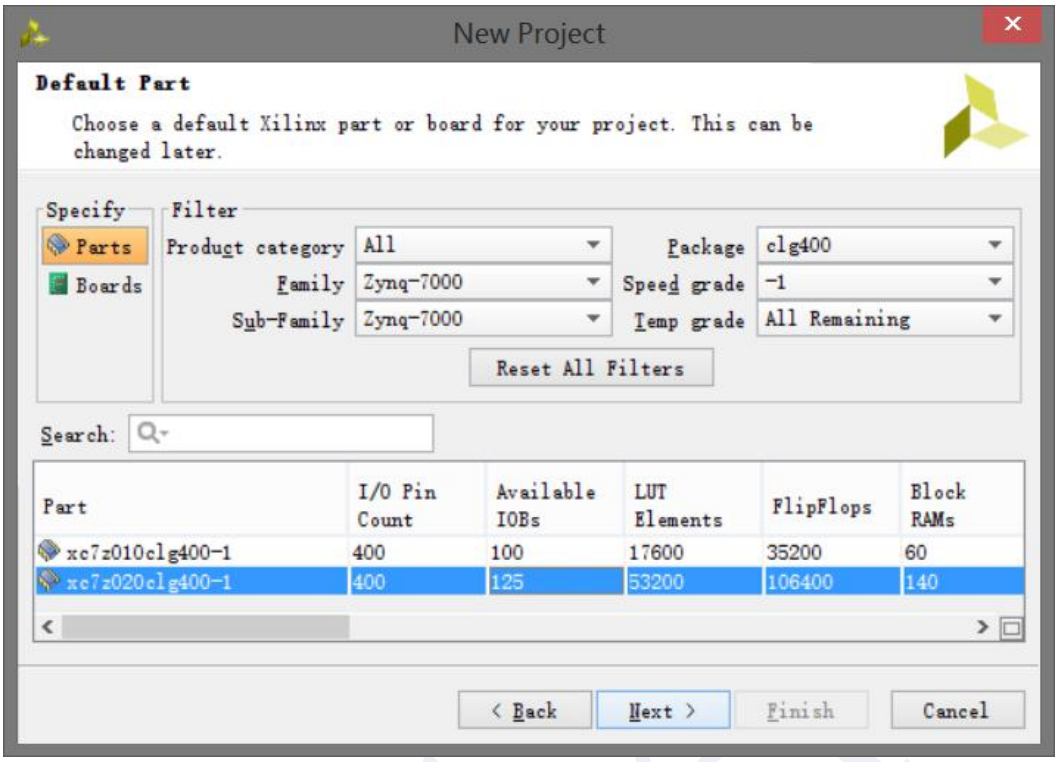


图 3-1-14 器件选型

（11）单击 Finish 创建 Vivado 项目。

使用 Windows 资源管理器，并查找到 c:\xup\fpga_flow\labs\lab1 目录。你会发现 lab1.data、lab1.srcs 目录和 lab1.xpr（Vivado）项目文件已创建。该 lab1.data 目录是为 Vivado 程序数据库所创建的。目录 constrs_1 和 sources_1 的 lab1.srcs 目录下被创建；其中 lab1.xdc（constraint）和 lab1.v（source）被分别放置。

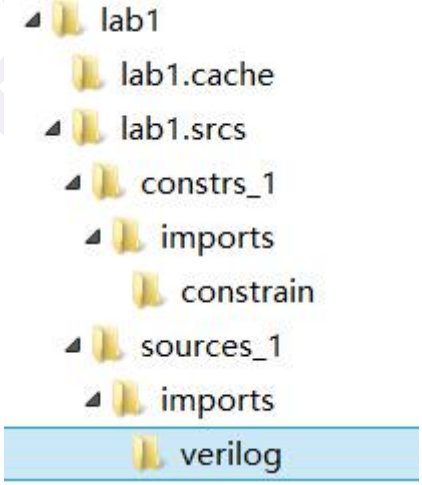


图 3-1-15 生成的目录结构

（12）打开 light.v 源代码并分析内容。在资源窗口中，双击打开 light.v 进入文本编辑模式。

Copyright © Robei

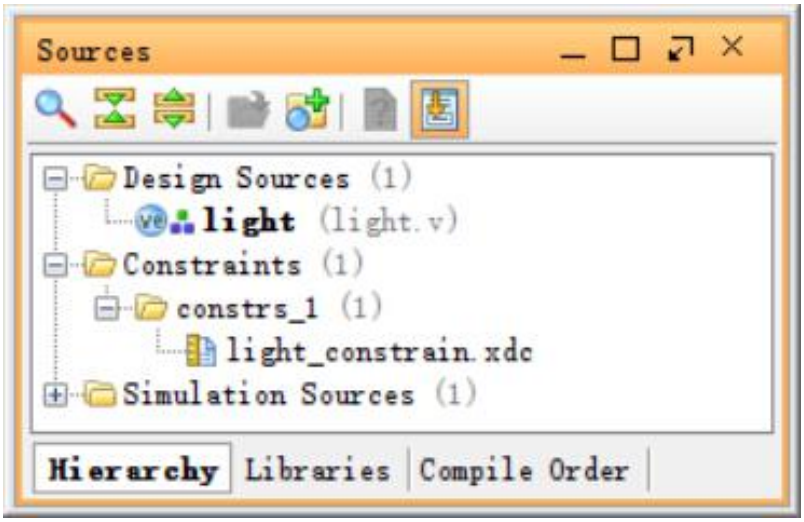


图 3-1-16 打开源文件

（13）打开 light.xdc 源代码并分析内容。在资源窗口中，展开约束文件夹，然后双击打开 light.xdc 进入文本编辑模式。约束代码如下：

```
set_property PACKAGE_PIN G15 [get_ports swt[0]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[0]]
set_property PACKAGE_PIN P15 [get_ports swt[1]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[1]]
set_property PACKAGE_PIN W13 [get_ports swt[2]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[2]]
set_property PACKAGE_PIN T16 [get_ports swt[3]]
set_property IOSTANDARD LVCMOS33 [get_ports swt[3]]

set_property PACKAGE_PIN M14 [get_ports led[0]]
set_property IOSTANDARD LVCMOS33 [get_ports led[0]]
set_property PACKAGE_PIN M15 [get_ports led[1]]
set_property IOSTANDARD LVCMOS33 [get_ports led[1]]
set_property PACKAGE_PIN G14 [get_ports led[2]]
set_property IOSTANDARD LVCMOS33 [get_ports led[2]]
set_property PACKAGE_PIN D18 [get_ports led[3]]
set_property IOSTANDARD LVCMOS33 [get_ports led[3]]
```

Robei

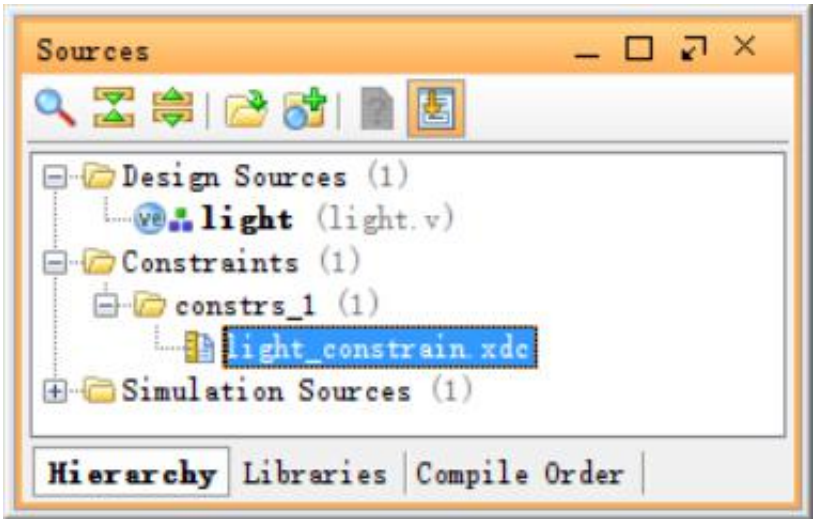


图 3-1-17 打开约束文件

（14）对 RTL 源文件进行分析。展开 the Open Elaborated Design 的下拉菜单中的 RTL 分析任务并且单击 Schematic 查看原理图。

该模型（设计）将制定并显示设计的原理图。

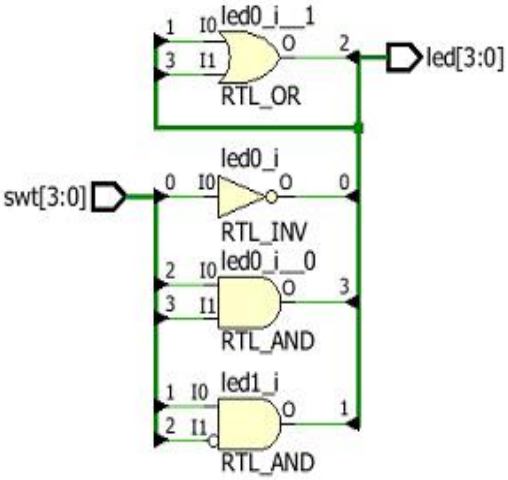


图 3-1-18 设计的逻辑视图

2. 使用 Vivado 综合工具来综合设计并且分析项目主要输出

（1）单击综合任务下拉菜单中的 Run Synthesis。综合过程将分析 light.v 文件并尝试生成门级网表文件。当这个过程完成了综合，将会弹出带有三个选项的完成对话框；

（2）选择 Open Synthesized Design 选项，然后单击 OK，因为我们想看看在功能实现之前综合的输出。如果有对话框弹出，单击是，关闭阐述设计；

（3）选择项目摘要选项卡 Project Summary，并了解各个窗口。如果你没有看到项目摘要选项卡，那么选择 Layout > Default Layout，或点击项目概要图标 。

Copyright © Robei

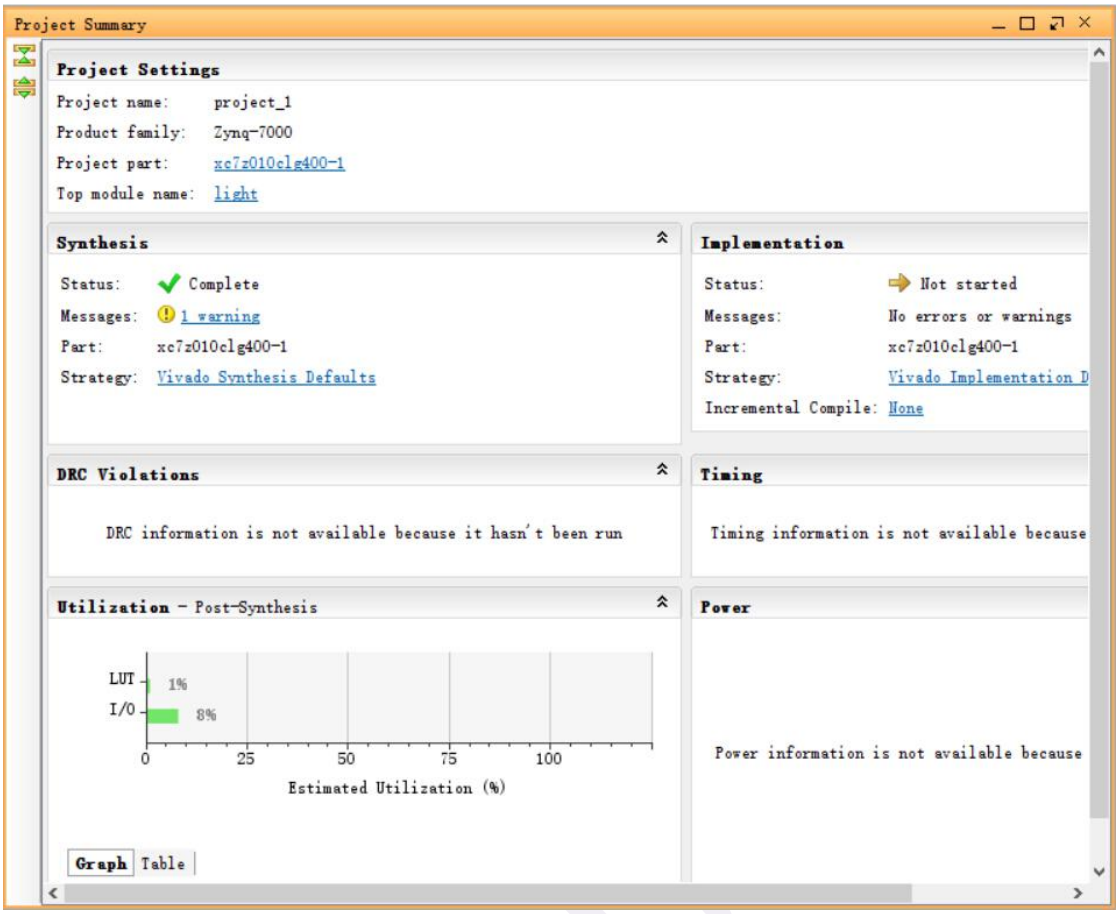


图 3-1-19 项目摘要视图

点击不同的链接，看看提供了什么允许你更改综合的设置信息。

（4）单击项目摘要选项卡中的 Table，如图 3-1-20 所示。请注意，估计有 3 个 LUT 和 8 个 IO（4 输入和 4 输出）被使用。

Resource	Utilization	Available	Utilization %
LUT	3	17600	0.02
I/O	8	100	8.00

图 3-1-20 资源利用率估算汇总

（5）在综合下拉菜单下，单击 Schematic，查看综合设计的示意图。

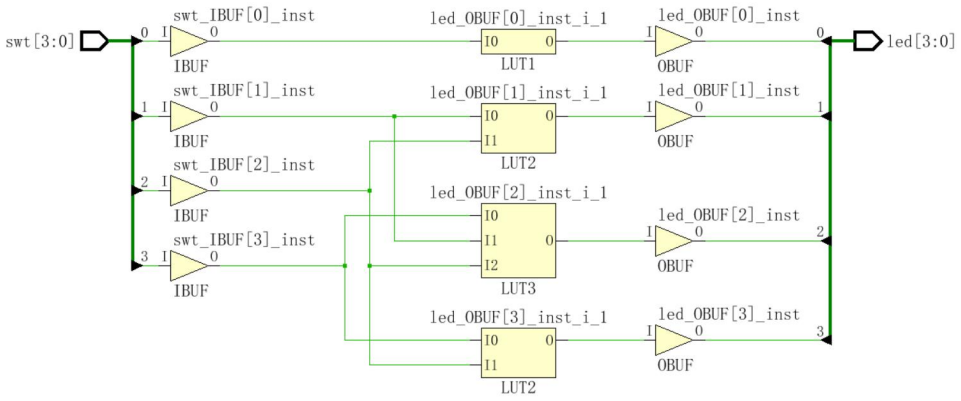


图 3-1-21 综合设计的示意图

注意到 IBUFs 和各 OBUF 被自动实例化（添加）到设计中对输入和输出进行缓存。逻辑门在 LUT（1 输入 LUT1，2 输入 LUT2，3 个输入 LUT3）中实现。四个门的 RTL 分析输出映射到四个 LUT 中的综合输出。

使用 Windows 资源管理器，验证 lab1.runs 文件夹是在 lab1 目录下。在 runs 目录下，synth_1 目录用来放置一些综合文件。

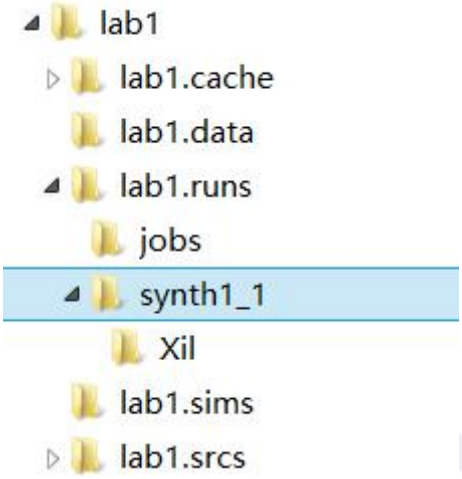


图 3-1-22 综合后的设计的目录结构

3. 使用 Vivado 实现设计的分析以及项目摘要输出

（1）单击在流程导航菜单中的实现任务 Run Implementation。实现过程将在综合设计上运行。当这个过程完成了将会弹出带有三个选项的实现完成对话框；

（2）选择 Open implemented design，单击确定，此时可查看在器件上实现设计的视图；

（3）出现提示，关闭综合设计，单击是。已实现的设计将被打开。单击确定查看器件视图；

（4）在网表窗口中，选择连线中的一个（例如 swt_IBUF[2]），并注意到在设备视图选项中即可显示连线在实际电路中的位置和走向；

（5）如果没有显示，单击连线资源图标 ，显示连线资源；

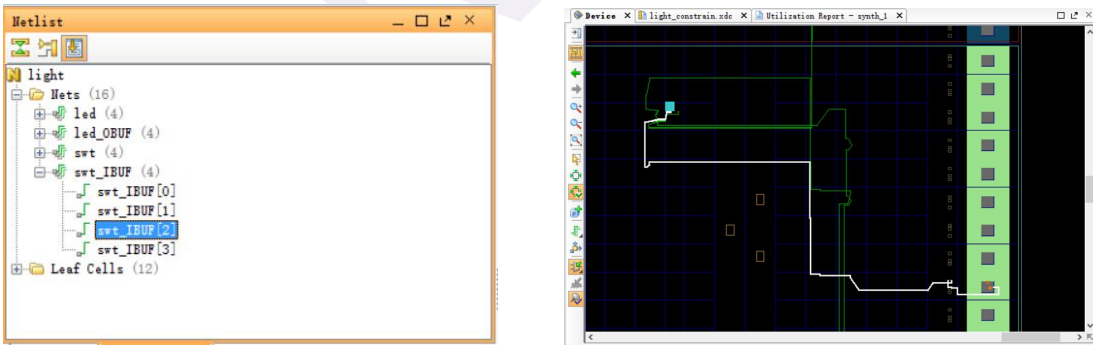


图 3-1-23 查看设计实现

（6）关闭实现设计的视图，然后选择 Project Summary 项目摘要选项卡（你可能需要更改为默认布局视图），并观察结果。选择 Post-Implementation。请注意，实际的资源利用率是 3 个 LUT 和 16 个的 IO。此外，它也表示在这个设计中没有定义时序约束（由于这个设计是组合逻辑电路）；

Copyright © Robei

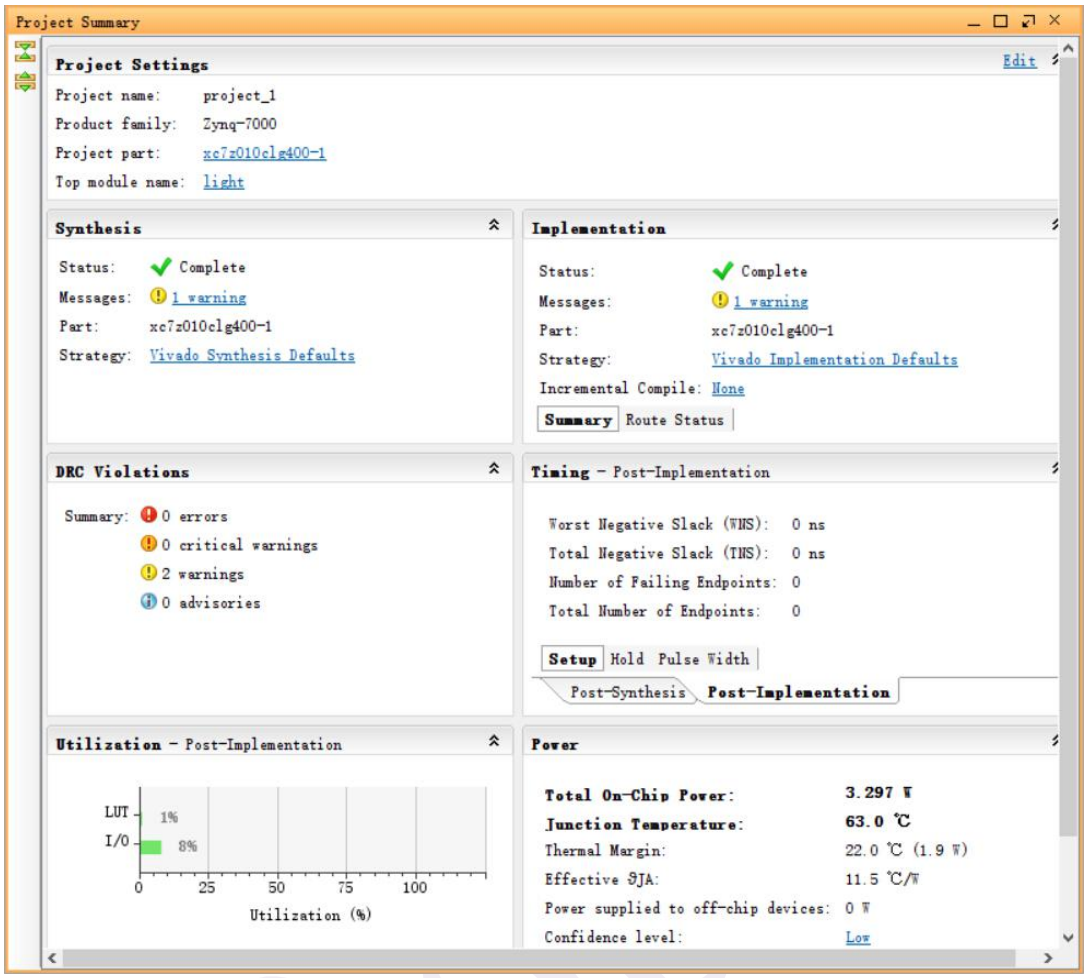


图 3-1-24 实现结果

使用 Windows 资源管理器，确认 synth_1 和 impl_1 被创建在 lab1_runs 的同一级目录下。impl_1 目录包含多个文件，包括实现情况的报告文件。

（7）在 Vivado 中，选择在底部面板中的 Reports 选项卡（如果不可见，单击窗口中的菜单栏，选择 Reports），然后双击 Place Design section 下的 Utilization Report 条目。如图 3-1-25 所示。该报告将在辅助视图窗口中显示资源利用率。注意，由于该设计是组合逻辑电路所以没有使用寄存器。

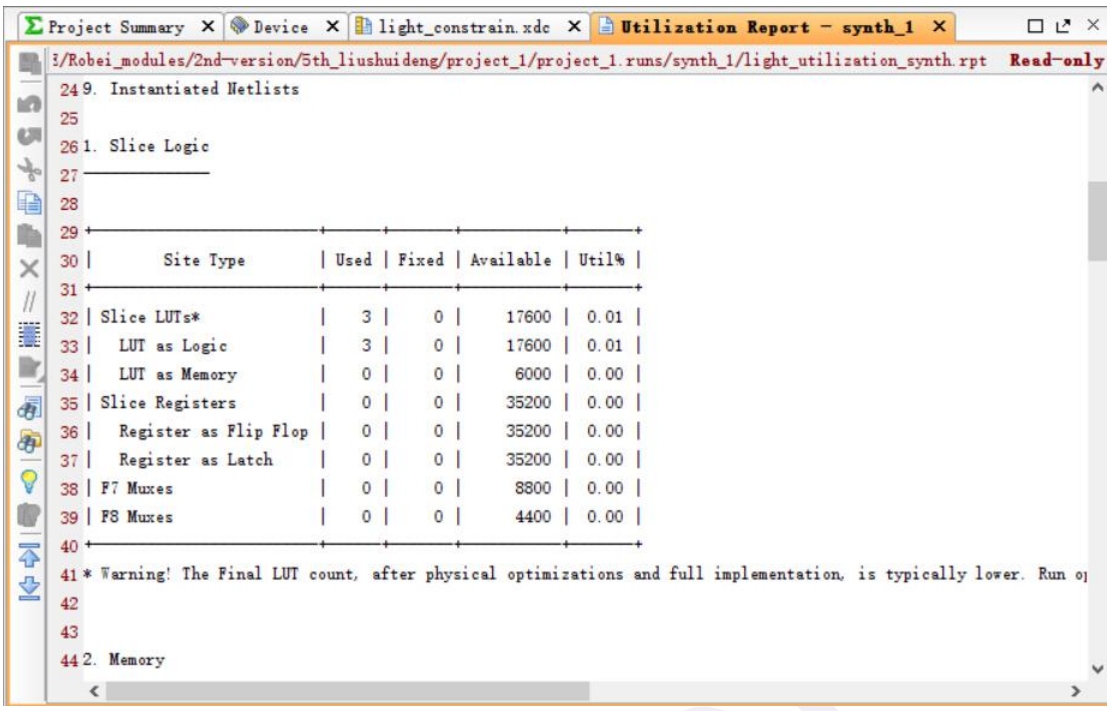


图 3-1-25 查看使用报告

4. 将设计在开发板上实现

将开发板上的电源开关拨到 ON。生成比特流，打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口（在电源开关的旁边）；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击程序和流程导航窗口中的调试任务下的 Generate Bitstream。比特流生成过程将在实现设计上运行。当这个过程完成比特流生成后，会弹出有三个选项的完成对话框，如图 3-1-27 所示：

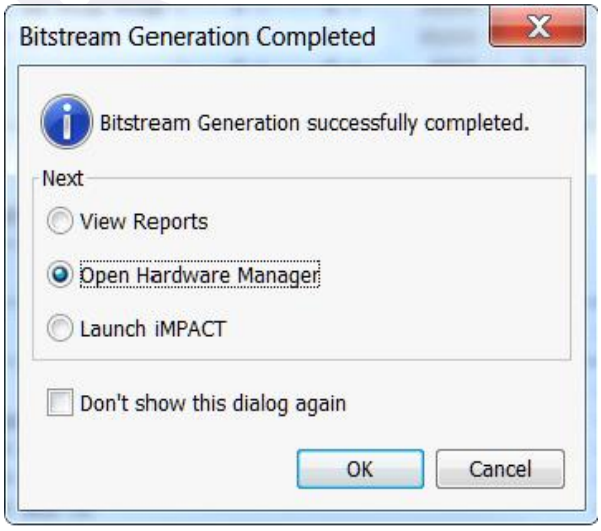


图 3-1-27 比特流生成

这一过程将已经生成的 lab1.bit 文件放在 lab1.runs 目录下的 impl_1 目录下。

（5）选择打开硬件管理器选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

 Copyright © Robei

（6）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target，如图 3-1-28 所示；

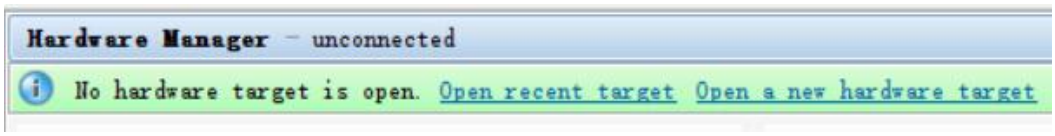


图 3-1-28 打开新的硬件目标

（7）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（8）单击 Next 以选择本地主机端口。应检测并确定为一个硬件目标的 JTAG 电缆，它使用 Xilinx_tcf。它也将显示检测到的硬件设备。

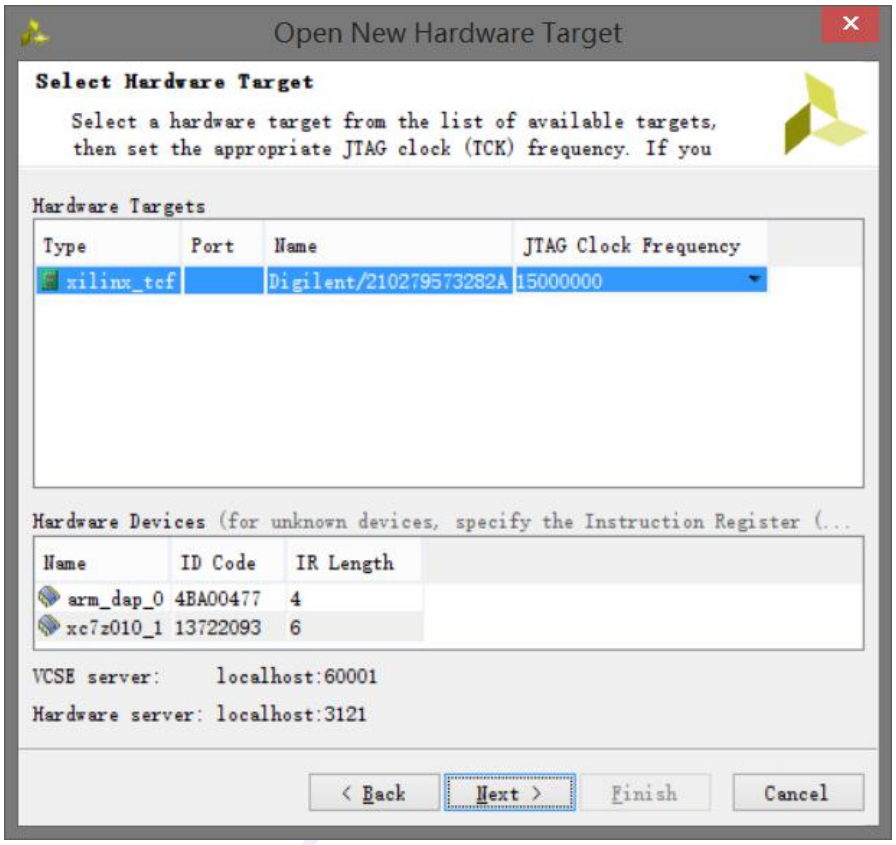


图 3-1-29 新的硬件指标的检测

（9）单击两次 Next，然后单击 Finish。未连接硬件会话状态更改为服务器名称并且器件被高亮显示，如图 3-1-30 所示。还要注意，该状态表明它还没有被编程；

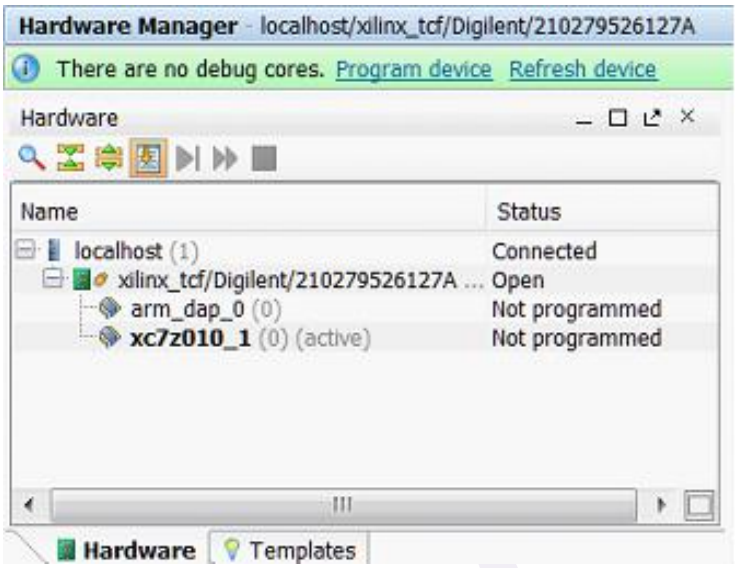


图 3-1-30 打开硬件会话

（10）选择器件，并验证 lab1.bit 被选为常规选项卡中的程序文件；

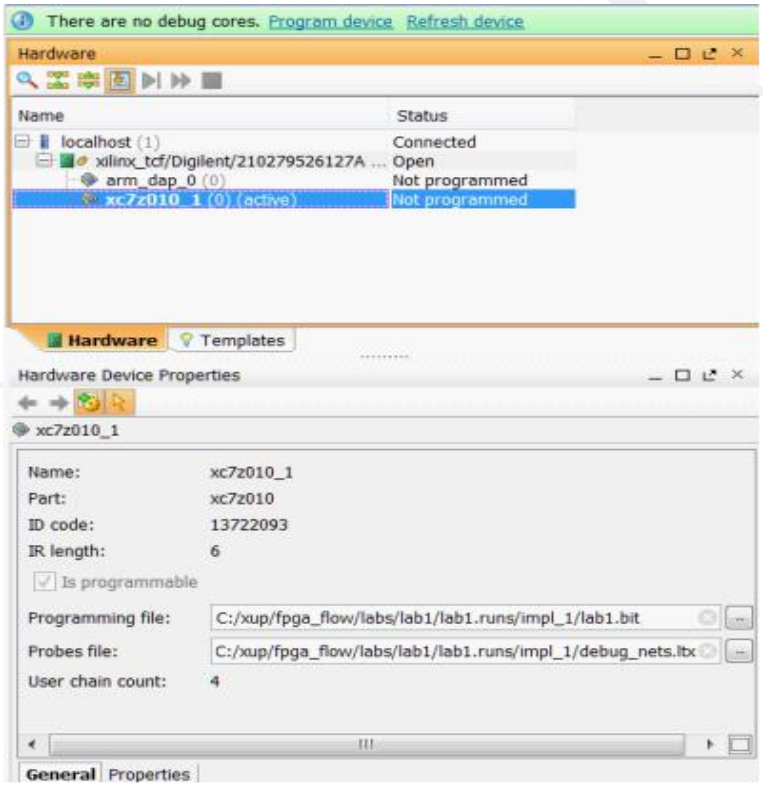


图 3-1-31 编程文件

（11）在器件上单击鼠标右键，选择程序器件，或单击 Program device > XC7z010_1 链接到目标 FPGA 器件进行编程；

Copyright © Robei

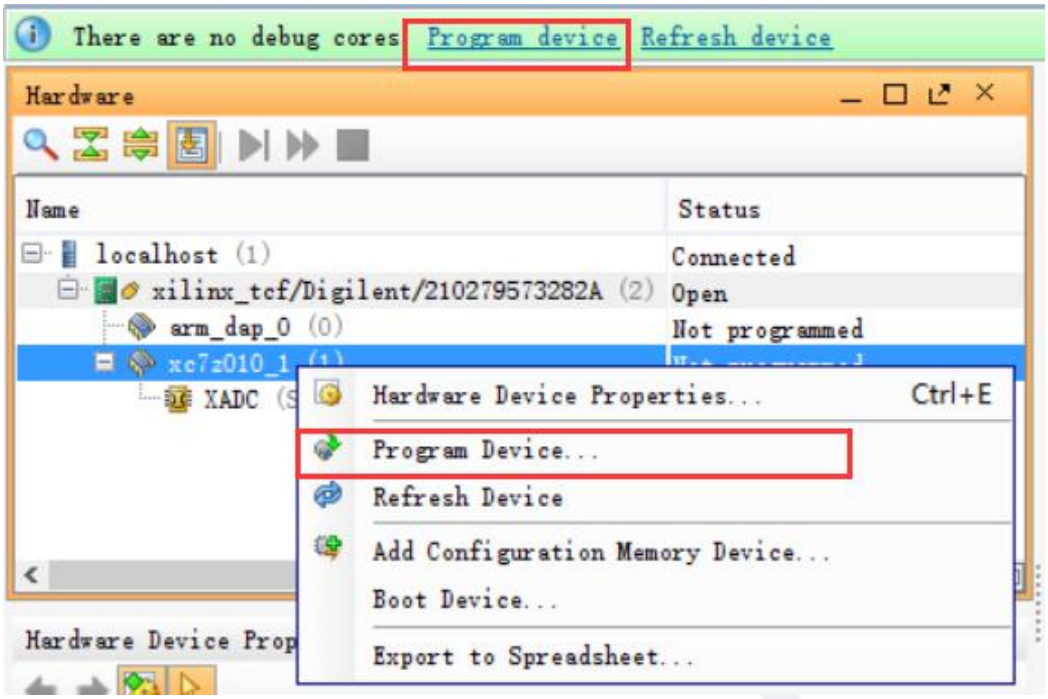


图 3-1-32 选择 FPGA 进行编程

（12）单击确定对 FPGA 进行编程。Done（完成）灯亮起时，说明对器件编程成功。

（13）通过开关的开闭来观察 LED（请参考前面的逻辑图）验证输出；

swt[0]为 0 时，led[0]亮起

swt[1]为 1，swt[2]为 0 时，led[1]亮起

swt[2]和 swt[3]同为 1 时 led[3]亮起

led[2]当 led[1]或 led[3]亮起时则亮，否则不亮

（14）如果验证正确，关闭开发板电源；

（15）选择 File > Close Hardware Manager 关闭器件的会话。

3.1.4. 总结

该项目介绍了使用 Robei 来简化设计的一种完整的设计流程，该项目通过 Robei 来进行对 Verilog 的设计之后使用 Vivado 以及 Zybo 开发板对设计进行综合，实现以及板级验证的详细过程。

3.2 实例六 自动售饮料机

3.2.1. 本章导读

了解自动售货机的工作流程以及各个工作状态，以及其 testbench，最后在 Robei 可视化仿真软件经行功能实现和仿真验证。

设计原理

自动售货机的信号定义：clk：时钟输入；reset：系统复位信号；half_dollar：代表投入 5 角硬币；one_dollar：代表投入 1 元硬币；half_out：表示找零信号；dispense：表示机器售出一瓶饮料。

当 reset=0 时，售货机处于工作状态，此时连续往售货机中投硬币（可以是 5 角也可以是一元），投入最后一枚硬币时，如果之前投入的硬币总和为 2.5 元，则可以取走一瓶饮料，如果少于 2.5 元则继续投币，如果为 3 元则显示可以取出一瓶饮料，而且找零显示信号为高电平。

投入硬币的总额	自动售饮料机给出的信号
<2.5 元	继续投币
=2.5 元	可以取出一瓶饮料
=3 元	可以取出一瓶饮料，并且找零

3.2.2. 设计流程

1. sell 模块的设计

（1）新建一个模型命名为 sell，类型为 module，同时具备 4 个输入和 2 个输出，每个引脚的属性和名称参照下图 3-2-1 经行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	clock
rst	input	wire	1	reset
one_dollar	input	wire	1	put $1
half_dollar	input	wire	1	put $0.5
half_out	output	reg	1	out $0.5
dispense	output	reg	1	sell

图 3-2-1 sell 引脚的属性

图 3-2-2 sell 界面图

 Copyright © Robei

（2）添加代码。点击模型下方的 Code 添加代码。

代码：

```
parameter idle=0,half=1,one=2,one_half=3,two=4;
reg[2:0] D;
always @(posedge clk)
begin
    if(rst)
    begin
        dispense=0;
        half_out=0;
        D=idle;
    end
    case(D)
        idle:
            if(half_dollar)
                D=half;
            else if(one_dollar)
                D=one;
        half:
            if(half_dollar)
                D=one;
            else if(one_dollar)
                D=one_half;
        one:
            if(half_dollar)
                D=one_half;
            else if(one_dollar)
                D=two;
        one_half:
            if(half_dollar)
                D=two;
            else if(one_dollar)
            begin
                dispense=1;
                D=idle;
            end
        two:
            if(half_dollar)
            begin
                dispense=1;
                D=idle;
            end
            else if(one_dollar)
            begin
```

Robei

```
                    dispense=1;
                    half_out=1;
                    D=idle;
                end
        endcase
    end
```

（3）保存模型到一个文件夹(文件夹路径不能有空格和中文)中，运行并检查有无错误。

2. sell_test 测试文件设计

（1）新建一个具有 4 个输入 2 个输出的 sell_test 测试文件，记得将 Module Type 设置为“testbench”，各个引脚配置如图 3-2-3 所示。

Name	Inout	DataType	Datasize
clk	input	reg	1
rst	input	reg	1
one_dollar	input	reg	1
half_dollar	input	reg	1
half_out	output	wire	1
dispense	output	wire	1

图 3-2-3 sell_test 引脚的属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里会出现模型，单击该模型并在 sell_test 上添加，并连接引脚，如下图 3-2-4 所示：

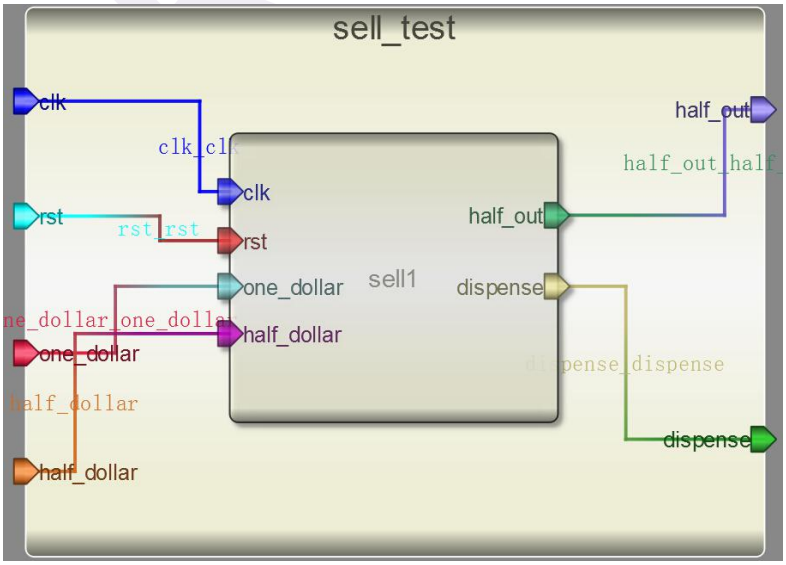


图 3-2-4 sell_test 模块界面

 Copyright © Robei

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

测试代码：

```
initial begin
    one_dollar=0;
    half_dollar=0;
    rst=1;
    clk=0;
    #100 rst=0;
    repeat(2)@(posedge clk);
    #2 one_dollar=1;
    repeat(1)@(posedge clk);
    #2 one_dollar=0;
    repeat(2)@(posedge clk);
    #2 one_dollar=1;
    repeat(1)@(posedge clk);
    #2 one_dollar=0;
    repeat(2)@(posedge clk);
    #2 one_dollar=1;
    repeat(1)@(posedge clk);
    #2 one_dollar=0;
    #20 rst=1;
    #100 rst=0;
    repeat(2)@(posedge clk);
    #2 one_dollar=1;
    repeat(1)@(posedge clk);
    #2 one_dollar=0;
    repeat(2)@(posedge clk);
    #2 one_dollar=1;
    repeat(1)@(posedge clk);
    #2 one_dollar=0;
    repeat(2)@(posedge clk);
    #2 half_dollar=1;
    repeat(1)@(posedge clk);
    #2 half_dollar=0;
    #20 rst=1;
    #5 $finish;
end
always #10 clk=~clk;
```

（5）执行仿真并查看波形。查看输出信息。

检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 3-2-5 所示：

Robei

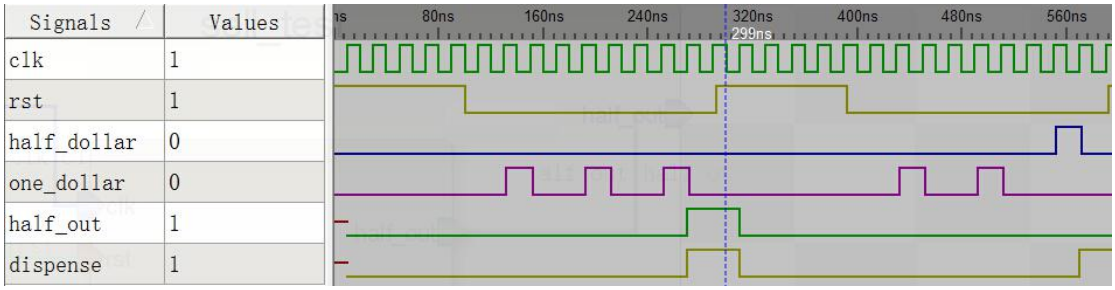


图 3-2-5 sell_test 仿真波形

3. sell_constrain 约束文件设计

（1）新建一个模块，命名为 sell_constrain，模块类型选择为 constrain，具有 4 个输入和 2 个输出，如图 3-2-6 所示。

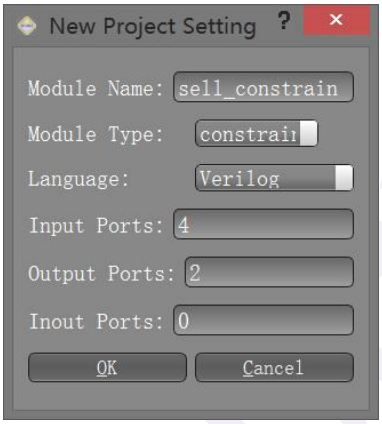


图 3-2-6 建立约束模块

（2）将约束模块和设计的 sell 模块保存到同一个目录下，用鼠标左键单击把 sell 模块从左侧的 ToolBox 添加到约束模块中；

（3）修改约束模块的端口名称。如图 3-2-7 所示。本设计中使用到的开发板硬件引脚如下：

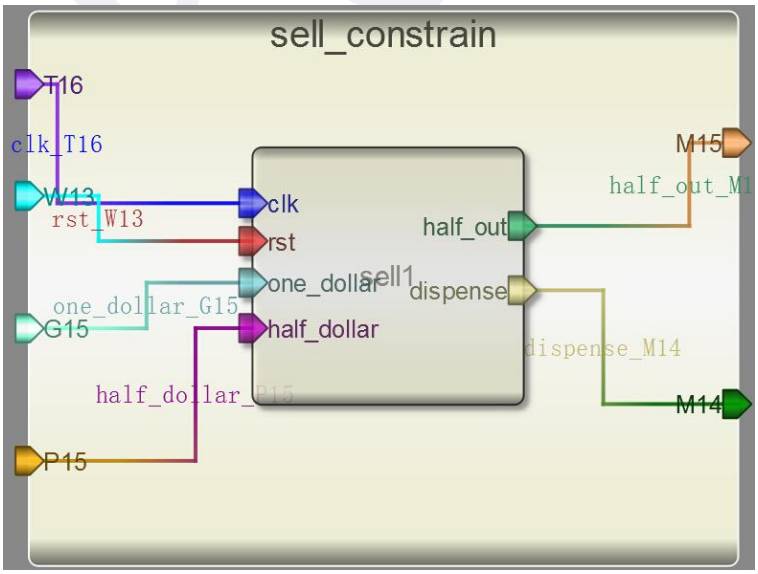


图 3-2-7 约束模块设计

clk 对应开发板的拨码开关 T16；

reset 对应开发板的拨码开关 W13；

 Copyright © Robei

one_dollar 对应开发板的拨码开关 G15；
half_dollar 对应开发板的拨码开关 P15；
dispense 对应开发板 LED 灯 M14；
half_out 对应开发板 LED 灯 M15；
（4）保存并执行，如果软件显示“Generate constrain file complete”，说明约束文件已经成功生成。

3.2.3. 板级验证

为了测试所设计 sell 的工作特性，我们选择合适的开发板进行板级间验证，该开发板需要搭载 XILINX 公司的 Z-7010 芯片，所以选用 VIVADO 设计平台进行 Synthesis、Implementation 和 Generate Bitstream，最终将生成的数据流文件下载到开发板内，并进行验证。

1. VIVADO 设计平台进行后端设计

1.1 启动 Vivado 软件并选择设备 XC7Z010CLG400-1 作为硬件对象，设计语言选用 Verilog，建立新的工程，添加通过 Robei 设计的文件 sell.v。
（1）打开 Vivado，选择开始>所有程序>Xilinx Design Tools> Vivado2013.4> Vivado2013.4；
（2）单击创建新项目 Create New Project 启动向导。你将看到创建一个新的 Vivado 项目对话框，单击 Next；
（3）在弹出的对话框中输入工程名 sell 及工程保存的位置，并确保 Create project subdirectory 复选框被选中，单击 Next；

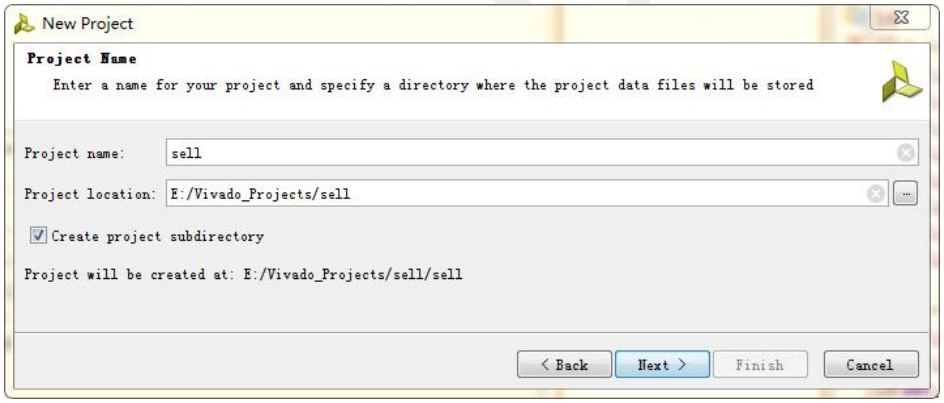


图 3-2-11 项目名称和位置输入

（4）选择项目类型表单的 RTL Project 选项，不勾选 Do not specify sources at this time 复选框，然后单击 Next；
（5）使用下拉按钮，选中 Verilog 作为目标文件和仿真语言；
（6）点击添加 Add Files 按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 sell.v，单击 Open，然后单击 Next 去添加现有的 IP 模型；
（7）由于我们没有任何的 IP 添加，跳过这一步，单击 Next 去添加约束形成；
（8）点击添加 Add Files 按钮，浏览到设计的 sell 模块目录下的 constrain 文件夹，选中其中的 sell_constrain.xdc 文件，然后单击 Next；
（9）在默认窗口中，按照图 5-1-13 所示设置 Filter 中的选项，然后在 Parts 中选择 XC7Z010CLG400-1，单击 Next；
（10）单击 Finish，本 Vivado 项目创建成功。

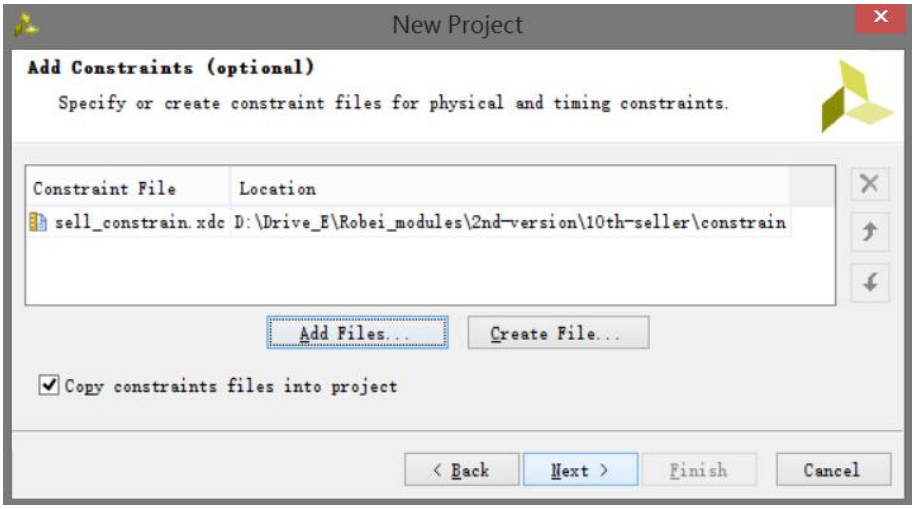


图 3-2-12 添加设计好的约束文件

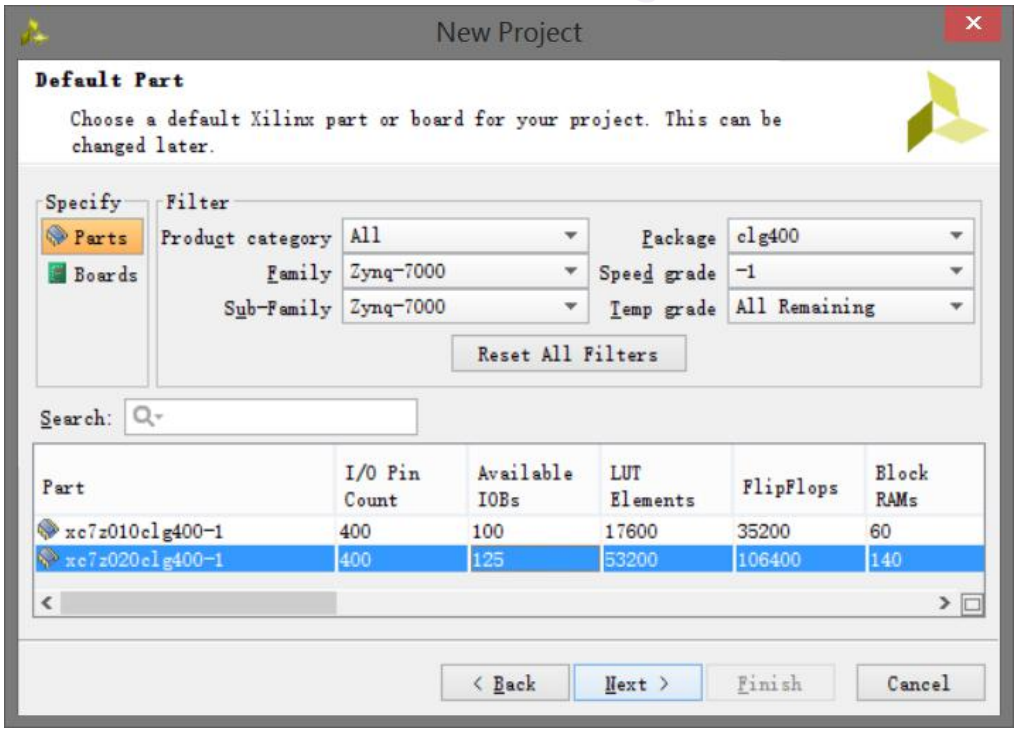


图 3-2-13 器件选型

1.2 打开 sell_constrain.xdc 文件，查看引脚约束源代码。

（1）在资源窗口 sources 中，展开约束文件夹，如图 3-2-14，然后双击打开 sell_constrain.xdc 进入文本编辑模式；

 Copyright © Robei

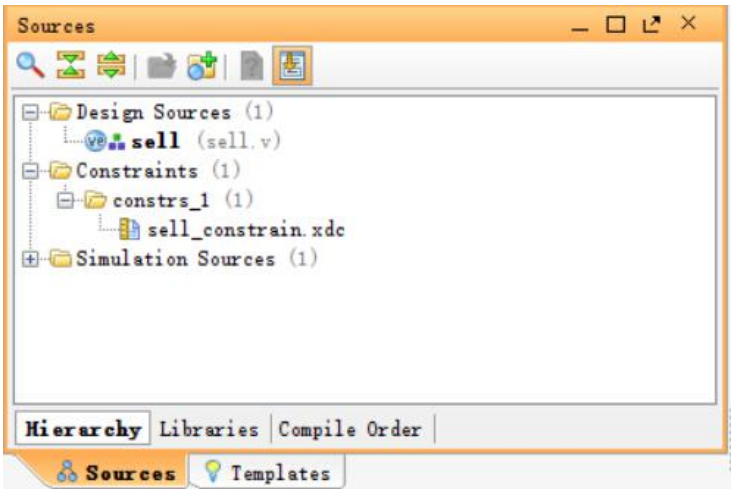


图 3-2-14 展开约束文件 sell_constrain.xdc

（2）Xilinx 设计约束文件分配 FPGA 位于主板上的开关和指示灯的物理 IO 地址，这些信息可以通过主板的原理图或电路板的用户手册来获得。

本次设计的约束文件代码是通过 Robei 软件自动生成，但是，Robei 软件目前生成的约束代码只有对输入输出端口的分配，在这个设计中，我们使用了一个通过开关控制的模拟时钟 clk，而非系统时钟，这种电路在综合的时候一般都会报错，所以，在约束文件最后，我们需要手动添加一句命令：

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]

来保证流程中不会出错。

完整的约束代码如下：

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN T16 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN W13 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property PACKAGE_PIN G15 [get_ports one_dollar]
set_property IOSTANDARD LVCMOS33 [get_ports one_dollar]
set_property PACKAGE_PIN P15 [get_ports half_dollar]
set_property IOSTANDARD LVCMOS33 [get_ports half_dollar]
set_property PACKAGE_PIN M15 [get_ports half_out]
set_property IOSTANDARD LVCMOS33 [get_ports half_out]
set_property PACKAGE_PIN M14 [get_ports dispense]
set_property IOSTANDARD LVCMOS33 [get_ports dispense]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

（3）点击 File->Save File 保存文件。

1.3 使用 Vivado 综合工具来综合设计，并进行 Implementation 任务实现。

（1）单击综合任务下拉菜单中的 Run Synthesis，综合过程将分析 sell.v 文件并生成网表文件。当综合过程完成了，且没有错误信息，将会弹出带有三个选项的完成对话框；

（2）如果有错误，根据错误信息提示修改，直至综合没有错误。然后选择 Run Implementation 选项，执行任务实现，然后单击 OK；

（3）任务实现过程将在综合后的设计上运行。当这个过程完成，且没有错误信息，将会弹出带有三个选项的任务实现完成对话框；

（4）如果有错误，根据错误信息提示修改，直至综合没有错误。

1.4 将开发板上的电源开关拨到 ON，生成比特流并打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击任务实现完成弹出的对话框中 Generate Bitstream 或者点击导航窗口中编程和调试任务中的 Generate Bitstream。比特流生成过程将在任务实现设计后运行。当完成比特流生成后会弹出有三个选项的完成对话框；

（5）这一过程将已经生成的 sell.bit 文件放在 sell.runs 目录下的 impl_1 目录下；

（6）选择打开硬件管理器 Open Hardware Manager 选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

（7）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target；

（8）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（9）单击 Next 以选择本地主机端口；

图 3-2-16 比特流生成

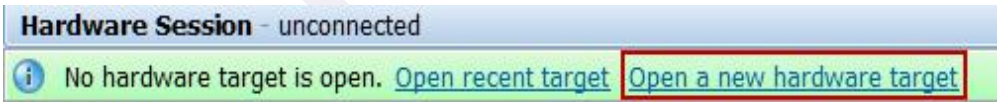


图 3-2-17 打开新的硬件目标

Copyright © Robei

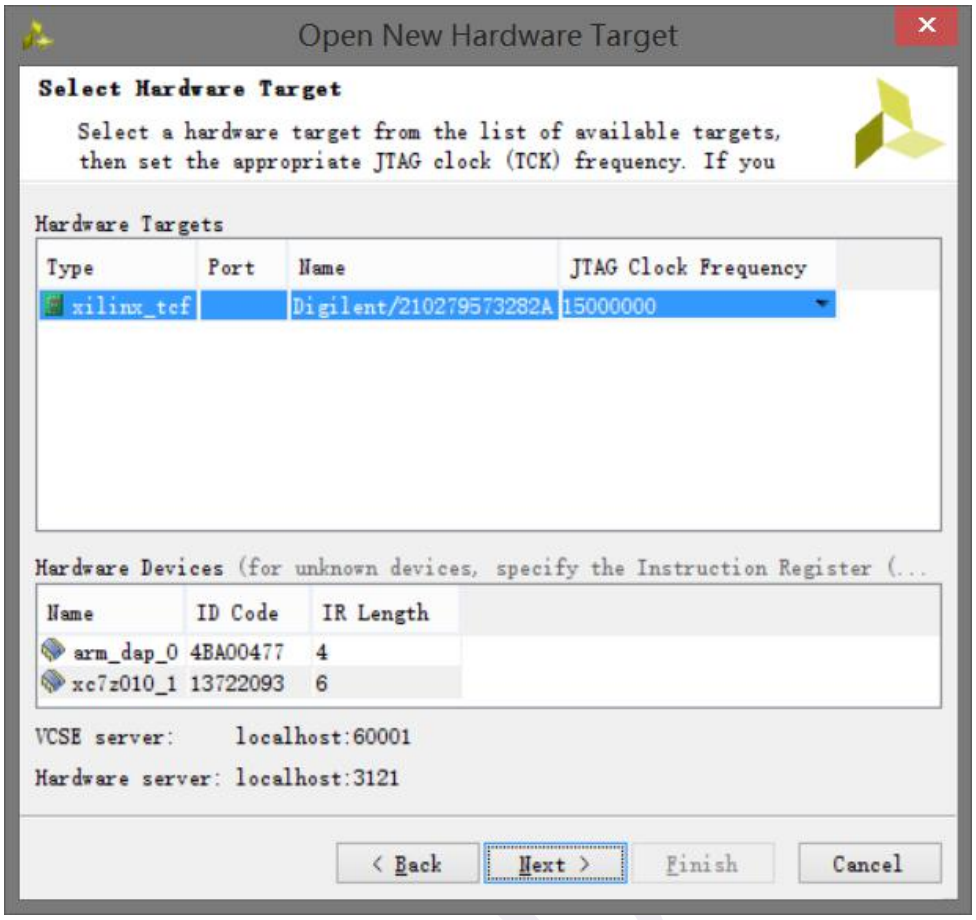


图 3-2-18 新的硬件指标的检测

（10）单击两次 Next，然后单击 Finish。未连接硬件会话状态更改为服务器名称并且器件被高亮显示，如图 3-2-19 所示。还要注意，该状态表明它还没有被编程；

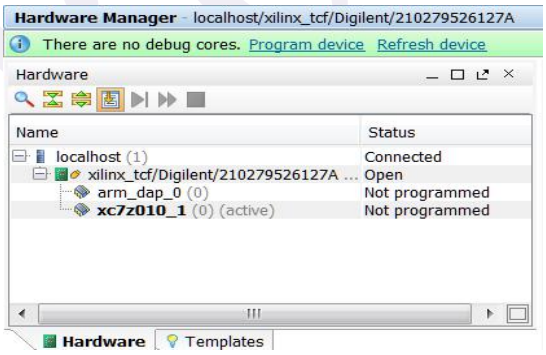


图 3-2-19 打开硬件会话

（11）选择器件，并验证 sell.bit 被选为常规选项卡中的程序文件；

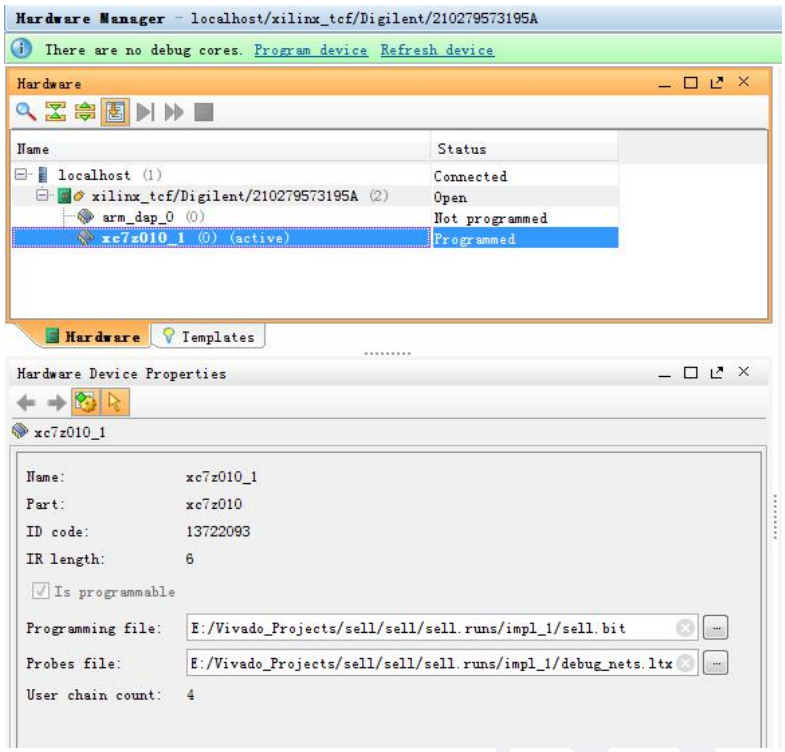


图 3-2-20 编程文件

（12) 在器件上单击鼠标右键，选择 Program device 或单击窗口上方弹出的 Program device-> XC7z010_1 链接到目标 FPGA 器件进行编程；

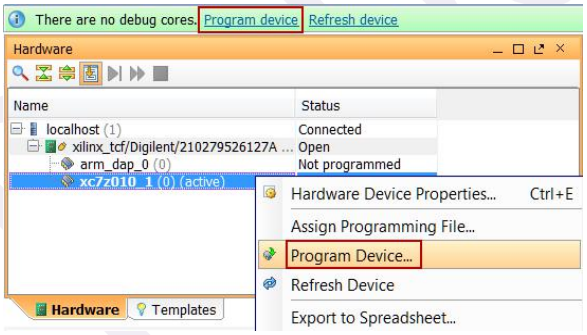


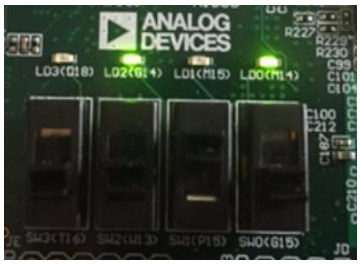
图 3-2-21 选择 FPGA 进行编程

（13）单击确定对 FPGA 进行编程。开发板上 Done 指示灯亮时，器件编程结束；
（14）通过控制拨动和按键开关的开闭来观察 LED（请参考前面的逻辑图）验证输出结果。

2. 开发板验证

首先，按住 reset (SW2)拨至高电平，再来回拨动 clk (SW3)1 到 2 次，进行复位操作；其次，设置不同的输入，这里分别进行了 one_dollar (SW0)，half_dollar (SW1)的验证。最后，按 clk 时钟键，经验证该自动贩卖机功能无误。

Copyright © Robei



one_dollar 时 half_out 灯亮　　half_dollar 时 half_out 灯不亮

图 3-2-22 不同输入时的 LED 灯显示实图

3.2.4. 问题与思考

思考做一个能卖多种价格的饮料的售货机。

Robei

第四章：复杂运算，板级体验

今天我们的设计将加深大家对 FPGA 设计的理解和熟练度。读者们在设计的过程中除了按照设计说明的流程进行外，还需要用一些时间分析设计代码，了解每一句 Verilog 语言的作用，甚至可以在保证正确的前提下进行一些修改来完成独特的功能。后端设计中由于开发板的硬件限制，这几个案例可能使用到了外接的电路板，如果读者们身边资源有限，可以通过修改数据位宽等尝试将自己的设计在有限资源的开发板上实现。

Robei

 Copyright © Robei

4.1 实例七 8 位移位寄存器的设计

4.1.1. 本章导读

设计目的

要求掌握 8 位移位寄存器原理，并根据原理设计 8 位移位寄存器模块以及设计相关 testbench，最后在 Robei 可视化仿真软件进行功能实现和仿真验证。

设计准备

有一个 8 比特的数据（初值设为 10011100）和一个移位设置数据 s，根据 s 的值不同，产生不同的移位。这里规定移位的方向是向右，由于是 8 比特，因此 s 的变化范围为 0 到 7。

4.1.2. 设计流程

1. shift 模型设计

（1）新建一个模型命名为 shift，类型为 module，同时具备 5 个输入和 1 个输出，每个引脚的属性和名称参照图 4-1-1 进行对应的修改。

Name	Inout	DataType	Datasize
clk	input	wire	1
clr	input	wire	1
en	input	wire	1
data_in	input	wire	8
set	input	wire	3
data_out	output	reg	8

图 4-1-1 shift 引脚的属性

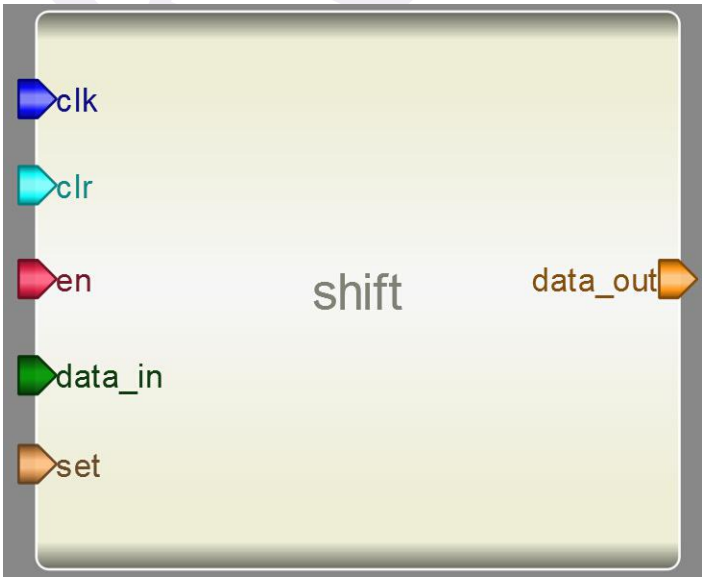


图 4-1-2 shift 界面图

（2）添加代码。点击模型下方的 Code 添加代码。
代码：

```
always@(posedge clk or negedge clr)
begin: shift_reg
    if(clr)
        data_out <= 8'b0;
    else if(en)
    begin
        case(set[2:0])
            3'b0: data_out <= data_in[7:0];
            3'b1: data_out <= {data_in[0],data_in[7:1]};
            3'd2: data_out <= {data_in[1:0],data_in[7:2]};
            3'd3: data_out <= {data_in[2:0],data_in[7:3]};
            3'd4: data_out <= {data_in[3:0],data_in[7:4]};
            3'd5: data_out <= {data_in[4:0],data_in[7:5]};
            3'd6: data_out <= {data_in[5:0],data_in[7:6]};
            3'd7: data_out <= {data_in[6:0],data_in[7]};
            default: data_out <= data_in[7:0];
        endcase
    end
end
```

（3）保存模型到一个文件夹(文件夹路径不能有空格和中文)中，运行并检查有无错误输出。

2. shift_test 测试文件设计

（1）新建一个具有 5 个输入和 1 个 输出的 shift_test 测试文件，记得将 Module Type 设置为“testbench”，各个引脚配置如图 4-1-3 所示。

Name	Inout	DataType	Datasize
clock	input	reg	1
clr	input	reg	1
en	input	reg	1
data	input	reg	8
set	input	reg	3
dataout	output	wire	8

图 4-1-3 shift_test 测试文件引脚的属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里会出现模型，单击该模型并在 shift _test 上添加，并连接引脚，如下图 4-1-4 所示：

 Copyright © Robei

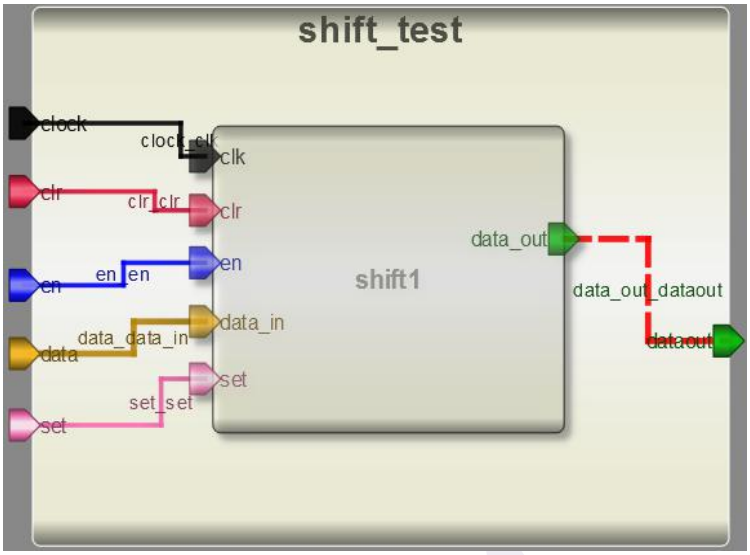


图 4-1-4 shift _test 界面图

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

测试代码：

```
initial begin
    clock=0;
    clr=0;
    en=1;
    data=8'b10011100;
    set=0;
    #1 clr=1;
    #2 clr=0;
    #40 $finish;
end

always #1 clock=~clock;
always #2 set=set+1;
```

（5）执行仿真并查看波形。查看输出信息。

检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 4-1-5 所示：

Signals	Values
clock	0
clr	0
set[2:0]	4
data[7:0]	9C
dataout[7:0]	93

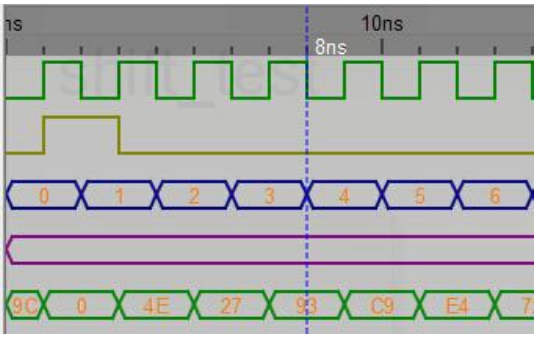


图 4-1-5 shift _test 的仿真波形

3. shift_constrain 测试文件的设计

（1）新建一个模块，命名为 shift_constrain，模块类型选择为 constrain，具有 14 个输入和 8 个输出，如图 4-1-6 所示。

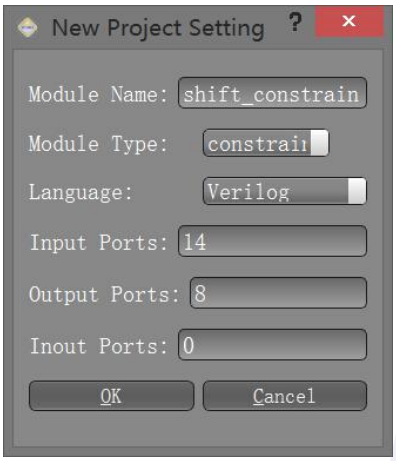


图 4-1-6 新建约束模块

（2）修改端口名称并进行连线。其中，shift 模块的输入 data_in 和输出 data_out 分别对应 8 个端口，输入 set 对应 3 个端口。本设计中，各个端口对应的开发板引脚如下所示：

clk 对应开发板按键开关 R18；

clr 对应开发板按键开关 P16；

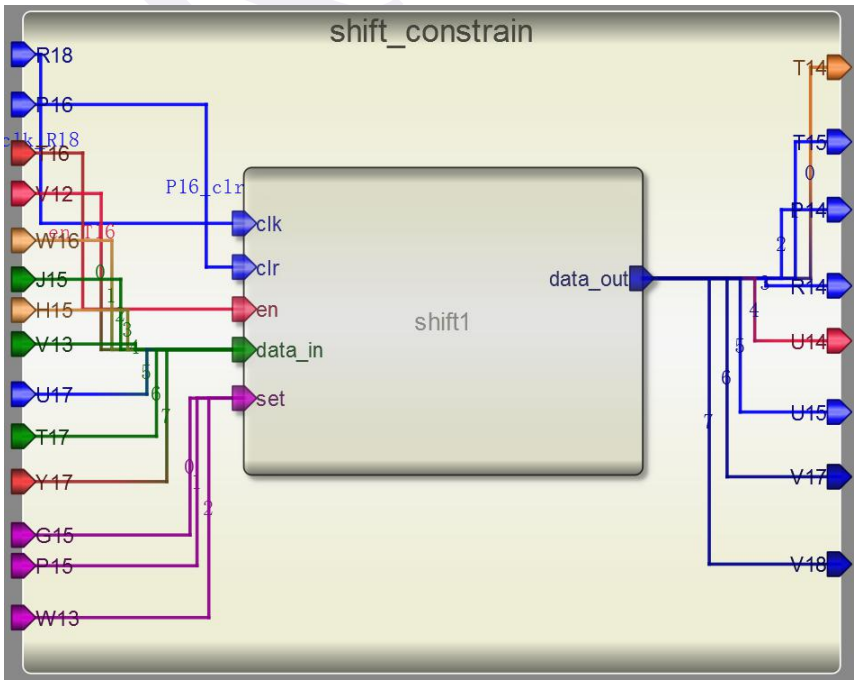
en 对应开发板拨码开关 T16；

set[0],set[1],set[2]对应 G15，P15，W13；

data_in[0]——data_in[7]分别对应 V12，W16，J15，H15，V13，U17，T17，Y17；

data_out[0]——data_out[7]分别对应 T14，T15，P14，R14，U14，U15，V17，V18；

输入 data_in 对应的连线名称需要改成 0——7，set 的连线名称则要改成 0,1,2。修改后的约束模块如图 4-1-7 所示。



 Copyright © Robei

图 4-1-7 约束模块的设计

（3）保存并执行，如果软件显示“Generate constrain file complete”，说明约束文件已经成功生成。

4.1.3. 板级验证

为了测试所设计 shift 的工作特性，选择开发板搭载 XILINX 公司的 Z-7010 芯片，选用 VIVADO 设计平台进行 Synthesis、Implementation 和 Generate Bitstream，最终将生成的数据流文件下载到开发板内，并进行验证。

1. VIVADO 设计平台进行后端设计

1.1 启动 Vivado 软件并选择设备 XC7Z010CLG400-1（ZYBO）作为硬件对象，设计语言选用 Verilog，建立新的工程，添加通过 Robei 设计的文件 shift.v。

（1）打开 Vivado，选择开始>所有程序>Xilinx Design Tools> Vivado2013.4> Vivado2013.4；

（2）单击创建新项目 Create New Project 启动向导。你将看到创建一个新的 Vivado 项目对话框，单击 Next；

（3）在弹出的对话框中输入工程名 shift 及工程保存的位置，并确保 Create project subdirectory 复选框被选中，单击 Next；

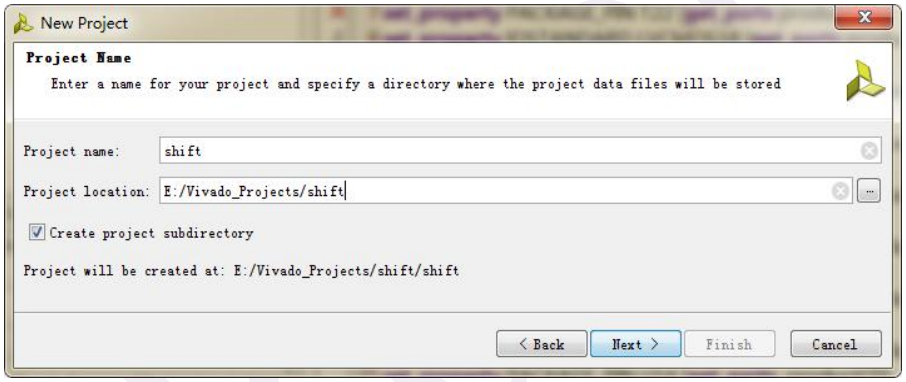


图 4-1-11 项目名称和位置输入

（4）选择项目类型表单的 RTL Project 选项，不勾选 Do not specify sources at this time 复选框，然后单击 Next；

（5）使用下拉按钮，选中 Verilog 作为目标文件和仿真语言；

（6）点击添加 Add Files 按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 shift.v，单击 Open，然后单击 Next 去添加现有的 IP 模型；

（7）由于我们没有任何的 IP 添加，跳过这一步，直接单击 Next 去添加约束形成；

（8）点击添加 Add Files 按钮，浏览到刚刚建立约束模块目录下的 constrain 文件夹，选择 shift_constrain.xdc，单击 Open 进行添加，然后单击 Next；

（9）在默认窗口中，按照图 4-1-12 所示，设置 Filer 中的选项，然后在 Parts 中选择对应的 XC7Z010CLG400-1，单击 Next；

（10）单击 Finish，本 Vivado 项目创建成功。

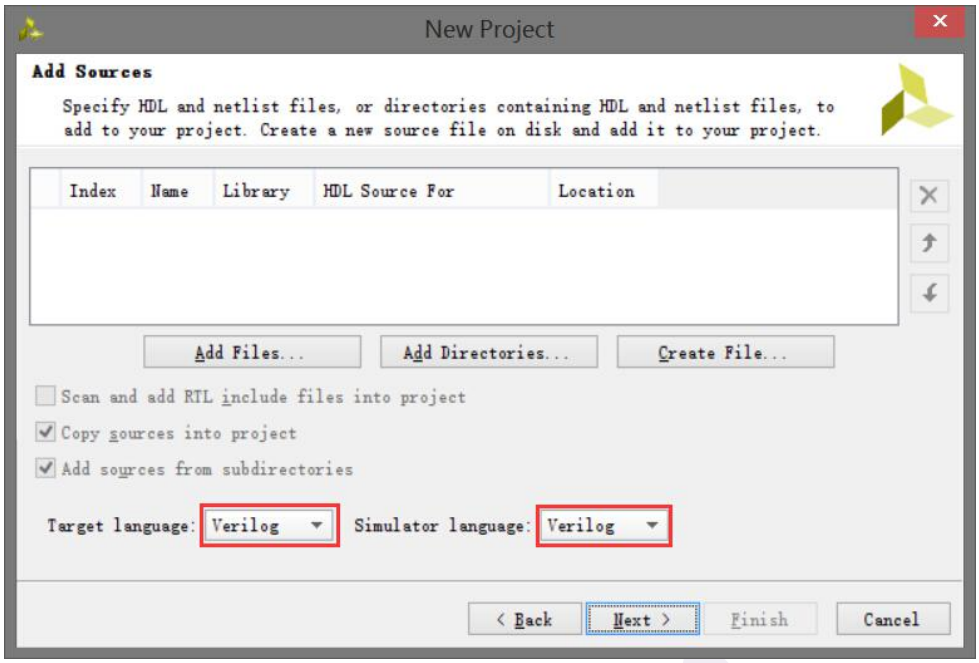


图 4-1-12 选择目标文件和仿真语言

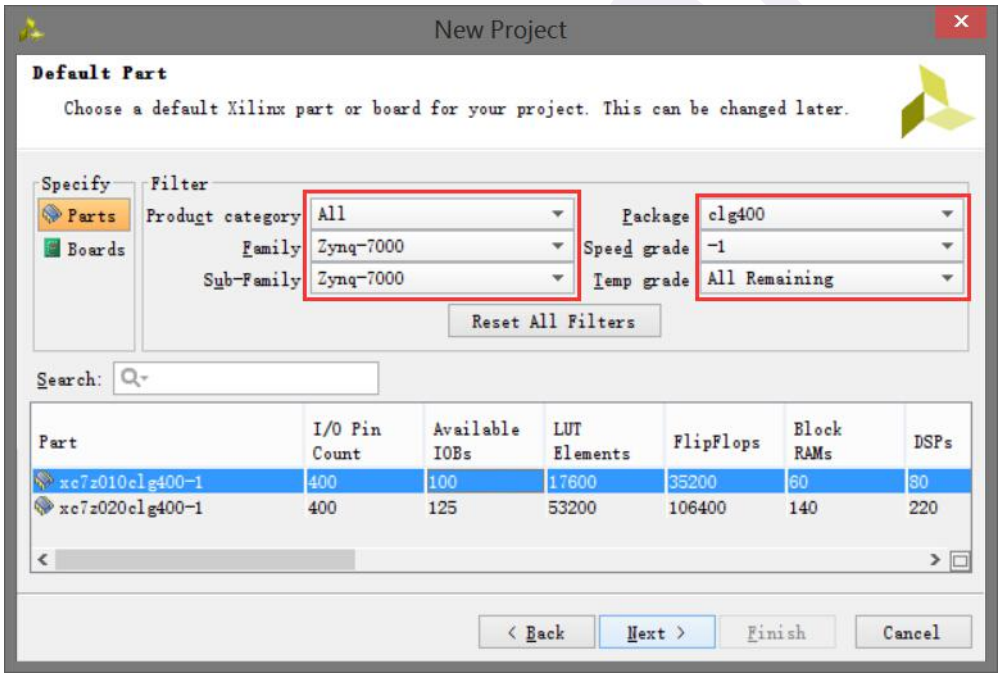


图 4-1-13 器件选型

1.2 打开 shift_constrain.xdc 文件，查看引脚约束源代码。

（1）在资源窗口 sources 中，展开约束文件夹，如下图 4-1-13 所示，然后双击即可打开 uart_led_pins.xdc 进入文本编辑模式；

Copyright © Robei

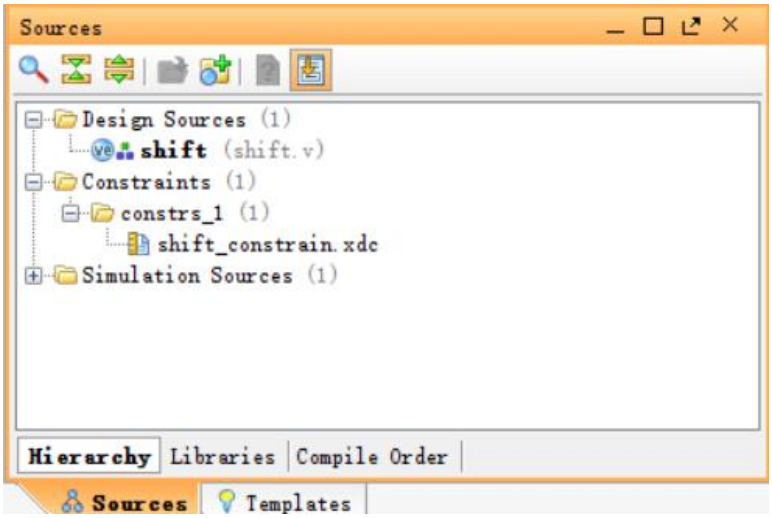


图 4-1-13 展开约束文件 shift_constrain.xdc

（2）Xilinx 设计约束文件分配 FPGA 位于主板上的开关和指示灯的物理 IO 地址，这些信息可以通过主板的原理图或电路板的用户手册来获得。

本次设计的约束文件代码是通过 Robei 软件自动生成，但是，Robei 软件目前生成的约束代码只有对输入输出端口的分配，在这个设计中，我们使用了一个通过开关控制的模拟时钟 clk，而非系统时钟，这种电路在综合的时候一般都会报错，所以，在约束文件最后，我们需要手动添加一句命令：

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]

来保证流程中不会出错。

完整的约束代码如下：

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN R18 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN P16 [get_ports clr]
set_property IOSTANDARD LVCMOS33 [get_ports clr]
set_property PACKAGE_PIN T16 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]
set_property PACKAGE_PIN V12 [get_ports data_in[0]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[0]]
set_property PACKAGE_PIN W16 [get_ports data_in[1]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[1]]
set_property PACKAGE_PIN J15 [get_ports data_in[2]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[2]]
set_property PACKAGE_PIN H15 [get_ports data_in[3]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[3]]
set_property PACKAGE_PIN V13 [get_ports data_in[4]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[4]]
set_property PACKAGE_PIN U17 [get_ports data_in[5]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[5]]
set_property PACKAGE_PIN T17 [get_ports data_in[6]]
```

Copyright © Robei

```
set_property IOSTANDARD LVCMOS33 [get_ports data_in[6]]
set_property PACKAGE_PIN Y17 [get_ports data_in[7]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[7]]
set_property PACKAGE_PIN G15 [get_ports set[0]]
set_property IOSTANDARD LVCMOS33 [get_ports set[0]]
set_property PACKAGE_PIN P15 [get_ports set[1]]
set_property IOSTANDARD LVCMOS33 [get_ports set[1]]
set_property PACKAGE_PIN W13 [get_ports set[2]]
set_property IOSTANDARD LVCMOS33 [get_ports set[2]]
set_property PACKAGE_PIN T14 [get_ports data_out[0]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[0]]
set_property PACKAGE_PIN T15 [get_ports data_out[1]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[1]]
set_property PACKAGE_PIN P14 [get_ports data_out[2]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[2]]
set_property PACKAGE_PIN R14 [get_ports data_out[3]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[3]]
set_property PACKAGE_PIN U14 [get_ports data_out[4]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[4]]
set_property PACKAGE_PIN U15 [get_ports data_out[5]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[5]]
set_property PACKAGE_PIN V17 [get_ports data_out[6]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[6]]
set_property PACKAGE_PIN V18 [get_ports data_out[7]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[7]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

（3）点击 File—>Save File 保存文件。

1.3 使用 Vivado 综合工具来综合设计，并进行 Implementation 任务实现。

（1）单击综合任务下拉菜单中的 Run Synthesis，综合过程将分析 shift.v 文件并生成网表文件。当综合过程完成了，且没有错误信息，将会弹出带有三个选项的完成对话框；

（2）如果有错误，根据错误信息提示进行修改，直至综合没有错误成功完成。然后选择 Run Implementation 选项，执行任务实现，然后单击 OK；

（3）任务实现过程将在综合后的设计上运行。当这个过程完成，且没有错误信息，将会弹出带有三个选项的任务实现完成对话框；

（4）如果有错误，根据错误信息提示修改，直至综合没有错误。

1.4 将开发板上的电源开关拨到 ON，生成比特流并打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击任务实现完成弹出的对话框中 Generate Bitstream 或者点击导航窗口中编程和调试任务中的 Generate Bitstream。当完成比特流生成后会弹出有三个选项的完成对话框；

（5）这一过程将已经生成的 shift.bit 文件放在 shift.runs 目录下的 impl_1 目录下；

 Copyright © Robei

（6）选择打开硬件管理器 Open Hardware Manager 选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

（7）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target；

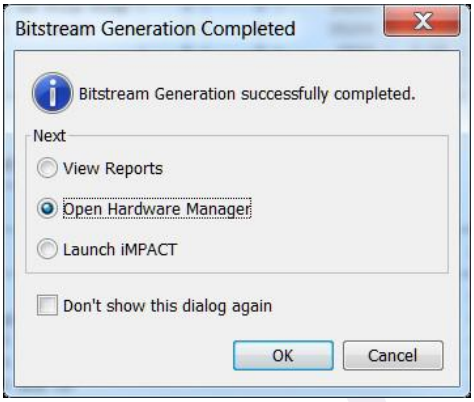


图 4-1-15 比特流生成

图 4-1-16 打开新的硬件目标

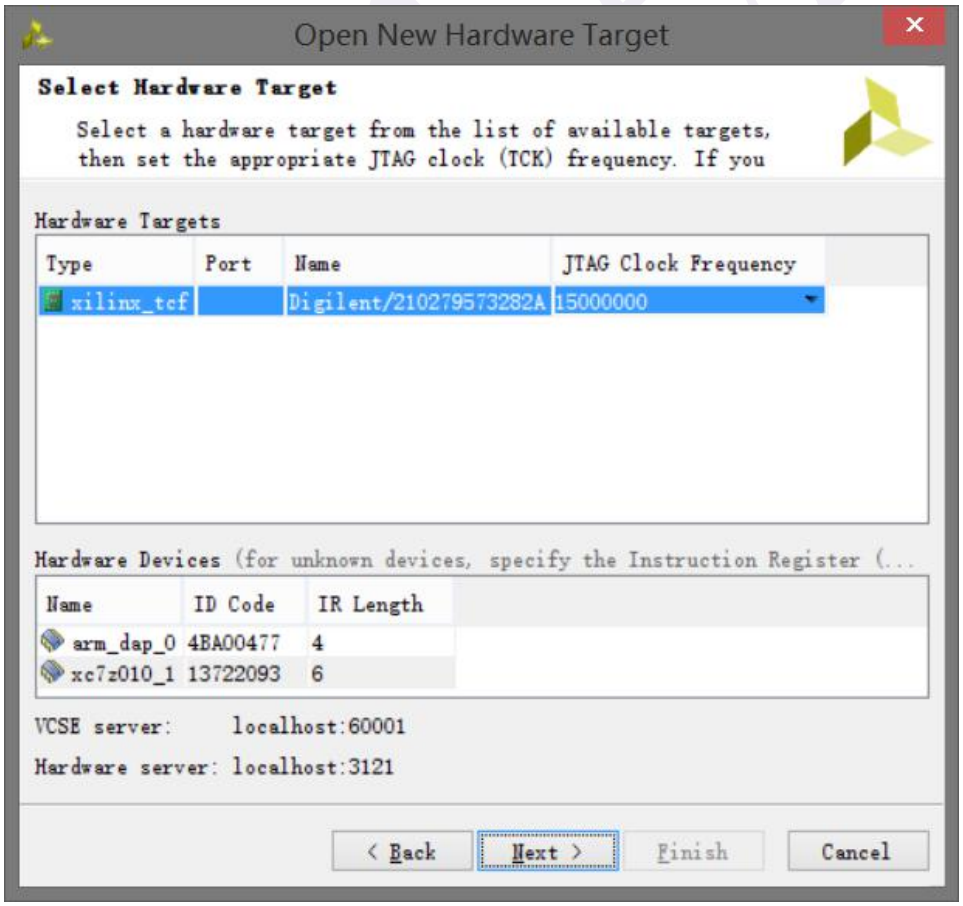


图 4-1-17 新的硬件指标的检测

（8）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（9）单击 Next 以选择本地主机端口；

（10）单击两次 Next，然后单击 Finish。未连接硬件会话状态更改为服务器名称并且器件被高亮显示。还要注意，该状态表明它还没有被编程；

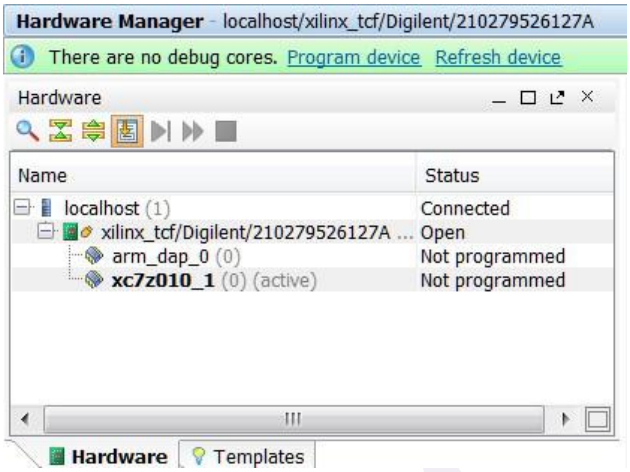


图 4-1-18 打开硬件会话

（11）选择器件，并验证 shift.bit 被选为常规选项卡中的程序文件；

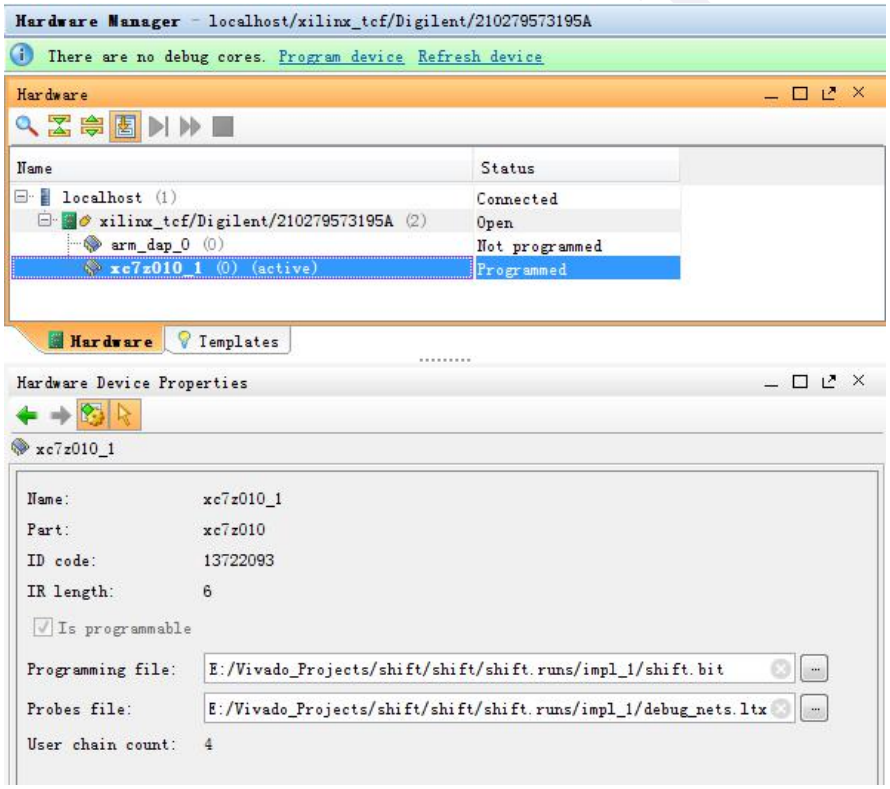


图 4-1-19 编程文件

（12）在器件上单击鼠标右键，选择 Program device 或单击窗口上方弹出的 Program device-> XC7z010_1 链接到目标 FPGA 器件进行编程，如图 4-1-20 所示；

Copyright © Robei

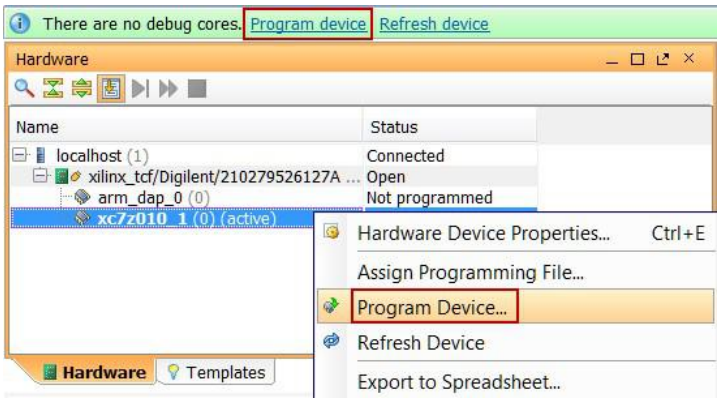


图 4-1-20 选择 FPGA 进行编程

（13）单击确定对 FPGA 进行编程。开发板上 Done 指示灯亮时，器件编程结束；
（14）通过控制拨动和按键开关的开闭来观察 LED（请参考前面的逻辑图）验证输出结果。

2. 开发板验证

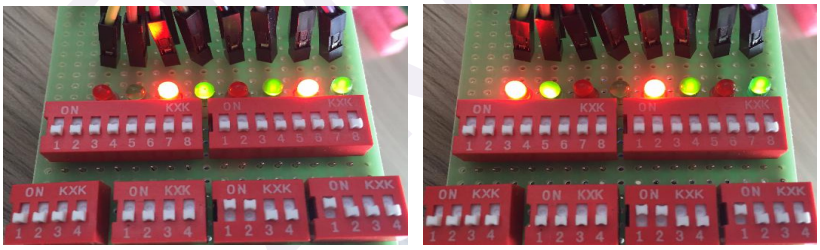
首先，按住 clr(BTN1)不放，按 1 到 2 次 clk(BTN0)，进行复位操作。

然后，将 en（SW3）设置为高电平，设置输入数据 data_in[0~7] (Sw[0~7])和位移控制信号 set[0~3] (SW0~1)，这里选择输入数据为 00110111，位移控制信号分别为 000，010，100，111 进行验证；

最后，每更改一次位移控制信号，按一下 clk 时间键，然后观察输出数据 data_out[0~7] (Led[0~7])，验证移位寄存器功能是否正确。

set = 000 时，data_out 应为 00110111；set = 010 时，data_out 应为 11001101；
set = 100 时，data_out 应为 01110011；set = 111 时，data_out 应为 01101110；

00110111 11001101



01110011 01101110

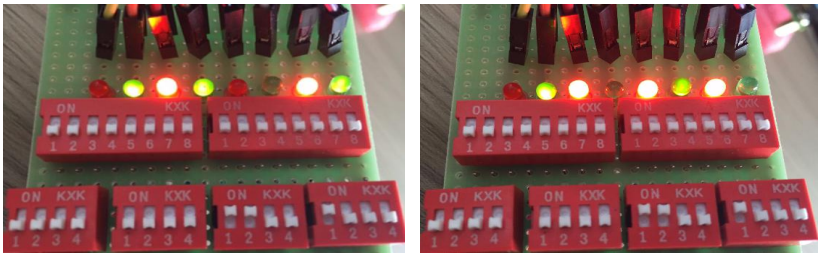


图 4-1-21 每一个 set[2:0]对应的 LED 灯显示实图

通过验证结果证明该 8 位移位寄存器符合设计的要求。

4.1.4. 问题与思考

1.如何在一个模块上面设计一个移位寄存器，该移位寄存器既有右移功能，还有左移、双向移动的功能，这些功能通过一个值的变化来切换？

2.如何实现扭环形移位寄存器？

4.2 实例八 带符号位小数的加法设计

4.2.1. 本章导读

设计目的

设计一个带符号位的小数加法器，该加数和被加数的总位数为 32 位，其中小数 15 位，整数占 16 位，剩下一位符号位。设计该加法器模块以及设计 testbench，最后在 Robei 可视化仿真软件进行功能实现和仿真验证。

设计原理

输入数据的最高位是符号位，其余的位数是数值位。首先通过比较两个输入数据的符号位判断输出数据的符号位：比如输入都是正数则结果一定是正数，输入都是负数则结果一定是负数，输入一正一负的话则比较两个数据的数值大小进行判定。

然后对两个数据的数值位进行计算，得到输出数据的数值位，最后给输出数据添加符号位完成全部运算。

模型格式：sum = addend + addend

输入格式：

|1| <- N-Q-1 bits -> | <--- Q bits --> |

|S| IIIIIIIIIIIIIII |FFFFFFFFFFFFFF|

输入数据：

a - addend 1 b - addend 2

输出格式：

|1| <- N-Q-1 bits -> | <--- Q bits --> |

|S| IIIIIIIIIIIIIII |FFFFFFFFFFFFFF|

输出数据：

c - result

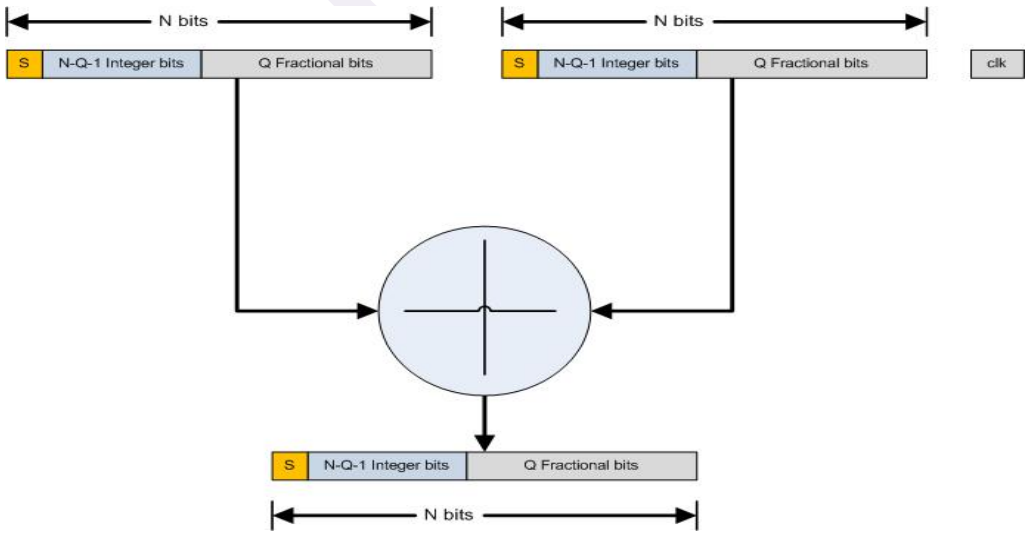


图 4-2-1 模型结构图

 Copyright © Robei

4.2.2. 设计流程

1. qadd 模型设计

（1）新建一个模型命名为 qadd，类型为 module，同时具备 2 个输入和 1 个输出，每个引脚的属性和名称参照下图 4-2-2 进行对应的修改。

Ports:

Name	Inout	DataType	Datasize
a	input	wire	4
b	input	wire	4
c	output	wire	4

图 4-2-2 qadd 引脚的属性

图 4-2-3 qadd 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

代码如下：

```
parameter N = 4;
reg [N-1:0] res;
assign c = res;
always @(a or b)
begin
    if(a[N-1] == b[N-1])
    begin
        res[N-2:0] = a[N-2:0] + b[N-2:0];
        res[N-1] = a[N-1];
    end
    else if(a[N-1] == 0 && b[N-1] == 1)
    begin
        if( a[N-2:0] > b[N-2:0] )
        begin
            res[N-2:0] = a[N-2:0] - b[N-2:0];
```

```
                res[N-1] = 0;
            end
            else
            begin
                res[N-2:0] = b[N-2:0] - a[N-2:0];
                if (res[N-2:0] == 0)
                    res[N-1] = 0;
                else
                    res[N-1] = 1;
            end
        end
        else
        begin
            if( a[N-2:0] > b[N-2:0] )
            begin
                res[N-2:0] = a[N-2:0] - b[N-2:0];
                if (res[N-2:0] == 0)
                    res[N-1] = 0;
                else
                    res[N-1] = 1;
            end
            else
            begin
                res[N-2:0] = b[N-2:0] - a[N-2:0];
                res[N-1] = 0;
            end
        end
    end
```

2. qadd_test 测试文件的设计

（1）新建一个 2 输入 1 输出的 qadd_test 测试文件，记得将 Module Type 设置为“testbench”，各个引脚配置如图 4-2-4 所示。

Name	Inout	DataType	Datasize	Function
a	input	reg	4	
b	input	reg	4	
c	output	wire	4	

图 4-2-4 qadd_test 测试文件引脚的属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现模型，单击该模型并在 qadd _test 上添加，并连接引脚，如下图 4-2-5 所示：

 Copyright © Robei

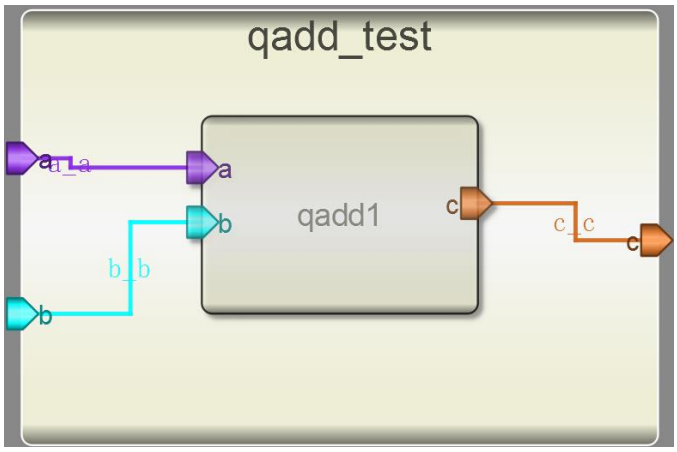


图 4-2-5 qadd_test 界面图

（4）输入激励代码。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。（由于在验证功能的时候，有无小数点并没有本质上的区别，故在验证时使用了两个4位二进制数，最高位为符号位）

激励代码：

```
initial begin
    a = 4'b0001;
    b = 4'b0011;
    #10
    a = 4'b1011;
    b = 4'b1011;
    #10
    a = 4'b0001;
    b = 4'b1101;
    #10
    a = 4'b1110;
    b = 4'b0011;
    #10
    $finish;
end
```

（5）执行仿真并查看波形。查看输出信息。检查没有错误之后查看波形。

点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 4-2-6 所示：

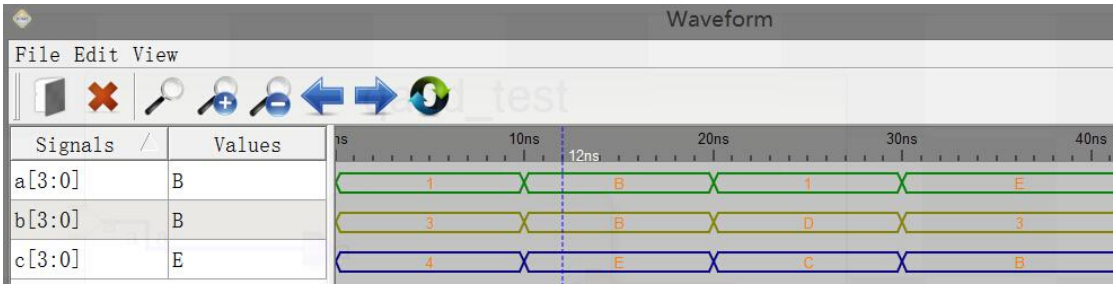


图 4-2-6 qadd _test 的仿真波形

Robei

3. 约束模块和约束文件设计

（1）新建一个模块，模块类型选择为 constrain，具有 8 个输入和 4 个输出。（因为设计的模块有两个 4 位的输入端和一个 4 位的输出端）

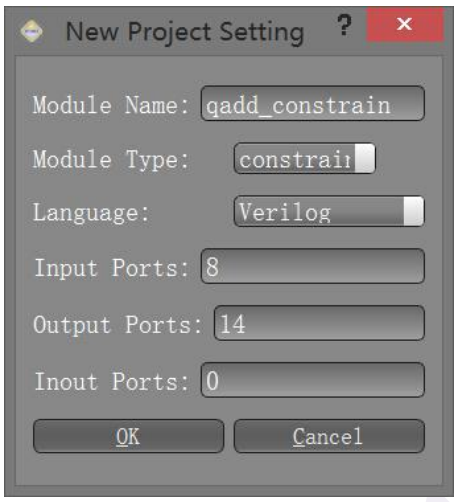


图 4-2-7 建立 constrain 模块

（2）连线。将 4 个输入与输入 a 相连，再将另外 4 个输入与 b 相连。4 个输出则全部与模块的输出 c 相连。

（3）修改约束模块端口和连接线的名称。由于本设计需要使用的硬件资源较多，因此使用了一些外接的开关和 LED 灯。这些硬件在开发板上对应的引脚如下：

a[0],a[1],a[2],a[3]分别对应的引脚为 V12,W16,J15,H15;

b[0],b[1],b[2],b[3]分别对应的引脚为 V13,U17,T17,Y17;

c[0],c[1],c[2],c[3]分别对应的引脚为 T14,T15,P14,R14;

修改完端口名称后记得把对应的连接线名称修改为 0,1,2,3。修改后的约束模块如图 4-2-8 所示：

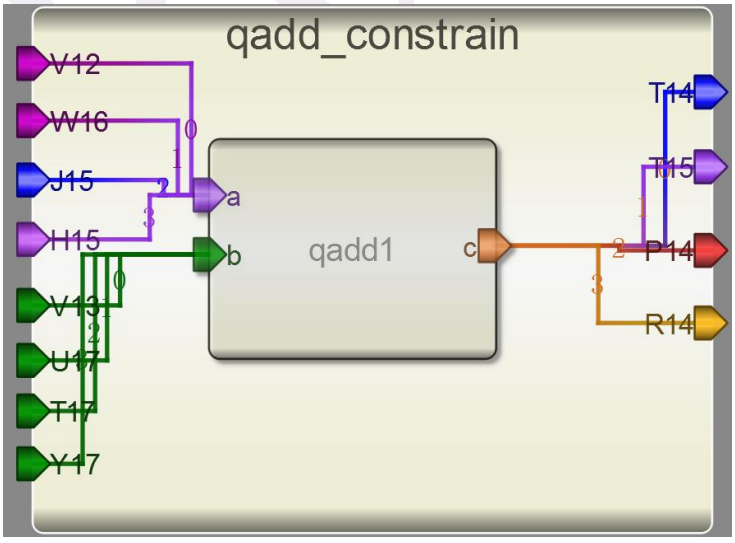


图 4-2-8 约束模块设计

（4）保存并运行，如果没有错误，即可在目录下的 constrain 文件夹里找到生成的 xdc 约束文件。可以通过 View->CodeView 查看约束文件的代码。

4.2.3. 板级验证

 Copyright © Robei

1. VIVADO 设计平台进行后端设计

1.1 启动 Vivado 软件并选择设备 XC7Z010CLG400-1 作为硬件对象，设计语言选用 Verilog，建立新的工程，添加通过 Robei 设计的文件 qadd.v。

（1）打开 Vivado，选择开始>所有程序>Xilinx Design Tools> Vivado2013.4> Vivado2013.4；

（2）单击创建新项目 Create New Project 启动向导。

（3）在弹出的对话框中输入工程名 qadd 及工程保存的位置，并确保 Create project subdirectory 复选框被选中，单击 Next；

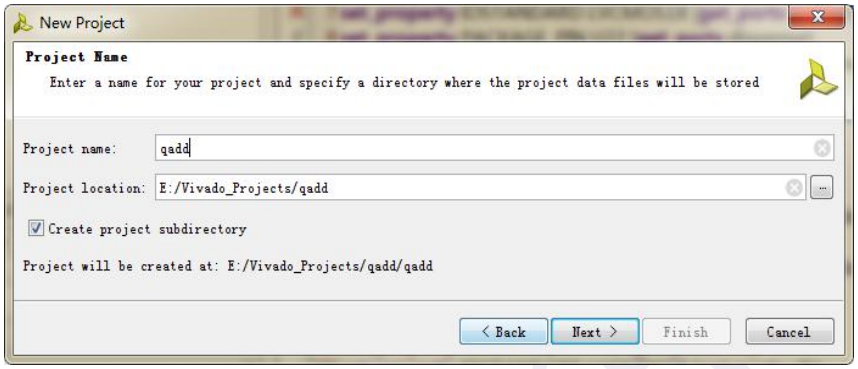


图 4-2-12 项目名称和位置输入

（4）选择项目类型表单的 RTL Project 选项，不要勾选 Do not specify sources at this time 复选框以便添加源文件，然后单击 Next；

（5）使用下拉按钮，选中 Verilog 作为目标文件和仿真语言；

（6）点击添加 Add Files 按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 qadd.v，单击 Open，然后单击 Next 去添加现有的 IP 模型；

（7）由于我们没有任何的 IP 添加，单击 Next 去添加约束形成；

（8）在添加约束文件界面，点击 Add Files，找到并选择刚才生成的 constrain 文件夹下的 qadd_constrain.xdc 文件，并点击确定；

（9）在器件选择窗口中，按照图 4-2-14 所示设置 Filer 中的选项，然后在 Parts 中选择 XC7Z010CLG400-1，单击 Next；

（10）单击 Finish，本 Vivado 项目创建成功。

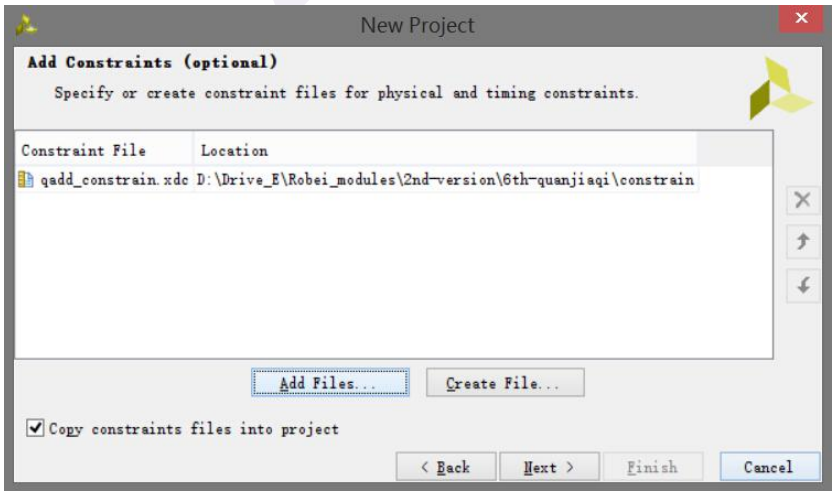


图 4-2-13 添加约束文件

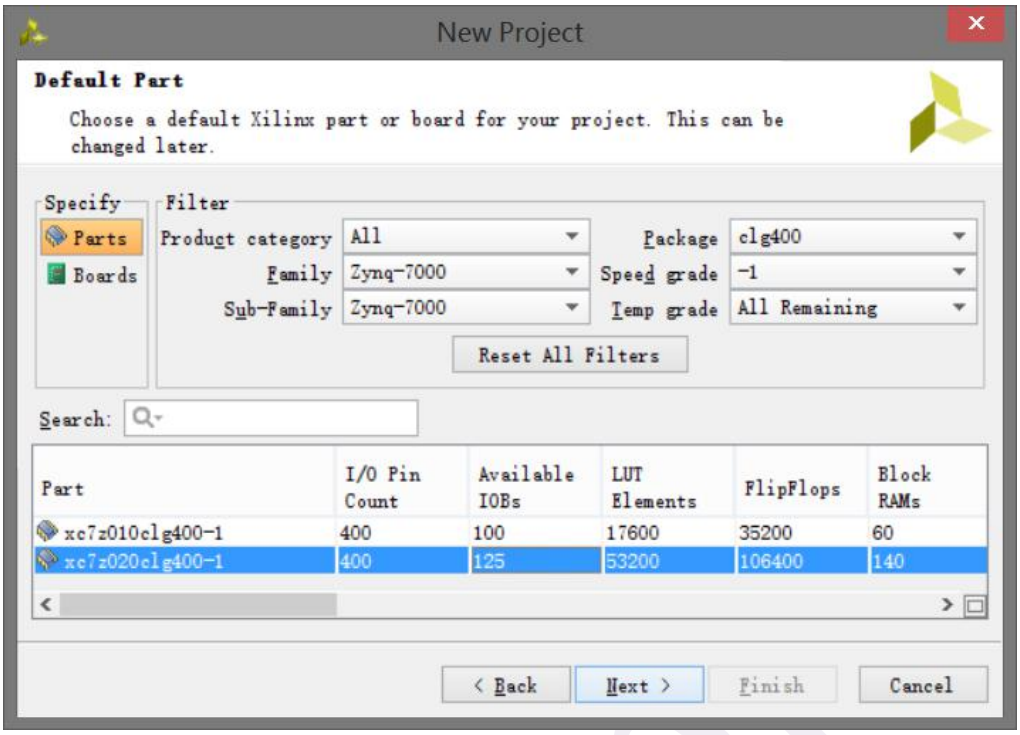


图 4-2-14 器件选型

1.2 打开 qadd_constrain.xdc 文件，查看引脚约束源代码。

（1）在资源窗口 sources 中，展开约束文件夹，然后双击打开 qadd_constrain.xdc 进入文本编辑模式；

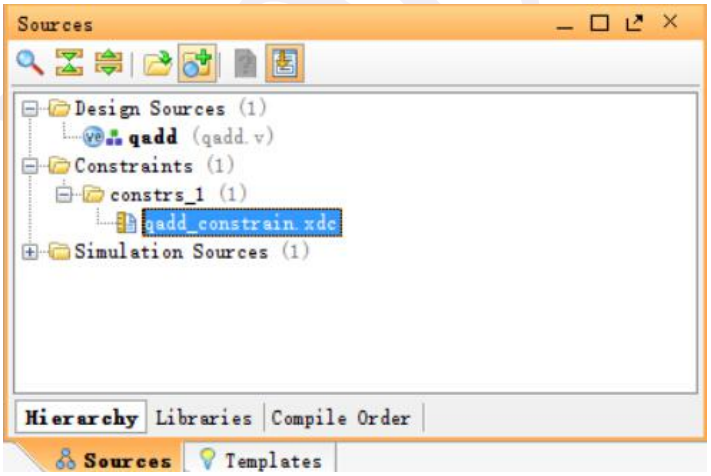


图 4-2-15 展开约束文件 qadd_constrain.xdc

（2）这个设计中的 xdc 文件是 Robei 软件自动生成的，我们以编辑的方式打开这个约束文件后，可以检查一下这里的代码是否和我们希望的约束文件一致。

qadd_constrain.xdc 约束文件代码如下：

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN V12 [get_ports a[0]]
set_property IOSTANDARD LVCMOS33 [get_ports a[0]]
```

 Copyright © Robei

```
set_property PACKAGE_PIN W16 [get_ports a[1]]
set_property IOSTANDARD LVCMOS33 [get_ports a[1]]
set_property PACKAGE_PIN J15 [get_ports a[2]]
set_property IOSTANDARD LVCMOS33 [get_ports a[2]]
set_property PACKAGE_PIN H15 [get_ports a[3]]
set_property IOSTANDARD LVCMOS33 [get_ports a[3]]
set_property PACKAGE_PIN V13 [get_ports b[0]]
set_property IOSTANDARD LVCMOS33 [get_ports b[0]]
set_property PACKAGE_PIN U17 [get_ports b[1]]
set_property IOSTANDARD LVCMOS33 [get_ports b[1]]
set_property PACKAGE_PIN T17 [get_ports b[2]]
set_property IOSTANDARD LVCMOS33 [get_ports b[2]]
set_property PACKAGE_PIN Y17 [get_ports b[3]]
set_property IOSTANDARD LVCMOS33 [get_ports b[3]]
set_property PACKAGE_PIN T14 [get_ports c[0]]
set_property IOSTANDARD LVCMOS33 [get_ports c[0]]
set_property PACKAGE_PIN T15 [get_ports c[1]]
set_property IOSTANDARD LVCMOS33 [get_ports c[1]]
set_property PACKAGE_PIN P14 [get_ports c[2]]
set_property IOSTANDARD LVCMOS33 [get_ports c[2]]
set_property PACKAGE_PIN R14 [get_ports c[3]]
set_property IOSTANDARD LVCMOS33 [get_ports c[3]]
```

（3）点击 File—>Save File 保存文件。

1.3 使用 Vivado 综合工具来综合设计，并进行 Implementation 任务实现。

（1）单击综合任务下拉菜单中的 Run Synthesis，综合过程会分析 qadd.v 文件并生成网表文件。当综合过程完成了，且没有错误信息，将会弹出带有三个选项的完成对话框；

（2）如果有错误，根据错误信息提示修改，直至综合没有错误。然后选择 Run Implementation 选项，执行任务实现，然后单击 OK；

（3）任务实现过程将在综合后的设计上运行。当这个过程完成，且没有错误信息，将会弹出带有三个选项的任务实现完成对话框；

（4）如果有错误，根据错误信息提示修改，直至综合没有错误。

1.4 将开发板上的电源开关拨到 ON，生成比特流并打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击任务实现完成弹出的对话框中 Generate Bitstream 或者点击导航窗口中编程和调试任务中的 Generate Bitstream。比特流生成过程将在任务实现设计后运行。当完成比特流生成后会弹出有三个选项的完成对话框；

（5）这一过程将已经生成的 qadd.bit 文件放在 qadd.runs 目录下的 impl_1 目录下；

（6）选择打开硬件管理器 Open Hardware Manager 选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

（7）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target；

图 4-2-17 比特流生成

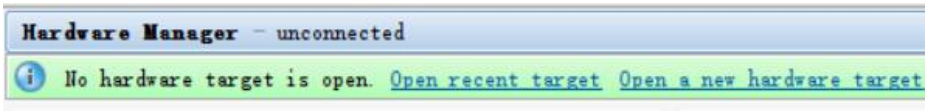


图 3-2-18 打开新的硬件目标

（8）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（9）单击 Next 以选择本地主机端口；

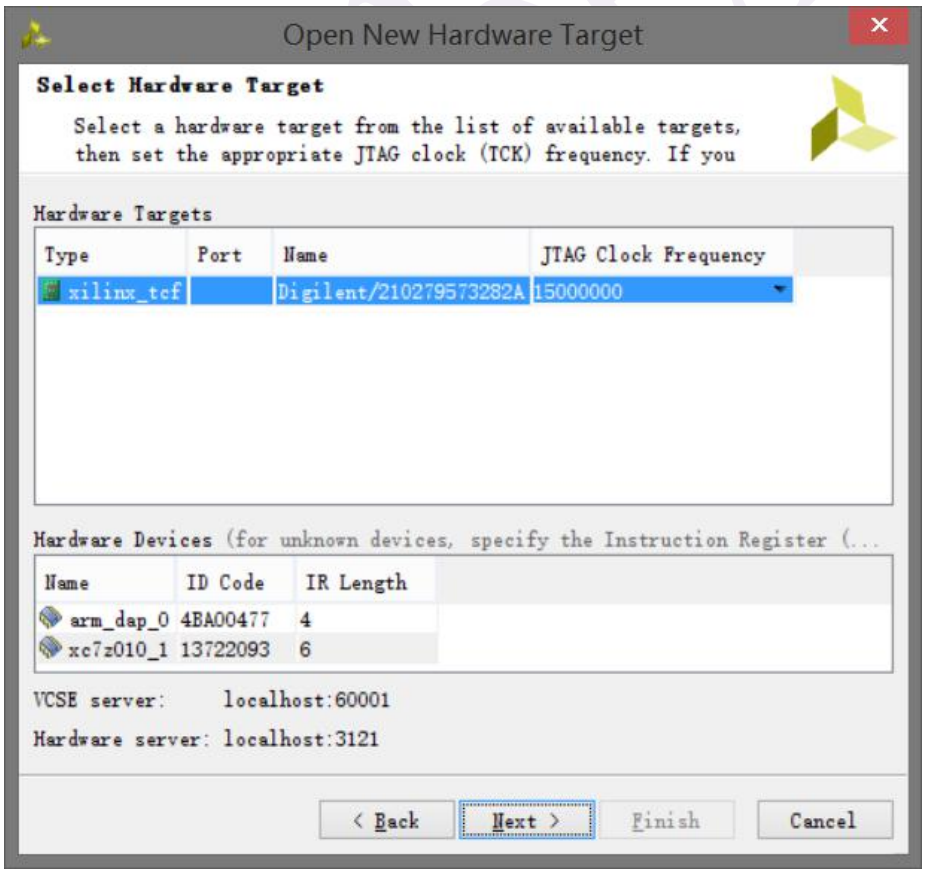


图 4-2-19 新的硬件指标的检测

（10）单击两次 **Next**，然后单击 **Finish**。未连接硬件会话状态更改为服务器名称并且器件被高亮显示，如图 4-2-20 所示。还要注意，该状态表明它还没有被编程；

 Copyright © Robei

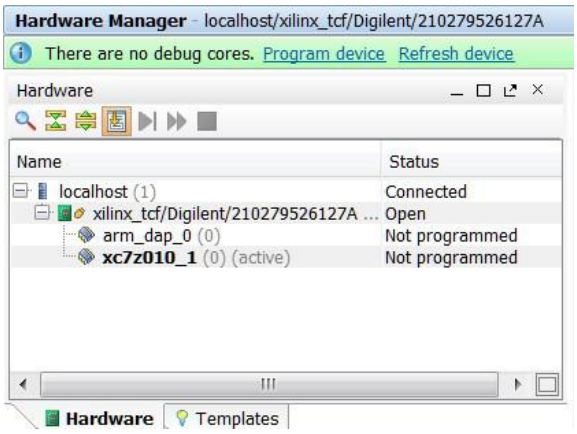


图 4-2-20 打开硬件会话

（11）选择器件，并验证 qadd.bit 被选为常规选项卡中的程序文件；

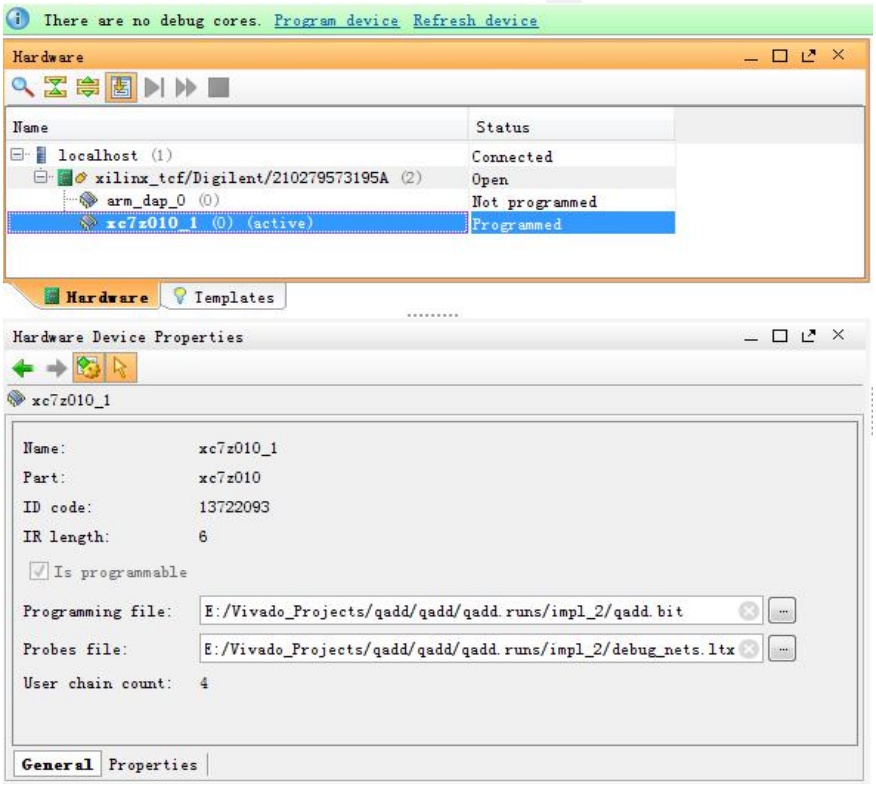


图 4-2-21 编程文件

（12）在器件上单击鼠标右键，选择 Program device 或单击窗口上方弹出的 Program device-> XC7z010_1 链接到目标 FPGA 器件进行编程；

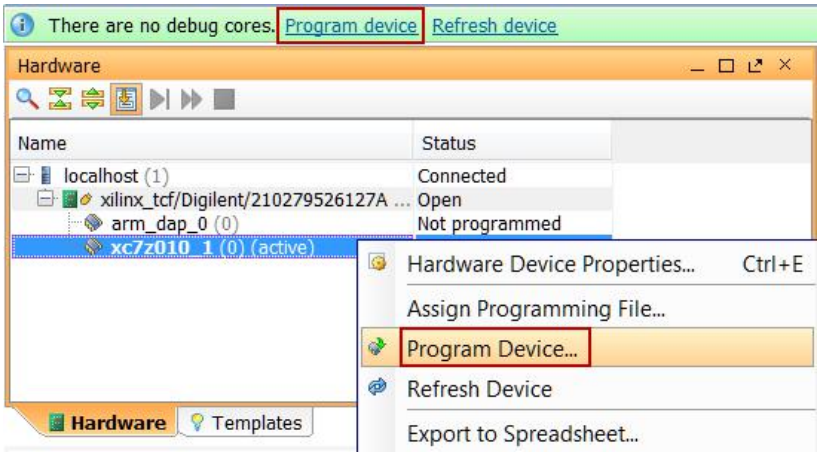


图 4-2-22 选择 FPGA 进行编程

（13）单击确定对 FPGA 进行编程。 开发板上 Done 指示灯亮时，器件编程结束；
（14）通过控制拨动和按键开关的开闭来观察 LED（请参考前面的逻辑图）验证输出结果。

2. 开发板验证

如图 4-2-23 所示，这里使用了一个焊接着开关和 LED 灯的开关板电路提供额外的硬件资源，通过杜邦线将开关板电路上的资源和开发板相连。

本次验证中，使用下排后两个四刀开关分别作为 a 和 b 的值，这里和 Xilinx Z7010 开发板上开关不同之处在于此处的开关拨上为 0，拨下为 1。紧贴开关的那一排 LED 灯的右边四位作为输出 c，点亮表示输出 1，不亮即输出 0。

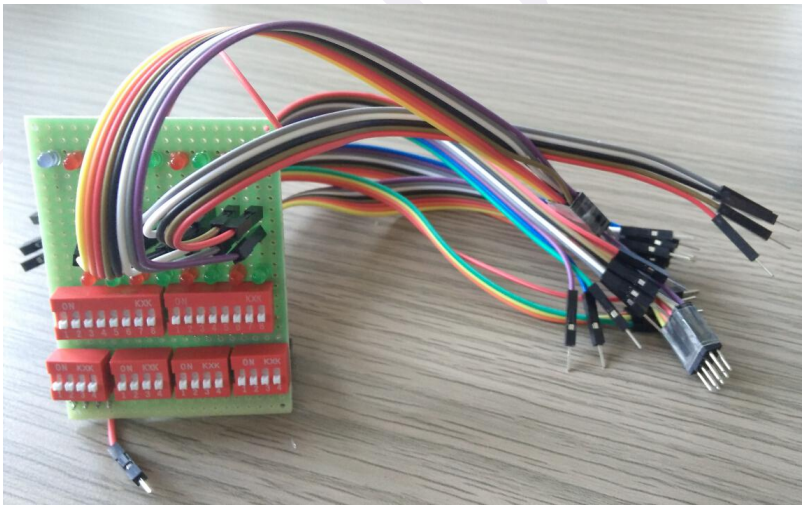
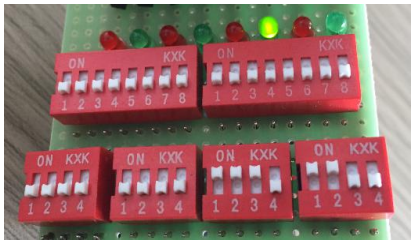


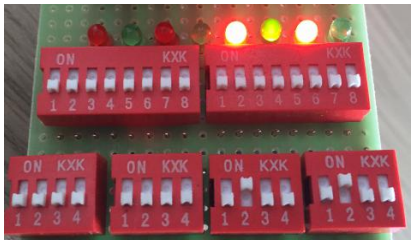
图 4-2-23 开关板电路实图

分别将 a，b 设置为
a = 0001，b = 0011；
a = 0001，b = 0011；
a = 0001，b = 0011；
a = 0001，b = 0011；
然后将得到的结果 c 与预期的结果相比较，经验证，该带符号位加法器运算无误。

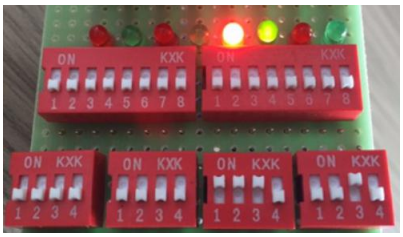
Copyright © Robei



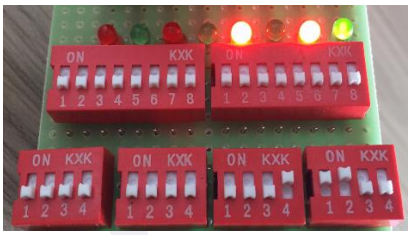
0001 + 0011 = 0100



1011 + 1011 = 1110



0001 + 1101 = 1100



1110 + 0011 = 1011

4.2.4. 问题与思考

1．该模型最大可以计算的位数在十进制下是多少位？
2．如何改进模型，使之有溢出的检测？

4.3 实例九 除法器设计

4.3.1. 本章导读

要求掌握除法器原理，并根据原理设计除法器模块以及设计对应的测试模块，最后在 Robei 可视化仿真软件经行功能实现和仿真验证。

设计原理

这个除法器的设计为传统除法器，因此十分简单，易懂：
（1）先取除数和被除数的正负关系，然后正值化被除数，由于需要递减的除数，所以除数应取负值和补码形式。
（2）被除数每一次递减，商数递增。
（3）直到被除数小于除数，递减过程剩下的是余数。
（4）输出的结果根据除数和被除数的正负关系。
下图 4-3-1 显示除法器模块的设计：

Copyright © Robei

图 4-3-1 除法器模块的设计

4.3.2. 设计流程

1. divider 模型设计

（1）新建一个模型命名为 divider，类型为 module，同时具备 5 输入 3 输出，每个引脚的属性和名称参照下图 4-3-2 经行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	clock
rst_n	input	wire	1	negedge reset
start_sig	input	wire	1	start signal
dividend	input	wire	8	dividend
divisor	input	wire	8	divisor
done_sig	output	wire	1	done signal
quotient	output	wire	8	quotient
remainder	output	wire	8	remainder

图 4-3-2 divider 引脚的属性

图 4-3-3 divider 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

代码：

```
reg[3:0] i;
reg[7:0] dsor;
reg[7:0] rd;
reg[7:0] dend;
reg[7:0] qent;
reg isneg;
reg isdone;

always @(posedge clk or negedge rst_n)
if(!rst_n)
    begin
        i<=4'd0;
```

 Copyright © Robei

```
            dend<=8'd0;
            dsor<=8'd0;
            rd<=8'd0;
            qent<=8'd0;
            isneg<=1'b0;
            isdone<=1'b0;
        end
    else if(start_sig)
        case(i)
            0: begin
                dend<=dividend[7]?~dividend+1:dividend;
                dsor<=divisor[7]?~divisor+1:divisor;
                rd<=divisor[7]?divisor:(~divisor+1);
                isneg<=dividend[7]^divisor[7];
                qent<=8'd0;
                i<=i+1;
            end
            1: begin
                if(dend<dsor)
                begin
                    qent<=isneg?(~qent+1):qent;
                    i<=i+1;
                end
                else
                begin
                    dend<=dend+rd;
                    qent<=qent+1;
                end
            end
            2:begin
                isdone<=1'b1;
                i<=i+1;
            end
            3:begin
                isdone<=1'b0;
                i<=4'd0;
            end
        endcase

    assign done_sig=isdone;
    assign quotient=qent;
    assign remainder=dend;
```

（3）保存模型到一个文件夹(文件夹路径不能有空格和中文)中，运行并检查有无错误。

2. divider_test 测试文件的设计

（1）新建一个 5 输入 3 输出的 divider_test 测试文件，将 Module Type 设置为“testbench”，各个引脚配置如图 4-3-4 所示。

Name	Inout	DataType	Datasize
clk	input	reg	1
rst_n	input	reg	1
start_sig	input	reg	1
dividend	input	reg	8
divisor	input	reg	8
done_sig	output	wire	1
quotient	output	wire	8
remainder	output	wire	8

图 4-3-4 divider_test 测试文件引脚的属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现模型，单击该模型并在 divider _test 上添加，并连接引脚，如下图 4-3-5 所示：

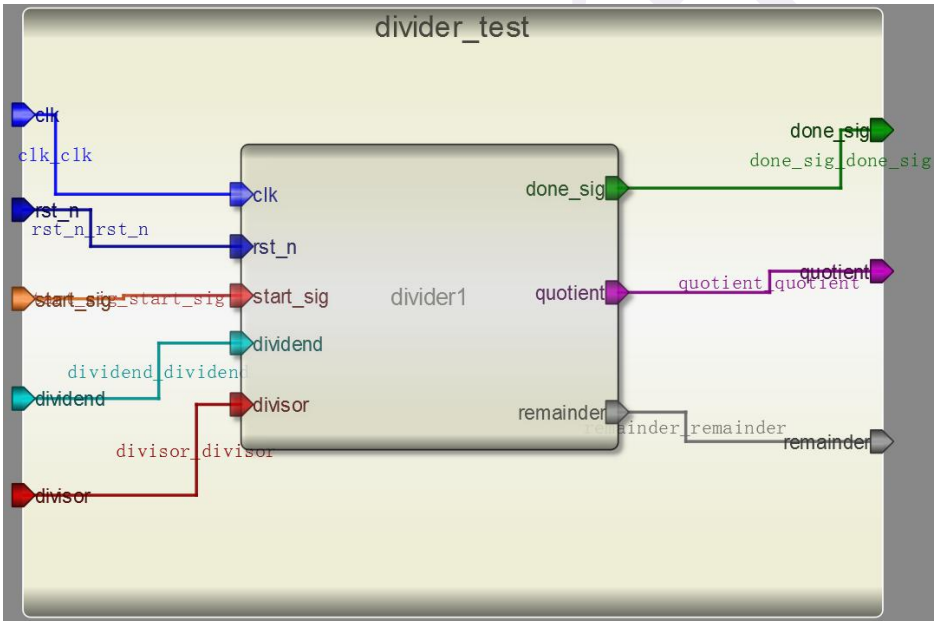


图 4-3-5 divider _test 界面图

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码在结束的时候要用$finish 结束。

测试代码：

```
initial
begin
   clk=1'b0;
   rst_n=1'b0;
   #10 rst_n=1'b1;
   #1000 $finish;
end
```

 Copyright © Robei

```
always #5 clk=~clk;
reg[3:0] i;
always @(posedge clk or negedge rst_n)
begin
    if(!rst_n)
    begin
      dividend<=8'd0;
      divisor<=8'd0;
      start_sig<=1'b0;
      i<=1'b0;
    end
    else
    begin
    case(i)
        0:if(done_sig)
        begin
            start_sig<=1'b0;
            i<=i+1;
        end
        else
        begin
            dividend<=8'd9;
            divisor<=8'd3;
            start_sig<=1'b1;
        end
        1:if(done_sig)
        begin
            start_sig<=1'b0;
            i<=i+1;
        end
        else
        begin
            dividend<=8'd3;
            divisor<=8'd9;
            start_sig<=1'b1;
        end
        2:if(done_sig)
        begin
            start_sig<=1'b0;
            i<=i+1;
        end
        else
        begin
            dividend<=8'd8;
```

Robei

```
                divisor<=8'd2;
                start_sig<=1'b1;
            end
            3:if(done_sig)
            begin
                start_sig<=1'b0;
                i<=i+1;
            end
            else
            begin
                dividend<=8'd8;
                divisor<=8'd4;
                start_sig<=1'b1;
            end
            4:if(done_sig)
            begin
                start_sig<=1'b0;
                i<=i+1;
            end
            else
            begin
                dividend<=8'd8;
                divisor<=8'd3;
                start_sig<=1'b1;
            end
            5:i<=4'd5;
        endcase
        end
    end
```

（5）执行仿真并查看波形。查看输出信息。检查没有错误之后查看波形。

点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 4-3-6 所示：

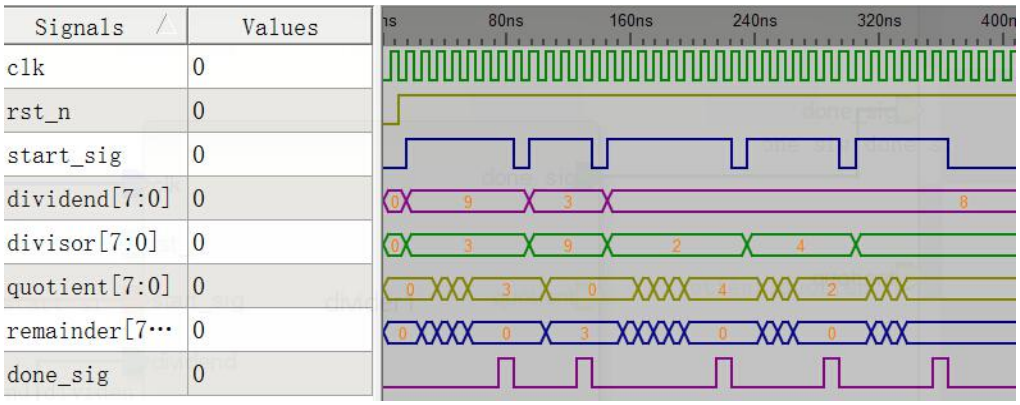


图 4-3-6 divider_test 的仿真波形

3. divider_constrain 约束文件的设计

 Copyright © Robei

由于开发板硬件资源限制，我们在后端设计中把除法器的 dividend 和 divisor 由 8 位改为 4 位，以达到更直观的验证效果。

（1）新建一个模块，命名为 divider_constrain，具备 11 个输入和 9 个输出。

（2）把约束模块保存到和设计的 divider 模块同一个目录下，先打开 divider 模块把 dividend，divisor，quotient，remainder 数据长度都改成 4 位，保存并执行，之后通过鼠标左键单击把 divider 模块添加进约束模块。

（3）连线：分别把 divider 模块的 clk，rst_n，start_sig 和约束模块的一个输入端相连；把 divider 模块的 dividend，divisor 分别和约束模块的四个输入端相连；然后把 divider 模块的输出 done_sig 和约束模块的一个输出端相连，把 quotient，remainder 和约束模块的 4 个输出端相连。

（4）修改约束模块的端口名称和连线名称。约束模块的端口名称即分配到开发板上的硬件引脚。本次设计中使用的引脚如下：

clk 对应开发板开关 G15；

rst_n 对应开发板开关 P15；

start_sig 对应开发板开关 W13；

dividend[0],dividend[1],dividend[2],dividend[3]分别对应 V13，U17，T17，Y17；

divisor[0],divisor[1],divisor[2],divisor[3]分别对应 V12，W16，J15，H15；

done_sig 对应开发板 LED 灯 M14；

quotient[0],quotient[1],quotient[2],quotient[3]分别对应 U14，U15，V17，V18；

remainder[0],remainder[1],remainder[2],remainder[3]分别对应 T14，T15，P14，R14；

修改完端口名称后，记得修改对应 divident，divisor，quotient，remainder 的连线名称分别为 0,1,2,3。完成修改后的模块图如图 4-3-7 所示。

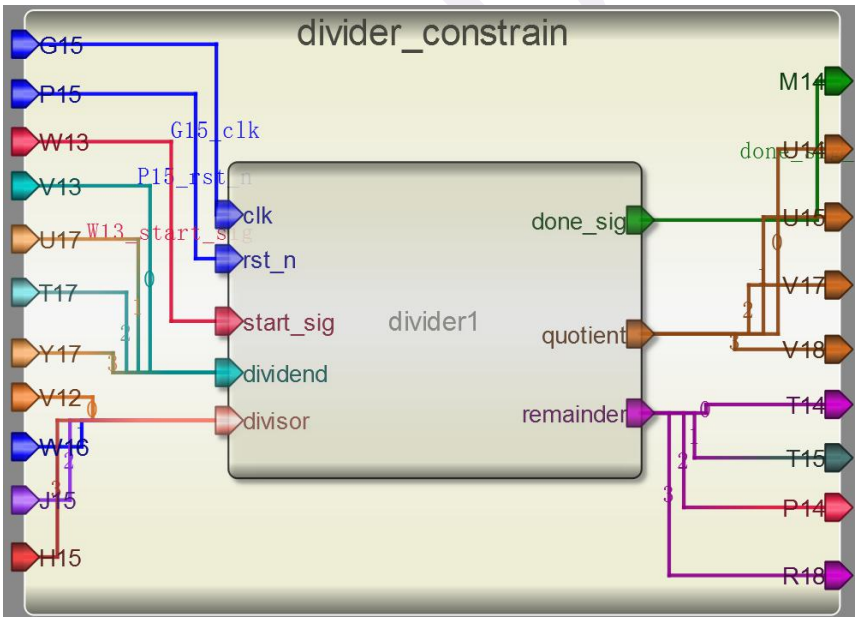


图 4-3-7 约束模块设计

（5）保存并运行，如果看到显示“Generate constrain file complete”字样，说明约束文件已经成功生成，同样也可以点击 View->CodeView 来查看代码。

4.3.3. 板级验证

为了测试所设计 divider 的工作特性，需选择搭载 XILINX 公司的 Z-7010 芯片的开发板，所以选用 VIVADO 设计平台进行 Synthesis、Implementation 和 Generate Bitstream，最终将生成的数据流文件下载到开发板内，并进行验证。

1. VIVADO 设计平台进行后端设计

1.1 启动 Vivado 软件并选择设备 XC7Z010CLG400-1 作为硬件对象，设计语言选用 Verilog，建立新的工程，添加通过 Robei 设计的文件 divider.v。

（1）打开 Vivado，选择开始>所有程序>Xilinx Design Tools> Vivado2013.4> Vivado2013.4；

（2）单击创建新项目 Create New Project 启动向导。你将看到创建一个新的 Vivado 项目对话框，单击 Next；

（3）在弹出的对话框中输入工程名 divider 及工程保存的位置，并确保 Create project subdirectory 复选框被选中，单击 Next；

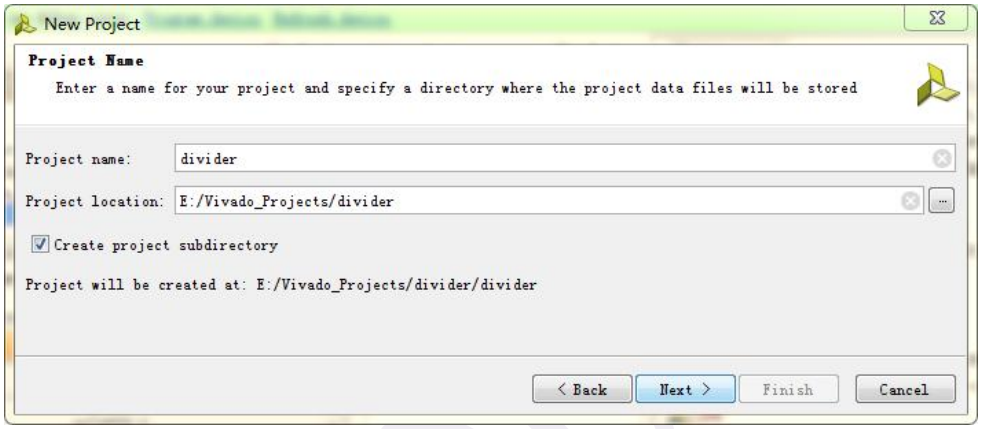


图 4-3-11 项目名称和位置输入

（4）选择项目类型表单的 RTL Project 选项，不要勾选 Do not specify sources at this time 复选框，然后单击 Next；

（5）使用下拉按钮，选中 Verilog 作为目标文件和仿真语言；

（6）点击添加 Add Files 按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 divider.v，单击 Open，然后单击 Next 去添加现有的 IP 模型；

（7）由于我们没有任何的 IP 添加，跳过这一步，单击 Next 去添加约束形成；

（8）点击添加 Add Files 按钮，浏览到刚刚设计的目录下找到 constrain 文件夹，选择其中的 divider_constrain.xdc 文件，单击 open 完成添加；

（9）在默认窗口中，按照图 4-3-13 所示设置 Filer 中的选项，然后在 Parts 中选择 XC7Z010CLG400-1，单击 Next；

（10）单击 Finish，本 Vivado 项目创建成功。

 Copyright © Robei

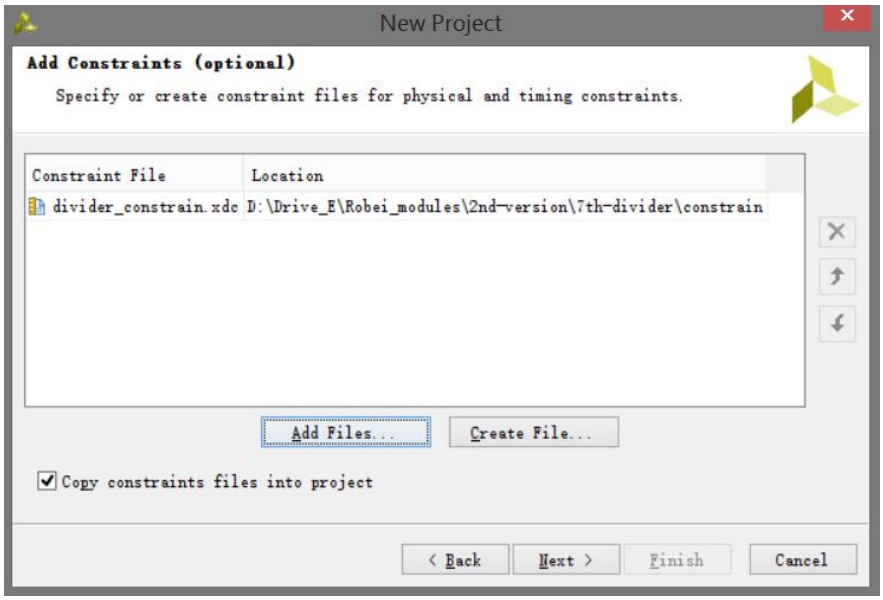


图 4-3-12 添加约束文件

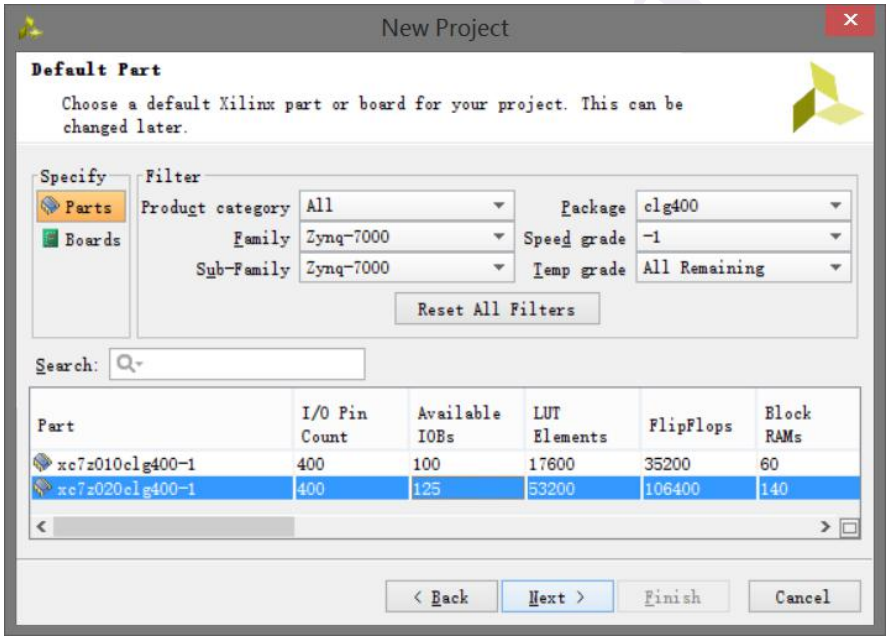


图 4-3-13 器件选型

1.2 打开 divider_constrain.xdc 文件，可以查看引脚约束源代码。

（1）在资源窗口 sources 中，展开约束文件夹，然后双击打开 divider_constrain.xdc 进入文本编辑模式；

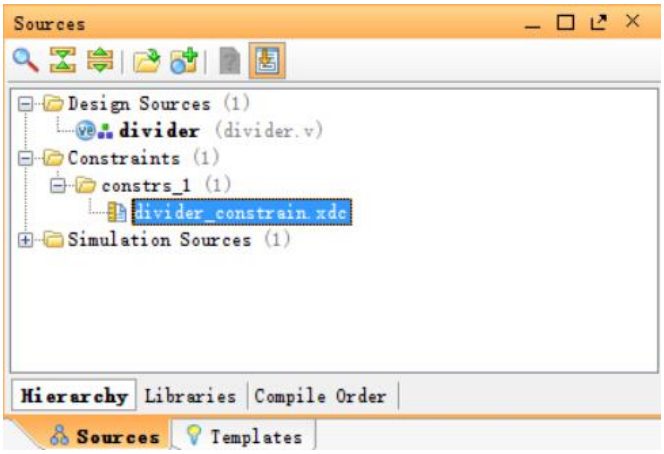


图 4-3-14 展开约束文件 divider_constrain.xdc

（2）Xilinx 设计约束文件分配 FPGA 位于主板上的开关和指示灯的物理 IO 地址，这些信息可以通过主板的原理图或电路板的用户手册来获得。

本次设计的约束文件代码是通过 Robei 软件自动生成，但是，Robei 软件目前生成的约束代码只有对输入输出端口的分配，在这个设计中，我们使用了一个通过开关控制的模拟时钟 clk，而非系统时钟，这种电路在综合的时候一般都会报错，所以，在约束文件最后，我们需要手动添加一句命令：

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

来保证流程中不会出错。

完整的约束代码如下：

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN G15 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN P15 [get_ports rst_n]
set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
set_property PACKAGE_PIN M14 [get_ports done_sig]
set_property IOSTANDARD LVCMOS33 [get_ports done_sig]
set_property PACKAGE_PIN V12 [get_ports divisor[0]]
set_property IOSTANDARD LVCMOS33 [get_ports divisor[0]]
set_property PACKAGE_PIN W16 [get_ports divisor[1]]
set_property IOSTANDARD LVCMOS33 [get_ports divisor[1]]
set_property PACKAGE_PIN J15 [get_ports divisor[2]]
set_property IOSTANDARD LVCMOS33 [get_ports divisor[2]]
set_property PACKAGE_PIN H15 [get_ports divisor[3]]
set_property IOSTANDARD LVCMOS33 [get_ports divisor[3]]
set_property PACKAGE_PIN V13 [get_ports dividend[0]]
set_property IOSTANDARD LVCMOS33 [get_ports dividend[0]]
set_property PACKAGE_PIN U17 [get_ports dividend[1]]
set_property IOSTANDARD LVCMOS33 [get_ports dividend[1]]
set_property PACKAGE_PIN T17 [get_ports dividend[2]]
set_property IOSTANDARD LVCMOS33 [get_ports dividend[2]]
set_property PACKAGE_PIN Y17 [get_ports dividend[3]]
```

 Copyright © Robei

```
set_property IOSTANDARD LVCMOS33 [get_ports dividend[3]]
set_property PACKAGE_PIN T14 [get_ports {remainder[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports remainder[0]]
set_property PACKAGE_PIN T15 [get_ports remainder[1]]
set_property IOSTANDARD LVCMOS33 [get_ports remainder[1]]
set_property PACKAGE_PIN P14 [get_ports remainder[2]]
set_property IOSTANDARD LVCMOS33 [get_ports remainder[2]]
set_property PACKAGE_PIN R14 [get_ports remainder[3]]
set_property IOSTANDARD LVCMOS33 [get_ports remainder[3]]
set_property PACKAGE_PIN U14 [get_ports quotient[0]]
set_property IOSTANDARD LVCMOS33 [get_ports quotient[0]]
set_property PACKAGE_PIN U15 [get_ports quotient[1]]
set_property IOSTANDARD LVCMOS33 [get_ports quotient[1]]
set_property PACKAGE_PIN V17 [get_ports quotient[2]]
set_property IOSTANDARD LVCMOS33 [get_ports quotient[2]]
set_property PACKAGE_PIN V18 [get_ports quotient[3]]
set_property IOSTANDARD LVCMOS33 [get_ports quotient[3]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

（3）点击 File—>Save File 保存文件。

1.3 使用 Vivado 综合工具来综合设计，并进行 Implementation 任务实现。

（1）单击综合任务下拉菜单中的 Run Synthesis，综合过程将会在 divider.v 文件以及所有分层文件中运行。当综合过程完成了，且没有错误信息，将会弹出带有三个选项的完成对话框；

（2）如果有错误，根据错误信息提示修改，直至综合没有错误。然后选择 Run Implementation 选项，执行任务实现，然后单击 OK；

（3）任务实现过程将在综合后的设计上运行。当这个过程完成，且没有错误信息，将会弹出带有三个选项的任务实现完成对话框；

（4）如果有错误，根据错误信息提示修改，直至综合没有错误。

1.4 将开发板上的电源开关拨到 ON，生成比特流并打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口（在电源开关的旁边）；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击任务实现完成弹出的对话框中 Generate Bitstream 或者点击导航窗口中编程和调试任务中的 Generate Bitstream。比特流生成过程将在任务实现设计后运行。当完成比特流生成后会弹出有三个选项的完成对话框；

（5）这一过程将已经生成的 divider.bit 文件放在 divider.runs 目录下的 impl_1 目录下；

（6）选择打开硬件管理器 Open Hardware Manager 选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

（7）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target；

（8）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（9）单击 Next 以选择本地主机端口；

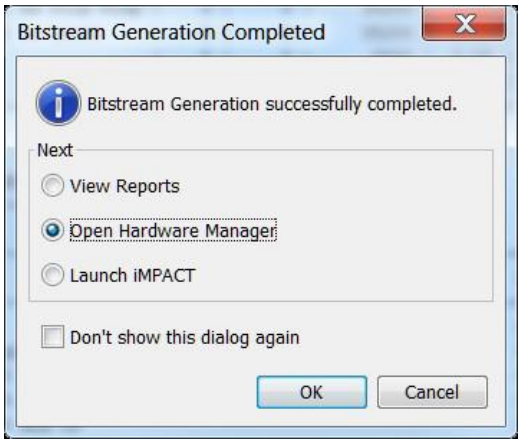


图 4-3-16 比特流生成

图 4-3-17 打开新的硬件目标

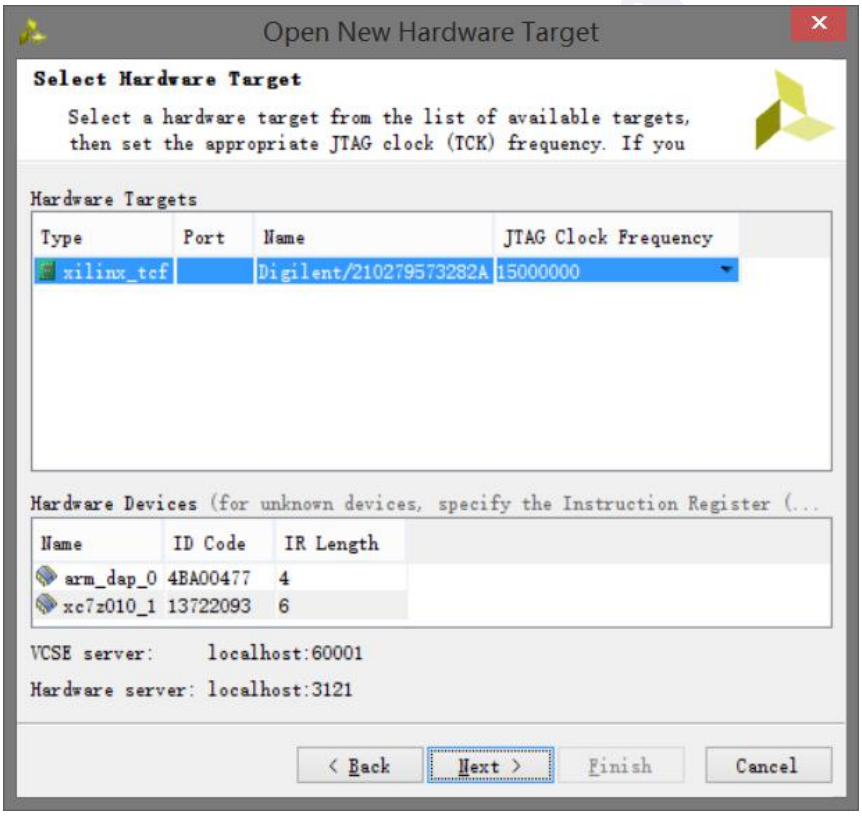


图 4-3-18 新的硬件指标的检测

（10）单击两次 Next，然后单击 Finish。未连接硬件会话状态更改为服务器名称并且器件被高亮显示，如图 4-3-19 所示。还要注意，该状态表明它还没有被编程；

 Copyright © Robei

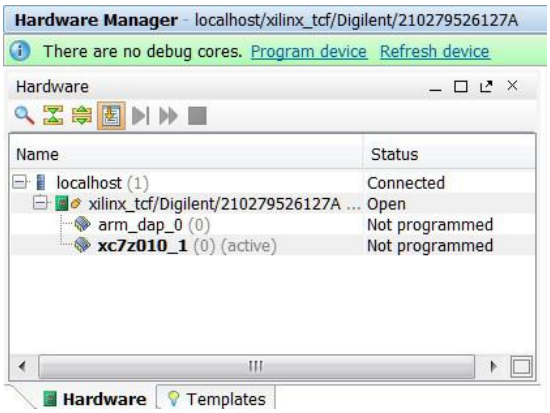


图 4-3-19 打开硬件会话

（11）在器件上单击鼠标右键，选择 Program device 或单击窗口上方弹出的 Program device-> XC7z010_1 链接到目标 FPGA 器件进行编程；

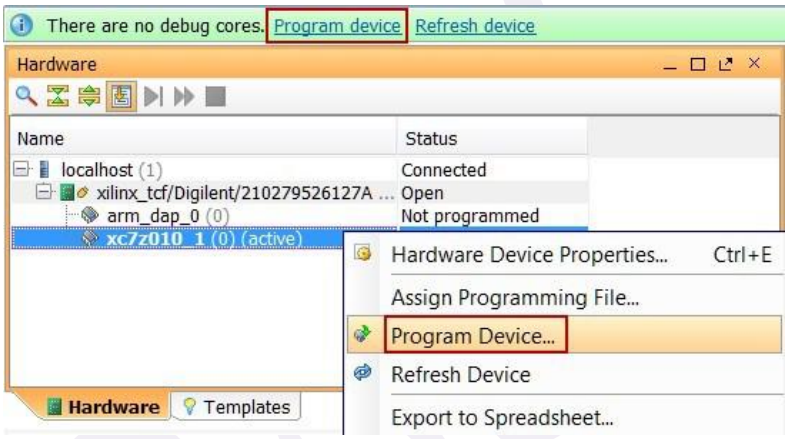


图 4-3-20 选择 FPGA 进行编程

（12）单击确定对 FPGA 进行编程。开发板上 Done 指示灯亮时，器件编程结束；
（13）通过控制拨动和按键开关的开闭来观察 LED（请参考前面的逻辑图）验证输出结果。

2. 开发板验证

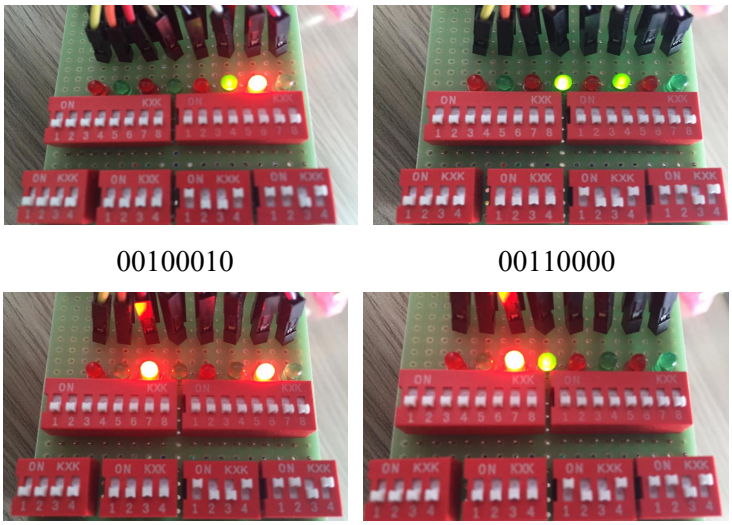
首先，将 rst_n(SW1)复位开关拨之低电平，再来回拨动 1 至 2 次 clk(BTNC)按键，进行复位操作，之后将 rst_n 拨至高电平；

然后，设置被除数和除数值，这里选择被除数为 6，除数为 2 进行验证；

除数 SW7~4，被除数 SW3~0，商 LD3~0，余数 LD7~4。

最后，每次按一下 clk 时钟键，给一个上升沿，程序进行一次运算。第一次上升沿后，quotient 为 0，reminder 为 6，故 LD7～0 的显示为 00000110；第二次上升沿后，quotient 为 1，reminder 为 4，故 LD7～0 的显示为 00010100；第三次上升沿后，quotient 为 2，reminder 为 2，故 LD7～0 的显示为 00100010；第四次上升沿后，quotient 为 3，reminder 为 0，故 LD7～0 的显示为 00110000，此时计算结束，done_sig 亮起，经验证该除法器本次运算无误。

00000110 00010100



00100010 00110000

图 4-3-22 每一个 clk 时钟的 LED 灯显示实图

重复以上步骤，设置不同的被除数和除数值进行多方位验证，通过验证结果证明该除法器符合设计的要求。

4.3.4. 问题与思考

挑战题：读者们可能发现以上设计的除法器每一次运算消耗的时钟不一致，当除数的数量越大，它消耗的时钟的就越大，所以下面请读者们上网搜索循环式除法器的资料，并以循环式除法器的算法设计一个运算过程消耗时钟一致的循环式除法器。

Copyright © Robei

第五章：认识协议，操作接口

今天我们来尝试设计一些电路中常见的协议和接口。其实协议和接口的概念看起来可能比较抽象，但实际上往往都只是一段代码，一个模块等。因为篇幅有限我们难以介绍到全部常见和常用的协议接口，希望读者们如果时间允许，可以自己查找其他的资料进行更多协议与接口的学习。相关的资料可以在 Robei 的官网上进行搜索，下载之后可以自行尝试。

Robei

5.1 实例十 FIFO

5.1.1. 本章导读

FIFO（First in First out）使用在需要产生数据接口的部分，用来存储、缓冲在两个异步时钟之间的数据传输。在异步电路中，由于时钟之间周期和相位完全独立，因此数据丢失概率不为零。使用 FIFO 可以在两个不同时钟域系统之间快速而方便地传输实时数据。这次的设计我们就来学习一下如何用 Robei 和 Verilog 设计一个 8 位 8 深度的 FIFO，并通过 VIVADO 设计平台进行板级验证。

设计准备

FIFO 的原理框图如下图 5-1-1 所示：

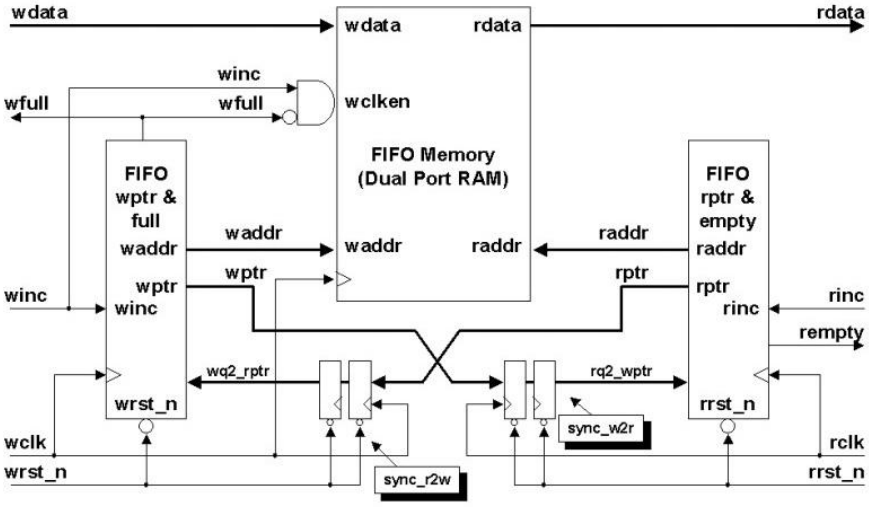


图 5-1-1 fifo 原理框图

通过分析，我们看到图 5-1-1 中有一个具有独立的读端口和独立的写端口的 RAM 存储器。这样选择是为了分析方便。如果是一个单端口的存储器，还应包含一个仲裁器，保证同一时刻只能进行一项操作（读或写），我们选择双口 RAM（无需真正的双口 RAM，因为我们只是希望有一个简单的相互独立的读写端口）是因为这些实例非常接近实际情况。读、写端口拥有两个计数器（wr_ptr、rd_ptr）产生的互相独立的读、写地址。计数器的值在读写使能信号来临时传递给“读指针”（rd）和“写指针”（wr）。写指针指向下一个将要写入的位置，读指针指向下一个将要读取的位置。每次写操作使写指针加 1，读操作使读指针加 1。左右两侧的模块为读写指针与满空信号产生模块。这两个模块的任务是给 FIFO 提供“空”（empty）和“满”（full）信号。这些信号告诉外部电路 FIFO 已经达到了临界条件：如果出现“满”信号，那么 FIFO 为写操作的临界状态，如果出现“空”信号，则 FIFO 为读操作的临界状态。写操作的临界状态表示 FIFO 已经没有空间来存储更多的数据，读操作的临界表示 FIFO 没有更多的数据可以读出。读写指针与满空信号产生模块还可告诉 FIFO 中“满”或“空”位置的数值。这是由指针的算术运算来完成了。实际的“满”或“空”位置计算并不是为 FIFO 自身提供的。它是作为一个报告机构给外部电路用的。但是，“满”和“空”信号在 FIFO 中却扮演着非常重要的角色，它为了能实现读与写操作各自的独立运行而阻塞性的管理数据的存取。这种阻塞性管理的重要性不是将数据复写（或重读），而是指针位置

 Copyright © Robei

可以控制整个 FIFO，并且使读、写操作改变着指针数值。如果我们不阻止指针在临界状态下改变状态，FIFO 还能都一边“吃”着数据一边“产生”数据，这简直是不可能的。

从功能上看，FIFO 的工作原理如下所述：复位时，读、写指针均为 0。这是 FIFO 的空状态，空标志（empty）为高电平，此时满标志（full）为低电平。当 FIFO 出现空标志（empty）时，不允许读操作，只能允许写操作。写操作写入到位置 0，并使写指针加 1。此时，空标志（empty）变为低电平。假设没有发生读操作且随后的一段时间中 FIFO 只有写操作，一定时间后，写指针的值等于 7。这就意味着在存储器中，要写入数据的最后一个位置就是下一个位置。在这种情况下，写操作将写指针变为 0，并将输出满标志（full）。

为了更好地判断空状态和满状态，这里设置一个四位的计数器（fifo_cnt），代表存储器（mem）中写入但还未读取的数据个数。当 FIFO 未进行任何读写操作时，计数器保持不变；当进行写操作时，计数器加 1；当进行读操作时，计数器减 1；当同时进行写操作和读操作时，计数器值保持不变。这样就可以根据计数器中的值来判断状态的空与满，即：当计时器 fifo_cnt=0 时，表示存储器处于空状态，输出空标志（empty）；当计数器 fifo_cnt=8 时，表示存储器处于满状态，输出满标志（full）。

设计要求

读写指针都指向一个内存的初始位置，每进行一次读写操作，相应的指针就递增一次，指向下一个内存位置。当指针移动到了内存的最后一个位置时，它又重新跳回初始位置。在 FIFO 非满或非空的情况下，这个过程将随着读写控制信号的变化一直进行下去。如果 FIFO 处于空的状态，下一个读动作将会导致向下溢(underflow)，一个无效的数据被读入；同样，对于一个满了的 FIFO，进行写动作将会导致向上溢出(overflow)，一个有用的数据被新写入的数据覆盖。这两种情况都属于误动作，因此需要设置满和空两个信号，对满信号置位表示 FIFO 处于满状态，对满信号复位表示 FIFO 非满，还有空间可以写入数据；对空信号置位表示 FIFO 处于空状态，对空信号复位表示 FIFO 非空，还有有效的数据可以读出。设计波形如图 5-1-2 所示。

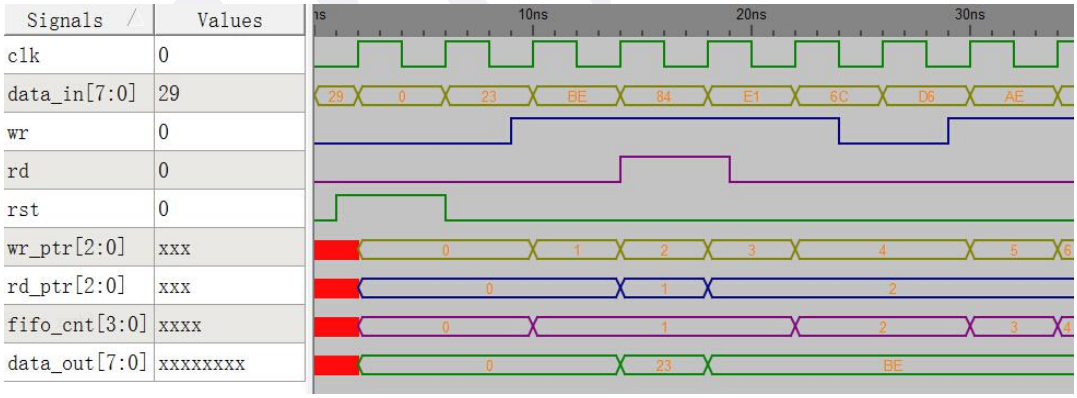


图 5-1-2　fifo 输出波形要求

5.1.2. 设计流程

1. 模型设计

（1）新建一个模型。点击软件工具栏上面的图标 ，或者点击“File”下拉菜单中的“New”，会有一个对话框弹出来（如图 5-1-3 所示）。在弹出的对话框中设置你所设计的模型。

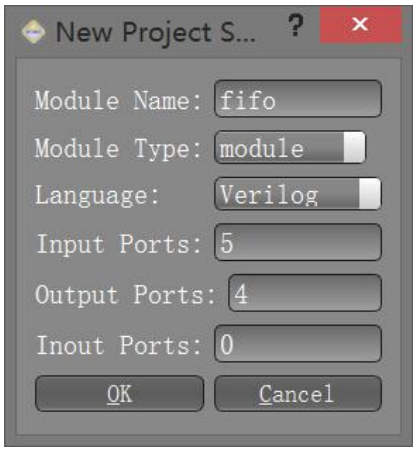


图 5-1-3 新建一个项目

参数填写完成后点击“OK”按钮，Robei 就会生成一个新的模块，名字就是 fifo，如图 5-1-4 所示：

图 5-1-4 fifo 界面图

（2）修改模型。在自动生成的界面图上进行名称的修改，输入引脚为 clk，rst，wr，rd 和 data_in，输出引脚修改成 full，empty，data_out。为了区分每个引脚，我们可以修改每个引脚的 Color 值，并点回车保存。如果选中模块，按“F1”键，就会自动生成一个 Datasheet，如图 5-1-5 所示。

Name	Inout	DataType	Datasize
clk	input	wire	1
rst	input	wire	1
data_in	input	wire	8
wr	input	wire	1
rd	input	wire	1
full	output	wire	1
empty	output	wire	1
data_out	output	reg	8
fifo_cnt	output	reg	4

图 5-1-5 Datasheet

Copyright © Robei

（3）输入算法。点击模型下方的 Code，进入代码设计区。

在代码设计区内输入以下 Verilog 代码：

```
reg [7:0] fifo_ram[0:7];
reg [2:0] rd_ptr, wr_ptr;
assign empty = (fifo_cnt==0);
assign full = (fifo_cnt==8);
always @( posedge clk )
begin: write
    if(wr && !full)
        fifo_ram[wr_ptr] <= data_in;
    else if(wr && rd)
        fifo_ram[wr_ptr] <= data_in;
end

always @( posedge clk )
begin: read
    if( rst )
        data_out<=0;
    else
    begin
        if(rd && !empty)
            data_out <= fifo_ram[rd_ptr];
        else if(rd && wr)
            data_out <= fifo_ram[rd_ptr];
    end
end

always @( posedge clk )
begin: pointer
    if( rst )
    begin
        wr_ptr <= 0;
        rd_ptr <= 0;
    end
    else
    begin
        wr_ptr <= ((wr && !full) || (wr && rd)) ? wr_ptr+1 : wr_ptr;
        rd_ptr <= ((rd && !empty) || (wr && rd)) ? rd_ptr+1 : rd_ptr;
    end
end

always @( posedge clk )
begin: count
    if( rst )
```

Robei

```
            fifo_cnt <= 0;
        else
        begin
            case ({wr,rd})
                2'b00 : fifo_cnt <= fifo_cnt;
                2'b01 : fifo_cnt <= (fifo_cnt==0) ? 0 : fifo_cnt-1;
                2'b10 : fifo_cnt <= (fifo_cnt==8) ? 8 : fifo_cnt+1;
                2'b11 : fifo_cnt<=fifo_cnt;
                default: fifo_cnt <= fifo_cnt;
            endcase
        end
    end
```

（4）保存。点击工具栏 图标，或者点击菜单“File”中的下拉菜单“Save as”，将模型保存到一个文件夹中。（文件夹路径不能有中文或空格）

（5）运行。在工具栏点击 或者点击菜单“Build”的下拉菜单“Run”，执行代码检查，如有错误，会在输出窗口中显示。如果没有错误提示，则模型 fifo 设计完成。

2. 测试模块设计

（1）新建一个模块。点击工具栏上的 图标，在弹出的对话框中参照图 5-1-8 进行设计。

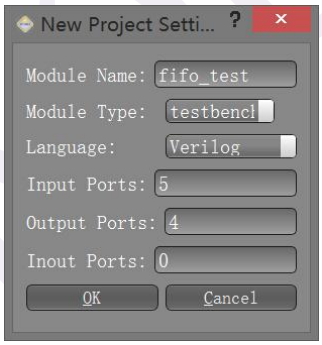


图 5-1-6 新建测试文件

Name	Inout	DataType	Datasize
clk	input	reg	1
rst	input	reg	1
data_in	input	reg	8
wr	input	reg	1
rd	input	reg	1
full	output	wire	1
empty	output	wire	1
data_out	output	wire	8
fifo_cnt	output	wire	4

图 5-1-7 引脚属性表

Copyright © Robei

（2）修改各个引脚的属性。选中每个引脚，在属性栏中对照图 5-1-7 进行修改引脚属性，同时可以修改其颜色，方便区分不同的引脚信号。

（3）另存为测试文件。点击工具栏 图标，将测试文件保存到 fifo 模型所在的文件夹下。

（4）加入模型。在 Toolbox 工具箱的 Current 栏里，会出现之前设计的 fifo 模型，单击该模型并在 fifo_test 上添加。

（5） 连接引脚。 点击工具栏中的 图标， 或者选择菜单“Tool”中的“connect”，如图 5-1-8 所示连接引脚。完成连接的模块如图 5-1-8 所示。这个时候，注意查看连接线的颜色。如果鼠标要变回选择模式，点击图标 。

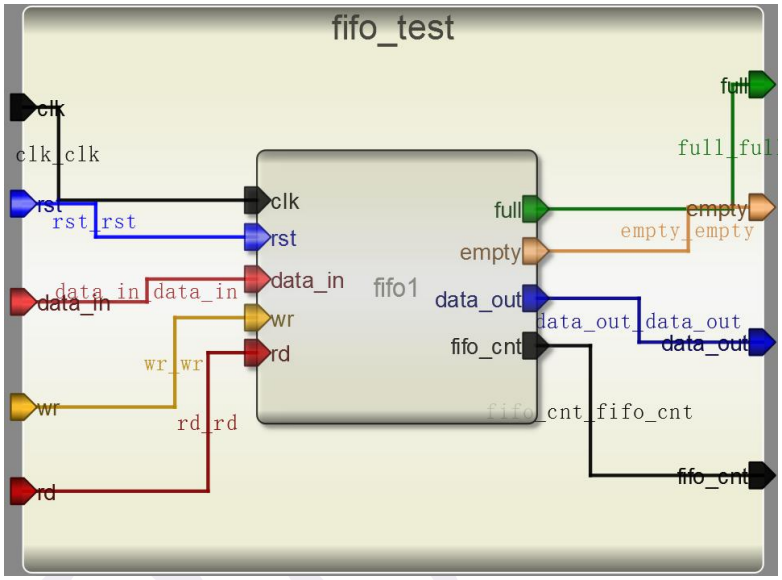


图 5-1-8　连接引脚后的测试模块

（6）输入激励。点击模块下方的“Code”，输入激励算法。激励代码在结束的时候要记得用$finish 结束。

激励代码：

```
initial begin
    rst=0;
    clk=0;
    wr=0;
    rd=0;
    data_in=0;
    #1 rst=1;
    #5 rst=0;
    #3 wr=1;
    #5 rd=1;
    #5 rd=0;
    #5 wr=0;
    #5 wr=1;
    #10 rd=1;
    #10 rd=0;
```

Copyright © Robei

```
        #14 $finish;
    end
    always
    begin
        #2 clk=~clk;
    end
    always @(posedge clk or negedge rst)
    begin
        if (rst)
        begin
            data_in<=0;
            wr<=0;
            rd<=0;
        end
        else
            data_in<=$random;
    end
```

（7）执行仿真并查看波形。点击工具栏 ，查看输出信息，检查没有错误之后点击 进行仿真，再点击 或者菜单“View”中的“Waveview”，波形查看器就会打开。点击右侧 Workspace 中的信号，进行添加并查看。点击波形查看器工具栏上的 图标进行自动缩放。如图 5-1-9 所示。分析仿真结果并对照真值表，查看设计波形与设计要求是否一致。

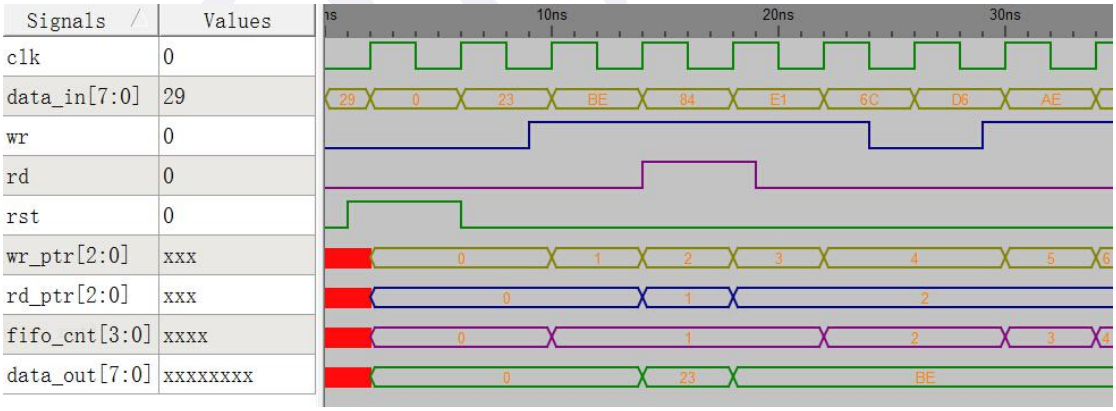


图 5-1-9 查看波形

3. 约束模块设计

在设计约束文件之前，我们还是要考虑到开发板硬件资源的限制，需要把设计中数据的位数修改为 4 位。

（1）打开之前设计的 fifo 模块，把输入 data_in，输出 data_out 的长度修改为 4 位，保存并执行。执行后模块的 Verilog 代码中数据长度也会被更新，便于之后的操作。

（2）新建一个模块，命名为 fifo_constrain，模块类型选择为 constrain，并将其保存到和之前设计的 fifo 模块的同一目录下。

Copyright © Robei

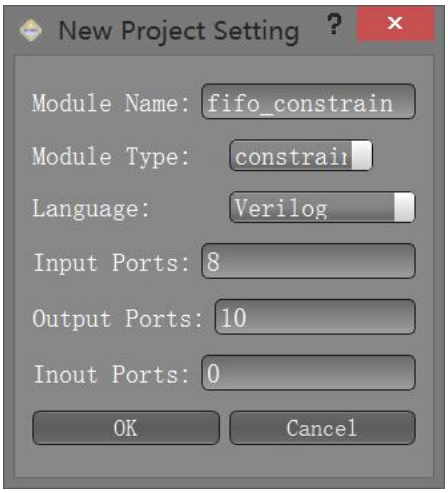


图 5-1-10 新建约束模块

（3）连线并修改端口名称。其中的 data_in，data_out，fifo_cnt 需要对应 4 个约束模块的端口，并且 4 根连线的名称分别为 0,1,2,3。本设计中使用的开发板引脚如下：

clk 对应开发板上按键开关 R18；

rst 对应开发板上按键开关 P16；

data_in[0],data_in[1],data_in[2],data_in[3]对应拨码开关 G15，P15，W13，T16；

wr 对应开发板上按键开关 Y16；

rd 对应开发板上按键开关 V16；

data_out[0],data_out[1],data_out[2],data_out[3]对应 LED 灯 M14，M15，G14，D18；

fifo_cnt[0],fifo_cnt[1],fifo_cnt[2],fifo_cnt[3]对应 U17，V13，H15，J15；

full 和 empty 分别对应 W16，V12；

完成后的约束模块如图 5-1-11 所示。

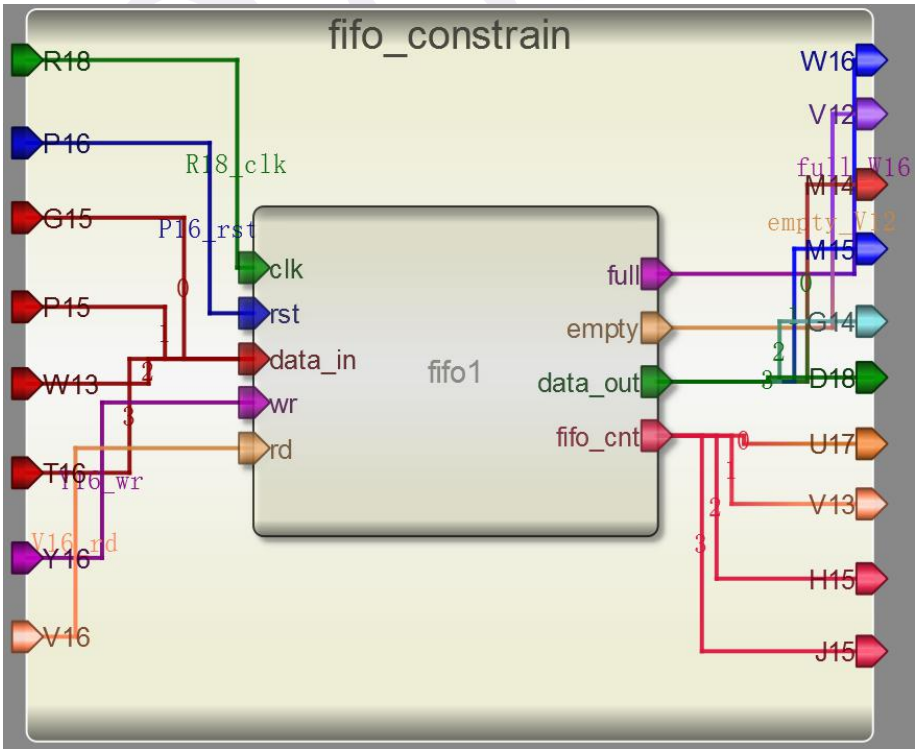


图 5-1-11 约束模块的设计

（4）保存并执行，如果软件显示“Generate constrain file complete”，则说明已经成功生成 fifo_constrain.xdc 文件。

5.1.3. 板级验证

为了测试所设计 FIFO 的工作特性，选择搭载 XILINX 公司的 Z-7010 芯片的开发板，所以选用 VIVADO 设计平台进行 Synthesis、Implementation 和 Generate Bitstream，最终将生成的数据流文件下载到开发板内，并进行验证。

1. VIVADO 设计平台进行后端设计

1.1 启动 Vivado 软件并选择设备 XC7Z010CLG400-1 作为硬件对象，设计语言选用 Verilog，建立新的工程，添加通过 Robei 设计的文件 fifo.v。

（1）打开 Vivado，选择开始>所有程序>Xilinx Design Tools> Vivado2013.4> Vivado2013.4；

（2）单击创建新项目 Create New Project 启动向导。你将看到创建一个新的 Vivado 项目对话框如图 5-1-15 所示，单击 Next；

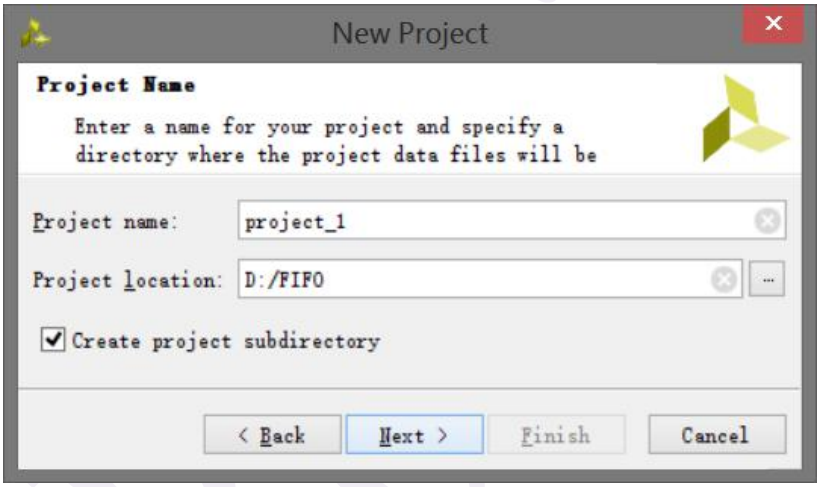


图 5-1-15 项目名称和位置输入

（3）在弹出的对话框中输入工程名 fifo 及工程保存的位置，并确保 Create project subdirectory 复选框被选中，单击 Next；

（4）选择项目类型表单的 RTL Project 选项，不勾选 Do not specify sources at this time 复选框，然后单击 Next；

（5）使用下拉按钮，选中 Verilog 作为目标文件和仿真语言；

（6）点击添加 Add Files 按钮，浏览到刚刚我们 Robei 项目的目录下打开 Verilog 文件夹，选择 fifo.v，单击 Open，然后单击 Next 去添加现有的 IP 模型；

（7）由于我们没有任何的 IP 添加，跳过这一步，单击 Next 去添加约束形成；

（8）点击添加 Add Files 按钮，浏览到刚刚设计 fifo 模块目录下的 constrain 文件夹，选中其中的 fifo_constrain.xdc 文件，单击 Open 进行添加。

（9）在默认窗口中，按照图 5-1-19 所示设置 Filer 中的选项，然后在 Parts 中选择 XC7Z010CLG400-1，单击 Next；

（10）单击 Finish，本 Vivado 项目创建成功。

Copyright © Robei

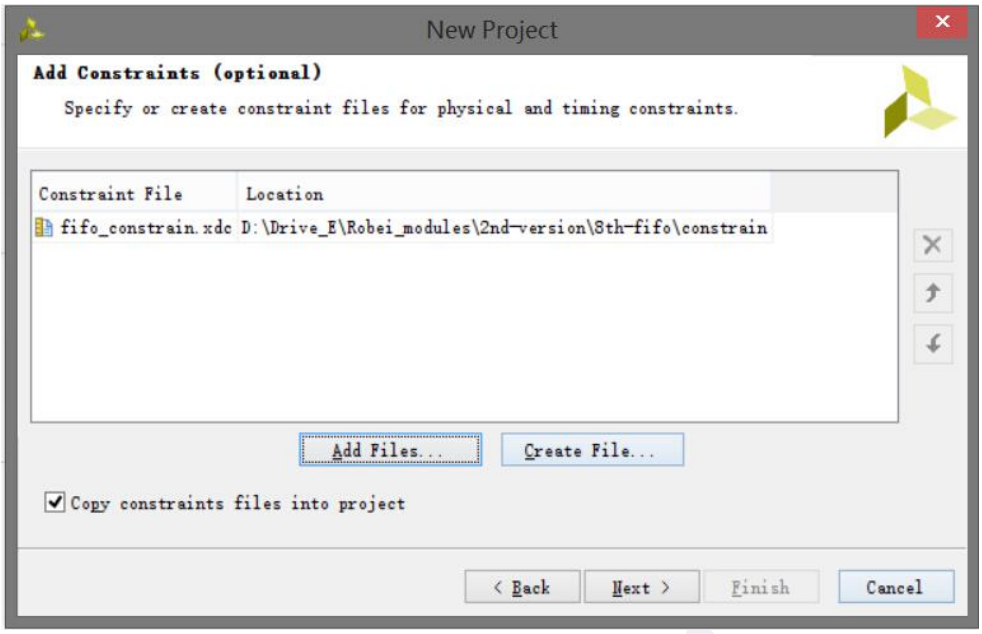


图 5-1-16 添加约束文件

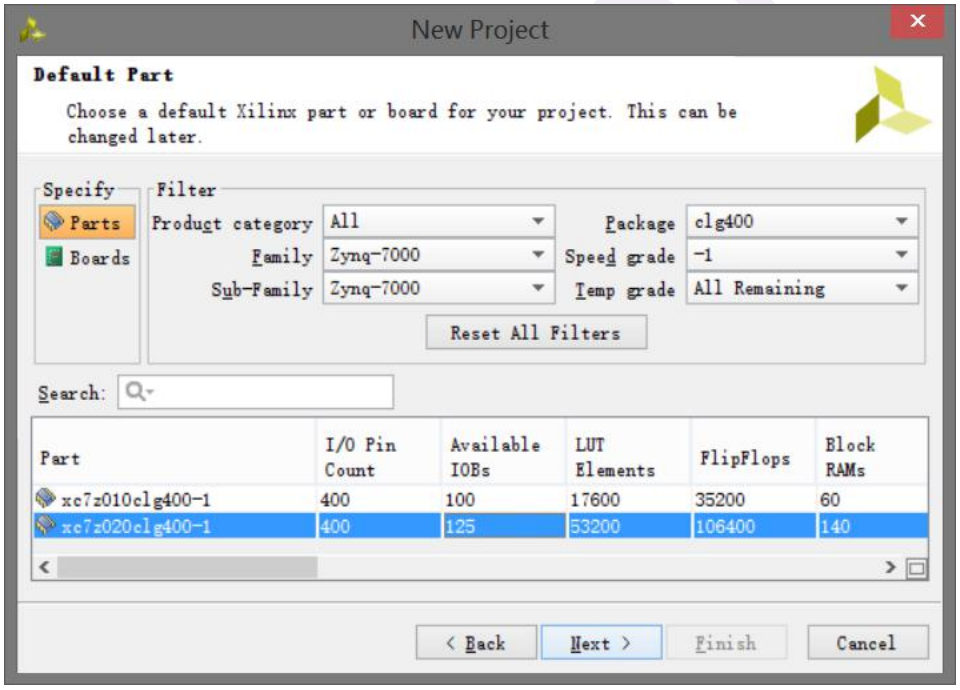


图 5-1-17 器件选型

1.2 打开 fifo_constrain.xdc 文件，查看引脚约束源代码。

（1）在资源窗口 sources 中，可以通过双击查看设计文件 fifo.v。

（2）在资源窗口 sources 中，展开约束文件夹，如图 5-1-18 所示，然后双击打开 fifo_constrain.xdc 进入文本编辑模式；

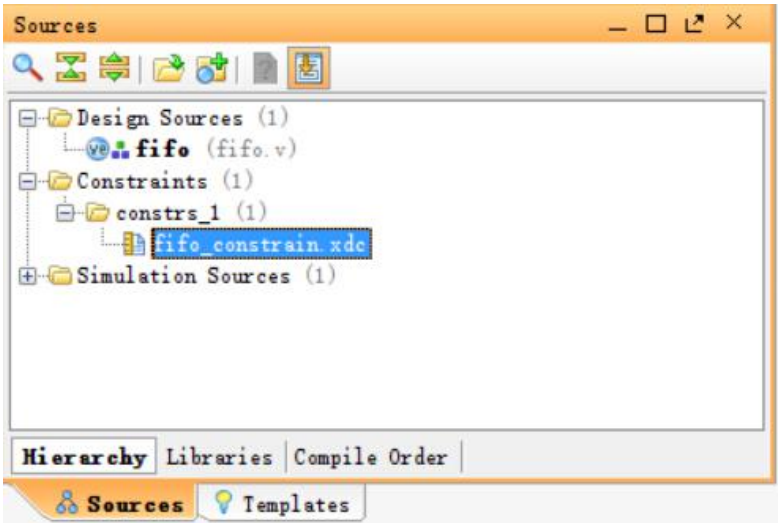


图 5-1-18 展开约束文件 fifo_constrain.xdc

（3）Xilinx 设计约束文件分配 FPGA 位于主板上的开关和指示灯的物理 IO 地址，这些信息可以通过主板的原理图或电路板的用户手册来获得。

本次设计的约束文件代码是通过 Robei 软件自动生成，但是，Robei 软件目前生成的约束代码只有对输入输出端口的分配，在这个设计中，我们使用了一个通过开关控制的模拟时钟 clk，而非系统时钟，这种电路在综合的时候一般都会报错，所以，在约束文件最后，我们需要手动添加一句命令：

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]

来保证流程中不会出错。

完整的约束代码如下：

```
#This file is generated by Robei!
#Pin Assignment for Xilinx FPGA with Software Vivado.
set_property PACKAGE_PIN R18 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN P16 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property PACKAGE_PIN G15 [get_ports data_in[0]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[0]]
set_property PACKAGE_PIN P15 [get_ports data_in[1]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[1]]
set_property PACKAGE_PIN W13 [get_ports data_in[2]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[2]]
set_property PACKAGE_PIN T16 [get_ports data_in[3]]
set_property IOSTANDARD LVCMOS33 [get_ports data_in[3]]
set_property PACKAGE_PIN Y16 [get_ports wr]
set_property IOSTANDARD LVCMOS33 [get_ports wr]
set_property PACKAGE_PIN V16 [get_ports rd]
set_property IOSTANDARD LVCMOS33 [get_ports rd]
set_property PACKAGE_PIN W16 [get_ports full]
set_property IOSTANDARD LVCMOS33 [get_ports full]
set_property PACKAGE_PIN V12 [get_ports empty]
```

 Copyright © Robei

```
set_property IOSTANDARD LVCMOS33 [get_ports empty]
set_property PACKAGE_PIN M14 [get_ports data_out[0]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[0]]
set_property PACKAGE_PIN M15 [get_ports data_out[1]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[1]]
set_property PACKAGE_PIN G14 [get_ports data_out[2]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[2]]
set_property PACKAGE_PIN D18 [get_ports data_out[3]]
set_property IOSTANDARD LVCMOS33 [get_ports data_out[3]]
set_property PACKAGE_PIN U17 [get_ports fifo_cnt[0]]
set_property IOSTANDARD LVCMOS33 [get_ports fifo_cnt[0]]
set_property PACKAGE_PIN V13 [get_ports fifo_cnt[1]]
set_property IOSTANDARD LVCMOS33 [get_ports fifo_cnt[1]]
set_property PACKAGE_PIN H15 [get_ports fifo_cnt[2]]
set_property IOSTANDARD LVCMOS33 [get_ports fifo_cnt[2]]
set_property PACKAGE_PIN J15 [get_ports fifo_cnt[3]]
set_property IOSTANDARD LVCMOS33 [get_ports fifo_cnt[3]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

（4）点击 File->Save File 保存文件。

1.3 使用 Vivado 综合工具来综合设计，并进行 Implementation 任务实现。

（1）单击综合任务下拉菜单中的 Run Synthesis，综合过程将会分析 fifo.v 文件并生成网表文件。当综合过程完成了，且没有错误信息，将会弹出带有三个选项的完成对话框；

（2）如果有错误，根据错误信息提示修改，直至综合没有错误。然后选择 Run Implementation 选项，执行任务实现，然后单击 OK；

（3）任务实现过程将在综合后的设计上运行。当这个过程完成，且没有错误信息，将会弹出带有三个选项的任务实现完成对话框；

（4）如果有错误，根据错误信息提示修改，直至综合没有错误。

1.4 将开发板上的电源开关拨到 ON，生成比特流并打开硬件会话，对 FPGA 进行编程。

（1）确保微型 USB 电缆连接到 PROG UART 接口（在电源开关的旁边）；

（2）确保 JP7 设置为 USB 提供电源；

（3）接通电源板上的开关；

（4）点击任务实现完成弹出的对话框中 Generate Bitstream 或者点击导航窗口中编程和调试任务中的 Generate Bitstream。比特流生成过程将在任务实现设计后运行。当完成比特流生成后会弹出有三个选项的完成对话框；

（5）这一过程将已经生成的 fifo.bit 文件放在 fifo.runs 目录下的 impl_1 目录下；

（6）选择打开硬件管理器 Open Hardware Manager 选项，然后单击确定。硬件管理器窗口将打开并显示“未连接”状态；

（7）点击 Open a new hardware target。如果之前已经配置过开发板你也可以点击最近打开目标链接 Open recent target；

Copyright © Robei

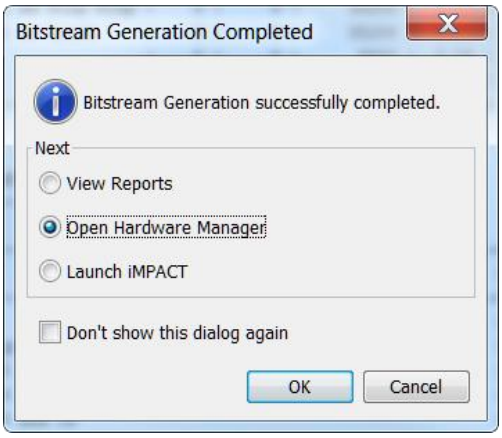


图 5-1-20 比特流生成

图 5-1-21 打开新的硬件目标

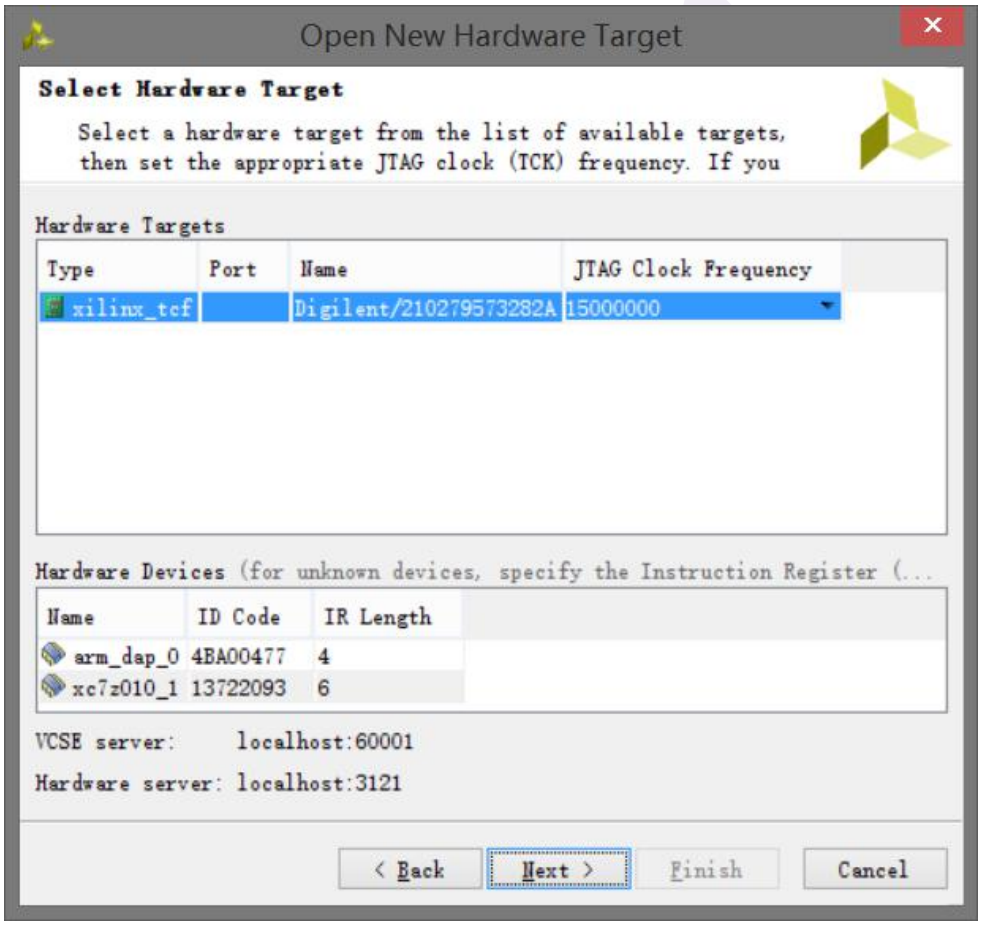


图 5-1-22 新的硬件指标的检测

（8）单击 Next 看 Vivado 自定义搜索引擎服务器名称的形式；

（9）单击 Next 以选择本地主机端口；

（10）单击两次 Next，然后单击 Finish。未连接硬件会话状态更改为服务器名称并且器件被高亮显示。还要注意，该状态表明它还没有被编程；

 Copyright © Robei

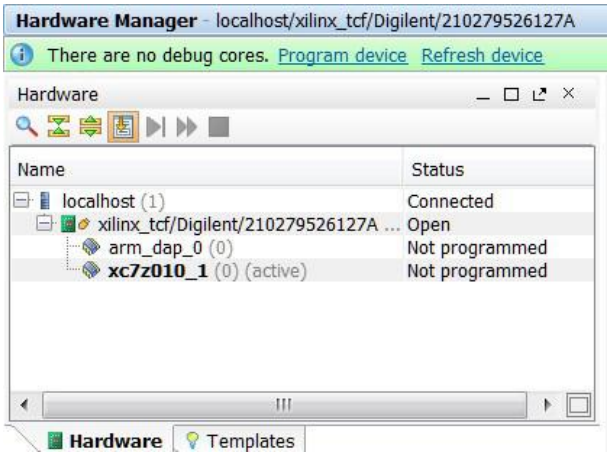


图 5-1-23 打开硬件会话

（11）在器件上单击鼠标右键，选择 Program device 或单击窗口上方弹出的 Program device-> XC7z010_1 链接到目标 FPGA 器件进行编程；

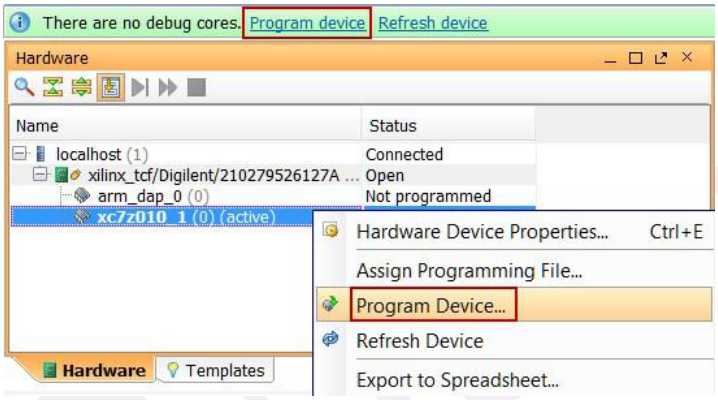


图 5-1-24 选择 FPGA 进行编程

（12）单击确定对 FPGA 进行编程。开发板上 Done 指示灯亮时，器件编程结束；

（13）通过控制拨动和按键开关的开闭来观察 LED（请参考前面的逻辑图）验证输出结果。

2. 开发板验证

首先，按住 res(BTN1)键不放，再按一次至两次 clk(BTN0)键进行复位；

其次，data_in 先输入 0001，即将拨动开关 SW0 拨到 1，然后按住 wr(BTN3)按键不放，再按一下 clk 按键将数据写进 FIFO 存储器，同理分别将 data_in 的 0011、0111、1111、1110、1100、1000、0000 数据写进存储器中；

最后，按住 rd(BTN2)键不放，再按 clk 键，每按一次 clk，就会从 FIFO 存储器中读出一个数据，并显示在 LED 灯上，验证读出数据是否正确。

0001 0011 0111

Copyright © Robei

1111 1110 1100

1000 0000

图 5-1-28 data_in 数据顺序输入及 LED 灯显示实图

由上述结果可验证本次设计的 FIFO 功能正常，达到了设计的目的。

5.1.4. 问题与思考

（1）使用 Robei 设计一个 8 位 4 深度的 FILO。
（2）使用 Robei 设计一个 SPI 总线。

Copyright © Robei

5.2 实例十一 SPI 总线接口的 verilog 的实现

5.2.1. 本章导读

项目中使用的许多器件需要 SPI 接口进行配置，比如 PLL：ADF4350，AD：AD9627，VGA：AD8372 等，本设计根据 SPI 协议，编写了一个简单的 SPI 读写程序，可以进行 32 位数据的读写，可以设置 SPI SCLK 相对于主时钟的分频比。

设计原理

SPI 总线系统是一种同步串行外设接口，它可以使 MCU 与各种外围设备以串行方式进行通信以交换信息，外围设置 FLASHRAM、网络控制器、LCD 显示驱动器、A/D 转换器和 MCU 等。SPI 总线系统可直接与各个厂家生产的多种标准外围器件直接连接，该接口一般使用 4 条线：串行时钟线（SCK）、主机输入/从机输出数据线 MISO、主机输出/从机输入数据线 MOSI 和低电平有效的从机选择线 SS(有的 SPI 接口芯片带有中断信号线 INT、有的 SPI 接口芯片没有主机输出/从机输入数据线 MOSI)。

SPI 的通信原理很简单，它以主从方式工作，这种模式通常有一个主设备和一个或多个从设备，需要至少 4 根线，事实上 3 根也可以（单向传输时）。也是所有基于 SPI 的设备共有的，它们是 SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选）。

（1）MOSI – 主设备数据输出，从设备数据输入

（2）MISO – 主设备数据输入，从设备数据输出

（3）SCLK – 时钟信号，由主设备产生

（4）CS – 从设备使能信号，由主设备控制

其中 CS 是控制芯片是否被选中的，也就是说只有片选信号为预先规定的使能信号时（高电位或低电位），对此芯片的操作才有效。这就允许在同一总线上连接多个 SPI 设备成为可能。

接下来就是负责通讯的 3 根线了。通讯是通过数据交换完成的，这里先要知道 SPI 是串行通讯协议，也就是说数据是一位一位的传输的。这就是 SCLK 时钟线存在的原因，由 SCLK 提供时钟脉冲，SDI，SDO 则基于此脉冲完成数据传输。数据输出通过 SDO 线，数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被读取。完成一位数据传输，输入也使用同样原理。这样，至少 8 次时钟信号的改变（上沿和下沿为一次），就可以完成 8 位数据的传输。

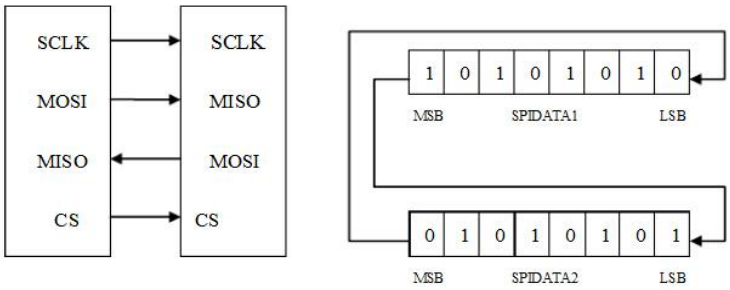


图 5-2-1 SPI 的环形总线结构

如图 5-2-1 所示，当第一个上升沿来的时候，SPIDATA1 将最高位 1 移除，并将所有数据左移 1 位，这时 MOSI 线为高电平，而 SPIDATA2 将最高位 0 移出，并将所有数据左移 1 位，这样 MISO 线为低电平。然后当下降沿到来的时候，SPIDATA1 将锁存 MISO 线上的电平，并将其移入其最低位，同样的，SPIDATA2 将锁存 MOSI 线上的电平，并将其移入最低位。经过 8 个脉冲后，两个移位寄存器就实现了数据的交换，也就是完成了一次 SPI 的时序。

5.2.2. 设计流程

1. spi_master 模型设计

（1）新建一个模型，名为 spi_master，类型为 module，具备 6 个输入，1 个输出，3 个输入输出，每个引脚的属性和名称如下图 5-2-2 所示。

Name	Inout	DataType	Datasize
addr	input	wire	2
in_data	input	wire	32
rd	input	wire	1
wr	input	wire	1
cs	input	wire	1
clk	input	wire	1
out_data	output	reg	32
miso	inout	wire	1
mosi	inout	wire	1
sclk	inout	wire	1

图 5-2-2 spi_master 的引脚属性

Copyright © Robei

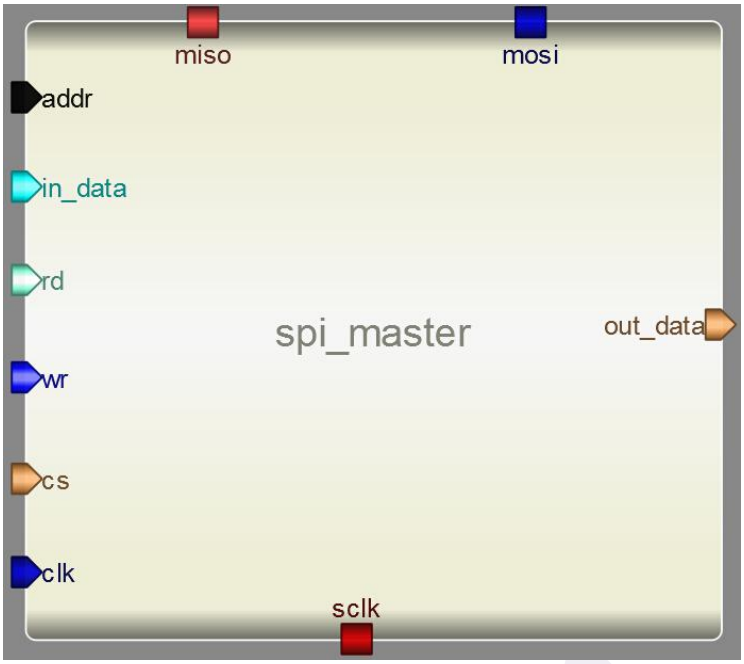


图 5-2-3 spi_master 的界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
reg sclk_buffer = 0;
reg mosi_buffer = 0;
reg busy = 0;
reg [31:0] in_buffer = 0;
reg [31:0] out_buffer = 0;
reg [7:0] clkcount = 0;
reg [7:0] clkdiv = 0;
reg [6:0] count = 0;

always@(cs or rd or addr or out_buffer or busy or clkdiv)
begin
    out_data = 32'bx;
    if(cs && rd)//selected and read
    begin
        case(addr)
        2'b00:
        begin
            out_data = out_buffer;
        end
        2'b01:
        begin
            out_data = {31'b0, busy};
        end
        2'b10:
```

```
            begin
                out_data = clkdiv;
            end
            endcase
        end
    end

    always@(posedge clk)
    begin
        if(!busy)
        begin
            if(cs && wr)
            begin
                case(addr)
                2'b00:
                begin
                    in_buffer = in_data;
                    busy = 1'b1;
                end
                2'b10:
                begin
                    clkdiv = in_data;
                end
                endcase
            end
        end
        else
        begin
            clkcount = clkcount + 1;
            if(clkcount >= clkdiv)
            begin
                clkcount = 0;
                if((count % 2) == 0)
                begin
                    mosi_buffer = in_buffer[31];
                    in_buffer = in_buffer << 1;
                end
                if(count > 0 && count < 65)
                begin
                    sclk_buffer = ~sclk_buffer;
                end
                count = count + 1;
```

 Copyright © Robei

```
                if(count > 65)
                begin
                    count = 0;
                    busy = 1'b0;
                end
            end
        end
    end

    always@(posedge sclk_buffer)
    begin
        out_buffer = out_buffer << 1;
        out_buffer[0] = miso;
    end

    assign sclk = sclk_buffer;
    assign mosi = mosi_buffer;
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

2. spi_master_tb 测试文件的设计

（1）新建一个 6 输入 1 输出 3 输入输出的 spi_master_tb 测试文件，记得将 Module Type 设置为“testbench”，各个引脚配置如图 5-2-4 所示。

Name	Inout	DataType	Datasize
addr	input	reg	2
in_data	input	reg	32
rd	input	reg	1
wr	input	reg	1
cs	input	reg	1
clk	input	reg	1
out_data	output	wire	32
miso	inout	wire	1
mosi	inout	wire	1
sclk	inout	wire	1

图 5-2-4 spi_master_tb 的引脚属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）添加模型。 在 Toolbox 工具箱的 Current 栏里，会出现之前创建的模型，单击该模型并在 spi_master_tb 上添加，并连接引脚，如下图 5-2-5 所示：

Copyright © Robei

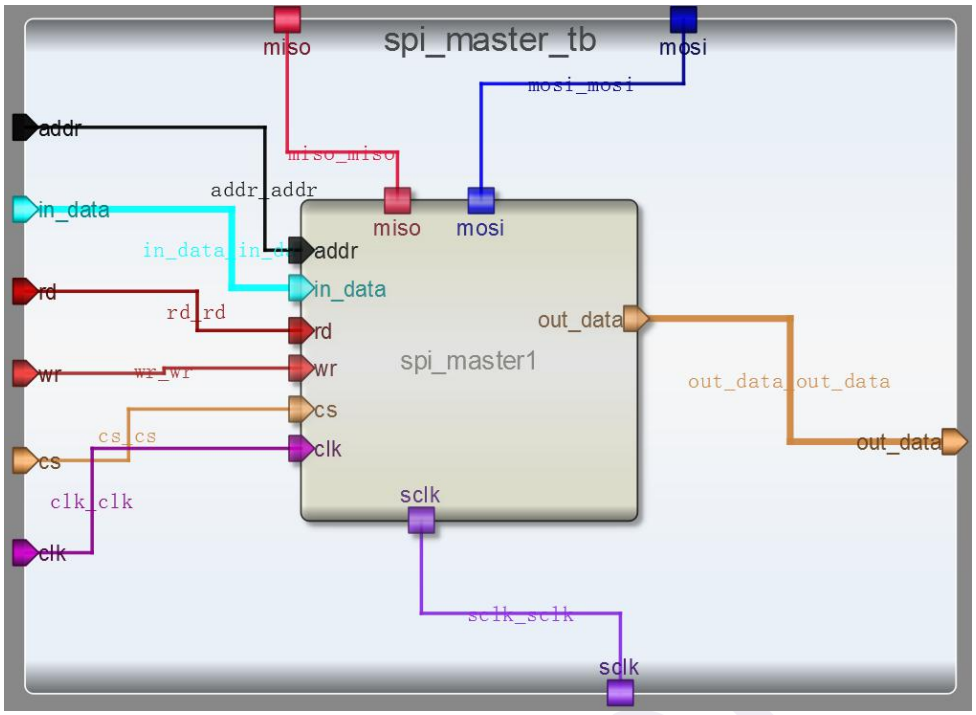


图 5-2-5 spi_master_tb 的界面图

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码用 $finish 结束。测试代码：

```
integer counter = 0;
initial begin
    addr = 0;
    in_data = 0;
    rd = 0;
    wr = 0;
    cs = 0;
    clk = 0;
    #20;
    addr = 2;
    in_data = 0;
    wr = 1;
    cs = 1;
    #20;
    addr = 0;
    in_data = 0;
    wr = 0;
    cs = 0;
    #20;
    for(counter = 0; counter < 256; counter = counter + 1)
    begin
        addr = 0;
```


Copyright © Robei

```
            in_data = counter;
            wr = 1;
            cs = 1;
            #20;
            addr = 0;
            in_data = 0;
            wr = 0;
            cs = 0;
            #20;
            addr = 1;
            cs = 1;
            rd = 1;
            #20;
            while(out_data[0] == 1'b1)
            begin
                #20;
            end
            cs = 0;
            rd = 0;
        end
        #20 $finish;
    end

    always #10 clk = ~clk;
```

（5）执行仿真并查看波形。查看输出信息。

检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 5-2-6 所示：

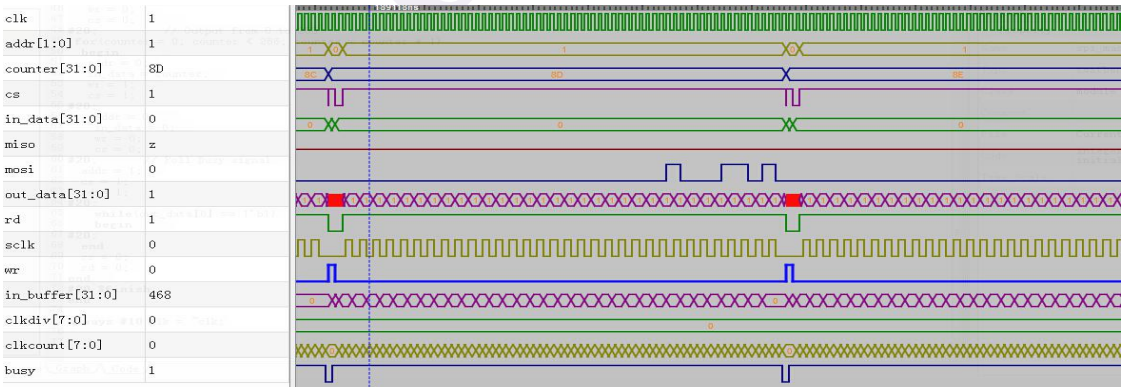


图 5-2-6 spi_master_tb 的仿真波形

5.2.3. SPI 接口协议的板级验证

在这里我们另外建立了一个 VIVADO 工程来验证 SPI 协议的工作流程。没有采用本案例之前的设计是因为之前设计的模块的数据宽度、接口数目等都超过了开发板的限制。新建的工程是一个简化的 SPI 模型，代码会在后面给出。

该模型使用四个拨码开关作为数据输入，使用三个按键开关作为 reset、clock、读使能信号，使用四个 LED 灯显示当前 master 中存储数据的值。模块代码如下：

```
module spi_sim(in, clk, rst, master, rd);
input [3:0] in;
input clk;
input rst;
output [3:0] master;
input rd;
reg [3:0] master;
reg [3:0] slave;
reg mosi;
reg miso;

always @ (posedge clk or negedge rst)
begin
    if (rst)
    begin
        master<=4'b0;
        slave<=4'b0;
        miso<=0;
        mosi<=0;
    end
    else if (rd)
    begin
        slave<=master;
        master<=in;
        miso<=0;
        mosi<=0;
    end
    else
    begin
        mosi<=master[3];
        master[3]<=master[2];
        master[2]<=master[1];
        master[1]<=master[0];
        master[0]<=miso;
        miso<=slave[3];
        slave[3]<=slave[2];
        slave[2]<=slave[1];
        slave[1]<=slave[0];
        slave[0]<=mosi;
```


Copyright © Robei

```
        end
    end
    endmodule
```

与此模块对应的约束文件的代码如下：

```
set_property PACKAGE_PIN T16 [get_ports {in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in[3]}]
set_property PACKAGE_PIN W13 [get_ports {in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in[2]}]
set_property PACKAGE_PIN P15 [get_ports {in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in[1]}]
set_property PACKAGE_PIN G15 [get_ports {in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in[0]}]
set_property PACKAGE_PIN Y16 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property PACKAGE_PIN V16 [get_ports rd]
set_property IOSTANDARD LVCMOS33 [get_ports rd]
set_property PACKAGE_PIN P16 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property PACKAGE_PIN D18 [get_ports {master[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {master[3]}]
set_property PACKAGE_PIN G14 [get_ports {master[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {master[2]}]
set_property PACKAGE_PIN M15 [get_ports {master[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {master[1]}]
set_property PACKAGE_PIN M14 [get_ports {master[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {master[0]}]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
```

在 VIVADO 软件中建立工程，并执行综合、实现、生成比特流等一系列流程后，即可在 ZYBO 开发板上验证 SPI 协议。

（1）按住 BTN3，再按几次 BTN1，完成 reset 工作对数据进行复位。

（2）将四个拨码开关组合一个数据，这里使用 1011 验证，之后按住 BTN2，按一下 BTN1，将 1011 这个数据读入 master。此时 LED 灯应当亮起，显示 master 的值，如图 5-2-7 所示。而这时 slave 的值应当是 0000。

图 5-2-7 master 读入拨码开关的值

（3）接下来，每按动一次 BTN1，即模拟一个时钟周期，完成一位数据传递。按动五次后，应该完成 master 和 slave 的数据互换，此时 LED 灯全灭。（按动四次后，最后一位数据还在 mosi 中，虽然 LED 也是全灭，但是数据互换还没有完成）

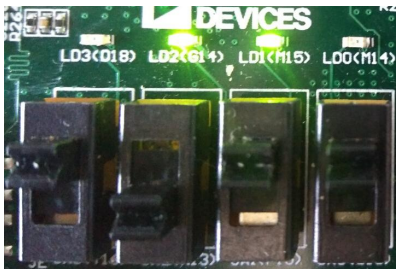


图 5-2-8 按动一次 BTN1

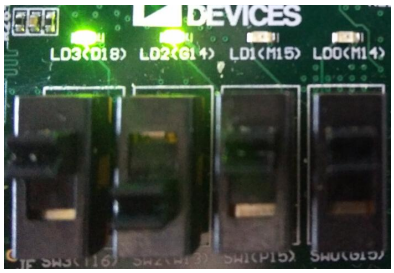


图 5-2-9 按动两次 BTN

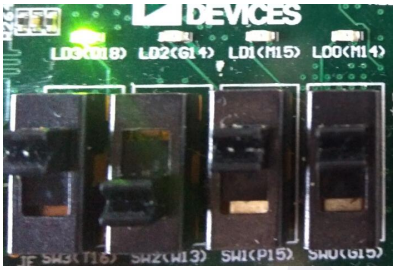


图 5-2-10 按动三次 BTN1

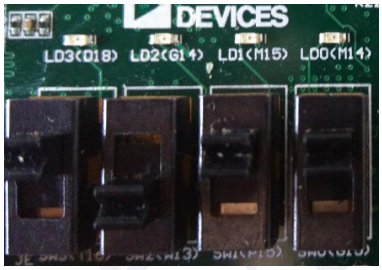


图 5-2-11 按动四次 BTN

（4）继续按动 BTN1 进行时钟操作，五次后，会看到 LED 灯又组成 1011，即实现了 SPI 的数据循环传递。

图 5-2-12 SPI 的数据传递

5.2.4. 问题与思考

本设计只是在 SPI 协议的基础上编写的一个简单的 SPI 读写程序，可以进行 32 位数据的读写，同时可以设置 SPI SCLK 相对于主时钟的分频比。在本次试验的基础上尝试设计具备全部功能的 MC68HC11A8 单片机的 SPI 接口，具有 12 种速率选择并支持四种传输模式。

Copyright © Robei

Robei

第六天：串口通信，系统设计

今天我们设计的案例比之前的难度有所提升，串口通信应用在软硬件开发的各个领域，主要实现 PC 端和开发板之间的通信。不同的串口通信会有不同的参数配置，读者们还需要根据实际应用进行修改。处理器是集成电路中核心的部分，每一个处理器都有自己独特的结构、组成、指令集和特定功能，虽然学会案例中的处理器设计并不意味着学会全部处理器的设计，但是能对处理器的很多基本结构有一个全面的了解。

Robei

 Copyright © Robei

6.1 实例十二 UART 的发送与接收模块设计

6.1.1. 本章导读

设计目的

（1）学习 UART 的工作原理，并用 verilog 设计编写 UART 的发送/接收模块。
（2）熟练运用 Robei 软件进行调试模拟仿真。

设计原理

UART 的帧格式

异步串行数据的一般格式是：起始位+数据位+结束位，其中起始位是 1 位，数据位是 8 位数据或 7 位数据加 1 位奇偶校验位，停止位是 2 位。如图 6-1-1 所示：

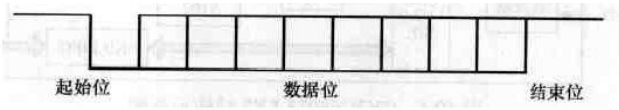


图 6-1-1

（1）接收原理：

由于 UART 是异步传输，没有传输同步时钟。为了能保证数据传输的正确性，采样模块利用 16 倍数据波特率的时钟进行采样，假设波特率为 115200，则采样时钟为 clk16x=115200×16。每个数据占据 16 个采样时钟周期，1bit 起始位+8bit 数据位+1bit 停止位=10bit，因此一帧共占据 16×10=160 个采样时钟，考虑到每个数据位可能有 1-2 个采样时钟周期的偏移，因此将各个数据位的中间时刻作为采样点，以保证采样不会滑码或误码。一般 UART 一帧的数据位数为 8，这样即使每个数据有一个时钟的误差，接收端也能正确地采样到数据。因此，采样时刻为 24(跳过起始位的 16 个时钟)、40、56、72、88、104、120、136、152(停止位)，如下图 6-1-2 所示：

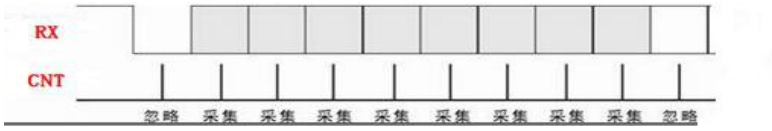


图 6-1-2 数据与采样时间点

其中，RX 为接收引脚，CNT 为对采样时钟进行计数的计数器。

（2）发送原理：

当并行数据准备好后，如果得到发送指令，则将数据按 UART 协议输出，先输出一个低电平的起始位，然后从低到高输出 8 个数据位，接着是可选的奇偶校验位，最后是高电平的停止位；

由于发送时钟 clk16x 为波特率的 16 倍，因此对 clk16x 计数到 16 时，发送 D0；计数到 32 时，发送 D1……依此类推，如图 6-1-3 所示；

图 6-1-3 数据与发送时间点

6.1.2. 设计流程

1. 接收模块的设计

（1）新建一个模型，名为 UART，类型为 module，具备 3 输入 2 输出，每个引脚的属性和名称如下图 6-1-4 所示。

Name	Inout	DataType	Datasize
clk16x	input	wire	1
rst_n	input	wire	1
rx	input	wire	1
DataReady	output	reg	1
DataReceived	output	reg	8

图 6-1-4 UART 的引脚属性

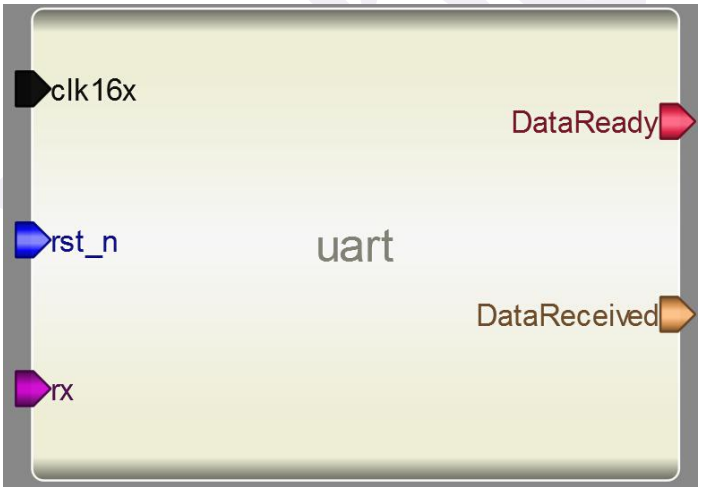


图 6-1-5 UART 的界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
reg[7:0] cnt;
reg trigger_r0;
reg [3:0] count;
wire neg_tri;

always @ (posedge clk16x or negedge rst_n)
begin
    if(!rst_n)
```

 Copyright © Robei

```
        begin
            trigger_r0<=1'b0;
        end
        else
        begin
            trigger_r0<=rx;
        end
    end

    assign neg_tri=trigger_r0&~rx;
    reg cnt_en;

    always @ (posedge clk16x or negedge rst_n)
    begin
        if(!rst_n)
            cnt_en<=1'b0;
        else if(neg_tri==1'b1)
            cnt_en<=1'b1;
        else if(cnt==8'd152)
            cnt_en<=1'b0;
    end

    always @ (posedge clk16x or negedge rst_n)
    begin
        if(!rst_n)
            cnt<=8'd0;
        else  if(cnt_en)
            cnt<=cnt+1;
        else
            cnt<=8'd0;
    end

    always @ (posedge clk16x or negedge rst_n)
    begin
        if(!rst_n)
        begin
            DataReceived<=8'b0;
            count<=0;
        end
        else if(cnt_en)
            case(cnt)
                8'd24:begin DataReceived[0]<=rx;count<=count+1;end
```

Robei

```
                8'd40:begin DataReceived[1]<=rx;count<=count+1;end
                8'd56:begin DataReceived[2]<=rx;count<=count+1;end
                8'd72:begin DataReceived[3]<=rx;count<=count+1;end
                8'd88:begin DataReceived[4]<=rx;count<=count+1;end
                8'd104:begin DataReceived[5]<=rx;count<=count+1;end
                8'd120:begin DataReceived[6]<=rx;count<=count+1;end
                8'd136:begin DataReceived[7]<=rx;count<=count+1;end
            endcase
    end

    always@(posedge clk16x or negedge rst_n)
    begin
        if(!rst_n)
            DataReady<=1'b0;
        else if(cnt==8'd152)
            DataReady<=1'b1;
        else
            DataReady<=1'b0;
    end
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

2. UARTTEST 测试文件的设计

（1）新建一个 3 输入 2 输出的 uarttest 测试文件，记得将 Module Type 设置为 “testbench”，各个引脚配置如图 6-1-6 所示。

Name	Inout	DataType	Datasize
clk16x	input	reg	1
rst_n	input	reg	1
rx	input	reg	1
DataReady	output	wire	1
DataReceived	output	wire	8

图 6-1-6 uarttest 的引脚属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）添加模型。在 Toolbox 工具箱的 Current 栏里会出现模型，单击该模型并在 uarttest 上添加，并连接引脚，如下图 6-1-7 所示：

Copyright © Robei

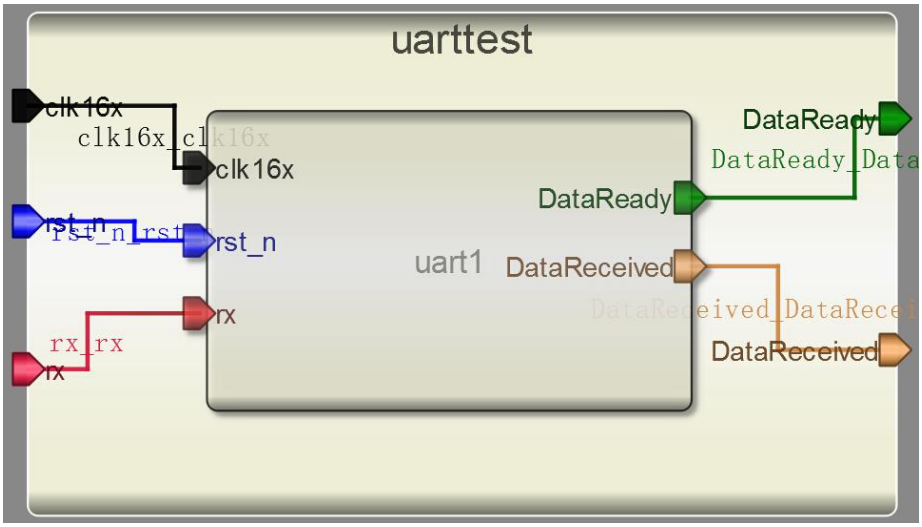


图 6-1-7 uarttest 的界面图

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码用 $finish 结束。测试代码：

```
initial begin
    clk16x=0;
    rst_n=0;
    rx=0;
    #2
    rst_n=1;
    #2
    rst_n=0;
    #2
    rst_n=1;
    #2
    rx=1;
    #32
    rx=0;
    #64
    rx=1;
    #128
    rx=0;
    #64
    rx=1;
    #32
    rx=0;
    #32
    rx=1;
    #100
    $finish;
end
```

Robei

always #1 clk16x=~clk16x;

（5）执行仿真并查看波形。查看输出信息。

检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 6-1-8 所示：

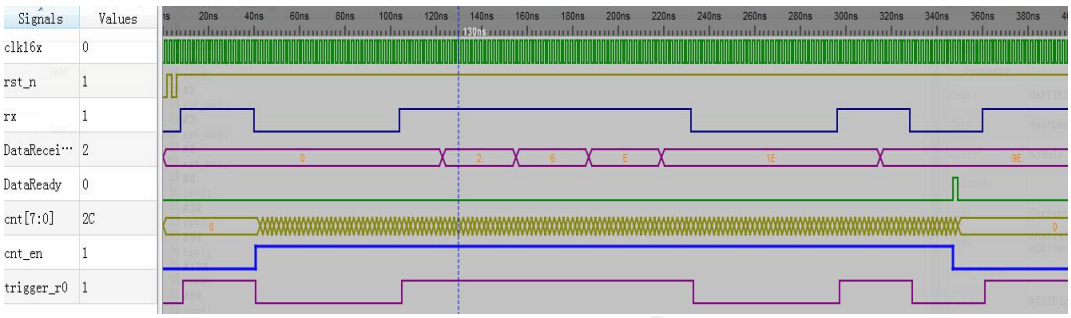


图 6-1-8 uarttest 的仿真波形

3. 发送模块设计

（1）新建一个模型，名为 uartsend，类型为 module，具备 4 输入 2 输出，每个引脚的属性和名称如下图 6-1-9 所示。

Name	Inout	DataType	Datasize
clk16x	input	wire	1
rst_n	input	wire	1
TransEn	input	wire	1
DataToTrans	input	wire	8
BufFull	output	reg	1
tx	output	reg	1

图 6-1-9 uartsend 的引脚属性

图 6-1-10 uartsend 的界面图

Copyright © Robei

（2）添加代码。点击模型下方的 Code 添加代码。

```
reg [7:0] cnt;
reg TransEn_r;
wire pos_tri;
reg cnt_en;

always@(posedge clk16x or negedge rst_n)
begin
    if(!rst_n)
        TransEn_r <= 1'b0;
    else
        TransEn_r <= TransEn;
end

assign pos_tri = ~TransEn_r & TransEn;
reg [7:0] ShiftReg;

always @ (posedge pos_tri or negedge rst_n)
begin
    if(!rst_n)
        ShiftReg <= 8'b0;
    else
        ShiftReg <= DataToTrans;
end

always @ (posedge clk16x or negedge rst_n)
begin
    if(!rst_n)
    begin
        cnt_en <= 1'b0;
        BufFull <= 1'b0;
    end
    else if(pos_tri==1'b1)
    begin
        cnt_en <=1'b1;
        BufFull <= 1'b1;
    end
    else if(cnt==8'd160)
    begin
        cnt_en<=1'b0;
        BufFull <= 1'b0;
    end
end
```

Robei

```
always @ (posedge clk16x or negedge rst_n)
begin
    if(!rst_n)
        cnt<=8'd0;
    else if(cnt_en)
        cnt<=cnt+1;
    else
        cnt<=8'd0;
end

always @ (posedge clk16x or negedge rst_n)
begin
    if(!rst_n)
    begin
        tx <= 1'b1;
    end
    else if(cnt_en)
        case(cnt)
        8'd0   :  tx <= 1'b0;
        8'd16  :  tx <= ShiftReg[0];
        8'd32  :  tx <= ShiftReg[1];
        8'd48  :  tx <= ShiftReg[2];
        8'd64  :  tx <= ShiftReg[3];
        8'd80  :  tx <= ShiftReg[4];
        8'd96  :  tx <= ShiftReg[5];
        8'd112 :  tx <= ShiftReg[6];
        8'd128 :  tx <= ShiftReg[7];
        8'd144 :  tx <= 1'b1;
        endcase
    else
        tx <= 1'b1;
end
```

（3）运行并检查有无错误输出，将文件保存到上面创建的模型所在的文件夹下。

4. UARTsendtest 测试文件的设计

（1）新建一个 4 输入 2 输出的 UARTsendtest 测试文件，记得将 Module Type 设置为“testbench”，各个引脚配置如图 6-1-11 所示。

Copyright © Robei

Name	Inout	DataType	Datasize
clk16x	input	reg	1
rst_n	input	reg	1
TransEn	input	reg	1
DataToTrans	input	reg	8
BufFull	output	wire	1
tx	output	wire	1

图 6-1-11 uartsendtest 的引脚属性

（2）另存为测试文件。将测试文件保存到上面创建的模型所在的文件夹下。

（3）添加模型。在 Toolbox 工具箱的 Current 栏里会出现模型，单击该模型并在 uartsendtest 上添加，并连接引脚，如下图 6-1-12 所示：

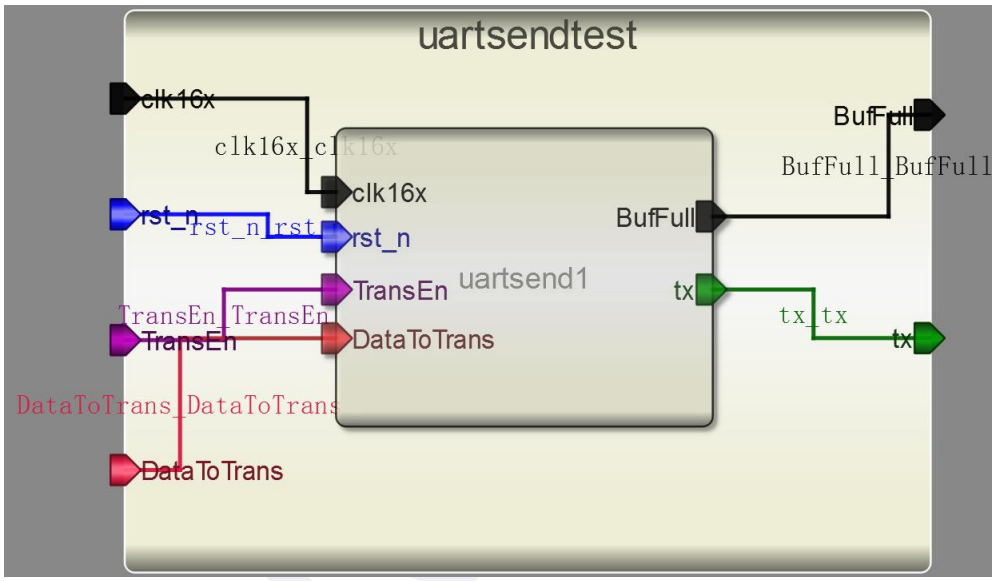


图 6-1-12 uartsendtest 的界面图

（4）输入激励。点击测试模块下方的“Code”，输入激励算法。激励代码用 $finish 结束。测试代码：

```
initial begin
    clk16x=0;
    rst_n=1;
    TransEn=0;
    DataToTrans=0;
    #2
    rst_n=0;
    #2
    DataToTrans=8'b10110010;
    #2
    rst_n=1;
```

```
        #2
        TransEn=1;
        #1000
        $finish;
    end

    always #1 clk16x=~clk16x;
```

（5）执行仿真并查看波形。查看输出信息。

检查没有错误之后查看波形。点击右侧 Workspace 中的信号，进行添加并查看分析仿真结果。如图 6-1-13 所示：

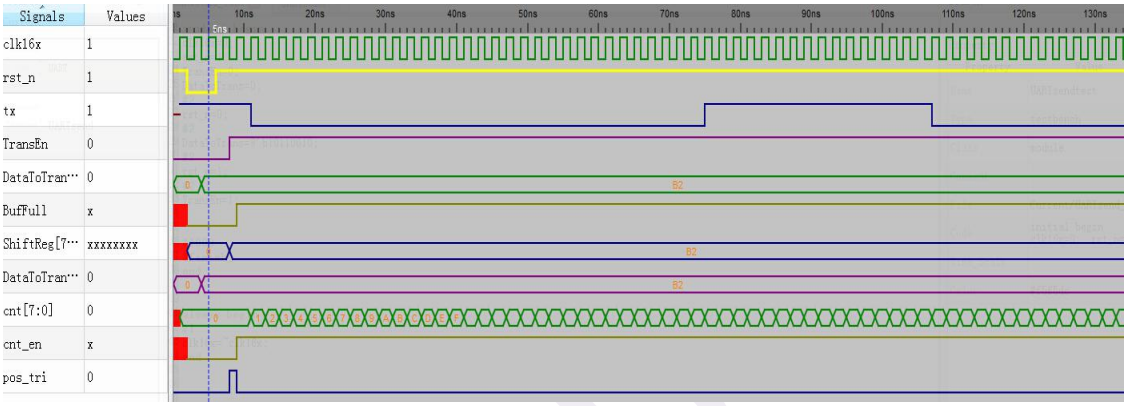


图 6-1-13　UARTsendtest 的仿真波形

6.1.3. 问题与思考

1、采样时钟 clk16x 必须是波特率的 16 倍，波特率任意设置如 57600、9600 等皆可，只要满足 16 倍关系。

2、在本设计模块的基础上，尝试在接收模块中添加校验位，尽可能提高传输精度，降低出错率。

Copyright © Robei

6.2 实例十三 Natalius 8 位 RISC 处理器

6.2.1. 本章导读

设计目的

了解并熟悉 Natalius 8 位 RISC 处理器的基本结构和运行原理。根据 Natalius 的指令集设计出可以验证一些简单功能的 testbench，最后通过 Robei 可视化仿真软件进行功能实现和仿真验证（由于 Robei 目前暂不支持$readmemh()命令，最后的仿真验证在 Modelsim 中进行）。

设计原理

1. Natalius 简介

Natalius 是一个结构紧凑、多功能且完全嵌入式的完全以 Verilog 设计的 8 位 RISC 处理器内核。Natalius 提供了一个可以运行在 python 控制台上的汇编器。Instruction memory 中存储了 2048 条指令，每条指令 16 位宽，其执行需要运行 3 个时钟周期。

2. 指令集

Natalius 包含了大部分处理器所具有的经典指令集。包括：存储访问、数学运算、逻辑运算和数据流控制。如下表 6-2-1 和表 6-2-2 所示。

Instruction	Description	Type
ldi	load immediate	memory access
ldm	load from memory	memory access
stm	store to memory	memory access
cmp	compare	arithmetic
add	addition	arithmetic
sub	subtraction	arithmetic
and	logic and	logical
oor	logic or	logical
xor	logic xor	logical
jmp	jump	flow control
jpz	jump if zero	flow control
jnz	jump if no zero	flow control
jpc	jump if carry	flow control
jnc	jump if no carry	flow control
csr	call subrutine	flow control

Instruction	Description	Type
ret	return subrutine	flow control
adi	add with imm	arithmetic
csz	csr if zero	flow control
cnz	csr if no zero	flow control
csc	csr if carry	flow control
cnc	csr if no carry	flow control
sl0	shift left zero fill	logical
sl1	shift left one fill	logical
sr0	shift right zero fill	logical
sr1	shiftright one fill	logical
rrl	rotary register left	logical
rrr	rotay register right	logical
not	logic not	logical
nop	no operation	nop

表 6-2-1

Opcode	Instr	Description	Use
2	ldi	load immediate	ldi rd,imm (rd=imm)
3	ldm	load from memory	ldm rd,port_addr (rd=data_in <= mem[port_addr])
4	stm	store to memory	stm rd,port_addr (rd=data_out => mem[port_addr])
5	cmp	compare	cmp rd,rs (affects carry and zero)
6	add	addition	add rd,rs (rd=rd+rs)
7	sub	subtraction	sub rd,rs (rd=rd-rs)
8	and	logic and	and rd,rs (rd=rd and rs)
9	oor	logic or	oor rd,rs (rd=rd or rs)
10	xor	logic xor	xor rd,rs (rd=rd xor rs)
11	jmp	jump	jmp inst_addr (pc=inst_addr)
12	jpz	jump if zero	jpz inst_addr (pc=inst_addr if zero)
13	jnz	jump if no zero	jnz inst_addr (pc=inst_addr if no zero)
14	jpc	jump if carry	jpc inst_addr (pc=inst_addr if carry)
15	jnc	jump if no carry	jnc inst_addr (pc=inst_addr if no carry)
16	csr	call subrutine	csr inst_addr (pc=inst_addr) save pc+1 in stack
17	ret	return subrutine	ret (pc=value stored in stack)
18	adi	add with imm	add rd,imm (rd=rd+imm)
19	csz	csr if zero	csr inst_addr (pc=inst_addr if zero) affects stack
20	cnz	csr if no zero	csr inst_addr (pc=inst_addr if no zero) affects stack
21	csc	csr if carry	csr inst_addr (pc=inst_addr if carry) affects stack
22	cnc	csr if no carry	csr inst_addr (pc=inst_addr if no carry) affects stack
23	sl0	shift left zero fill	sl0 rd (rd <= {rd[6:0],0})
24	sl1	shift left one fill	sl1 rd (rd <= {rd[6:0],1})
25	sr0	shift right zero fill	sr0 rd (rd <= {0,rd[7:1]})
26	sr1	shiftright one fill	sr1 rd (rd <= {1,rd[7:1])
27	rrl	rotary register left	rrl rd (rd <= {rd[6:0],rd[7]})
28	rrr	rotay register right	rrr rd (rd <= {rd[0],rd[7:1])
29	not	logic not	sub rd,rs (rd=rd-rs)
30	nop	no operation	no operation (take 3 clk)

表 6-2-2

3. Natalius 接口信号

Natalius 处理器顶层的接口信号如下图 6-2-1 所示，每个信号的具体含义列在表 6-2-3 中

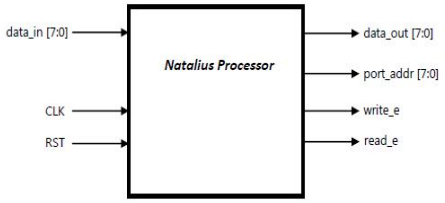


图 6-2-1

Signal	Direction	Description
clk	input	**clock input:** All Natalius registers are clocked from the rising clock edge
rst	input	**reset input:** To reset the Natalius processor, this rst is asynchronous input and it set program counter register to zero address
data_in[7:0]	input	**Input data port:** The data is captured on the rising edge of CLK (used in *ldm* instruction)
data_out[7:0]	output	**Output data port:** Output data appears on this port for three CLK cycles during a stm instruction, capture this data when write_e is high (uded in *stm* instruction)
port_addr[7:0]	output	**Port address:** This addresses the peripheral port to the input or output by instruction *ldm* or *stm*

表 6-2-3

4. 汇编器脚本使用

安装 Python：

（1）下载 Python

打开浏览器，输入地址：https://www.python.org/downloads/release/python-343/拖动到页面最下方，根据自己的电脑配置选择相应的版本，如图 6-2-2 所示。

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		4281ff86778db65892c05151d5de738d	19554643	SIG
XZ compressed source tarball	Source release		7d092d1bba6e17f0d9bd21b49e441dd5	14421964	SIG
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later	548f79e55708130c755bbd0f1ddd921c	24734803	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86b29d7dddc60b4b3fc5848de55ca704	23170148	SIG
Windows debug information files	Windows		b3d8752e74a502db97bd0c6ef30ac60f	36900012	SIG
Windows debug information files for 64-bit binaries	Windows		6c1be415ae552e190ef0fb06a5de9473	24301250	SIG
Windows help file	Windows		d5703787758eb1a674101ee2b0bc28be	7405996	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	f6ade29acaf8fcdc0463e69a6e7ccf87	25550848	SIG
Windows x86 MSI installer	Windows		cb450d1cc616bfc8f7a2d6bd88780bf6	24846336	SIG

图 6-2-2 选择版本

由于操作的电脑是 Windows7 64 位操作系统，所以选择 Windows x86-64 MSI installer 进行下载。

（2）安装 Python

双击下载下来的安装包进行安装。并一定要记清安装时选择的目录。默认的安装目录是“C:\python34”。

使用 assembler.py 对 code.asm 进行转换：

（1）编写 code.asm 代码

根据“Natalius 8 bit RISC Processor.pdf”文档中第 3 页 Table 3，并参照第 4 页到第 5 页的 5.1 Example 编写一段测试代码。

编写的一段简单的测试代码如下：

```
ldi r1, 22
ldi r2, 80
ldi r3, 36
ldi r4, 45
add r1, r2
stm r1, 11
add r3, r4
stm r3, 25
add r1, r3
stm r1, 32
```

其实现的基本功能是两个数的相加。将测试代码保存为“test.asm”。然后将测试代码“test.asm”和从 OpenCores“Natalius 8 bit RISC Processor”下载下来的“assembler.py”一起放在文件夹中，本教程中选择放置于“E:\python_test”中。

（2）在 CMD 环境中使用 Python 将 code.asm 转换为 instruction.mem

打开“开始”菜单，在“搜索程序和文件”搜索框中输入“cmd”并按下回车键，进入如下图 6-2-3 所示的界面：

Robei

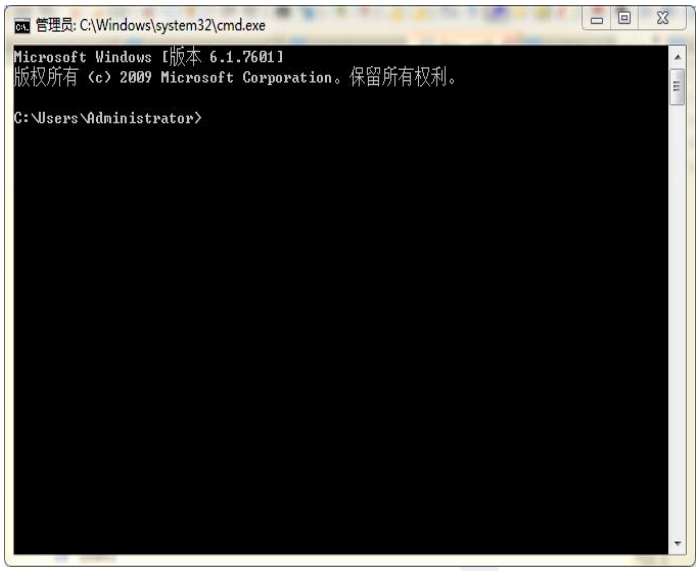


图 6-2-3 进入 cmd 界面

a. 设定 Python 路径

如果安装 Python 的时候没有更改路径，输入以下命令设定 Python 路径：

set path=%path%;C:\python34

b. 打开“test.asm”所在文件夹

本教程中“test.asm”放置于“E:\python_test”，故输入以下命令：

“E:”回车

“cd python_test”回车

然后可以输入 dir 或者 dir /b 查看目录中的内容。

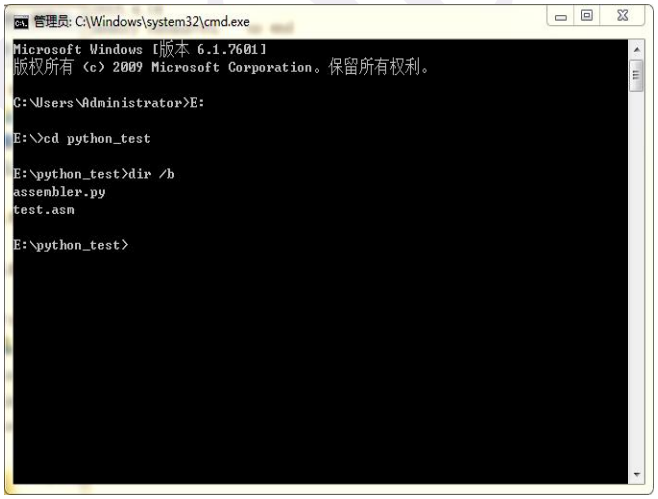


图 6-2-4 进入 python_test 目录

c. 输入命令，进行转换

然后输入“assembler.py -s test.asm”，转换完成之后可以看到“E:\python_test”下多出了一个“instructions.mem”文件。该文件即是我们进行仿真时需要用到的文件。

6.2.2. 设计流程

1. ALU 模型设计

（1）新建一个模型命名为 ALU，类型为 module，同时具备 4 输入 3 输出，每个引脚的属性

 Copyright © Robei

和名称参照下图 6-2-5 进行对应的修改。

Name	Inout	DataType	Datasize	Function
a	input	wire	8	
b	input	wire	8	
opalu	input	wire	3	
sh	input	wire	3	
dshift	output	reg	8	
zero	output	wire	1	
carry	output	wire	1	

图 6-2-5 ALU 引脚的属性

图 6-2-6 ALU 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
reg [7:0] resu;
wire [7:0] result;
always@(a or b)
    case (opalu)
        0: resu <= ~a;
        1: resu <= a & b;
        2: resu <= a ^ b;
        3: resu <= a | b;
        4: resu <= a;
        5: resu <= a + b;
        6: resu <= a - b;
        default: resu <= a + 1;
    endcase

assign zero=(resu==0);
assign result=resu;
assign carry=(a<b);
```

```
always@(result)
    case (sh)
            0: dshift <= {result[6:0], "0"};
            1: dshift <= {result[6:0], result[7]};
            2: dshift <= {"0", result[7:1]};
            3: dshift <= {result[0], result[7:1]};
            4: dshift <= result;
            5: dshift <= {result[6:0], "1"};
            6: dshift <= {"1", result[7:1]};
            default: dshift <= result;
    endcase
```

（3）保存模型(存储文件夹路径不能有空格和中文)，运行并检查有无错误输出。

2. stack 模型设计

（1）新建一个模型命名为 stack，类型为 module，同时具备 7 输入 2 输出，每个引脚的属性和名称参照下图 6-2-7 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
rst	input	wire	1	
wr_en	input	wire	1	
rd_en	input	wire	1	
ldpc	input	wire	1	
selpc	input	wire	1	
ninst_addr	input	wire	11	
PC	output	reg	11	
out	output	wire	11	

图 6-2-7　stack 引脚的属性

Copyright © Robei

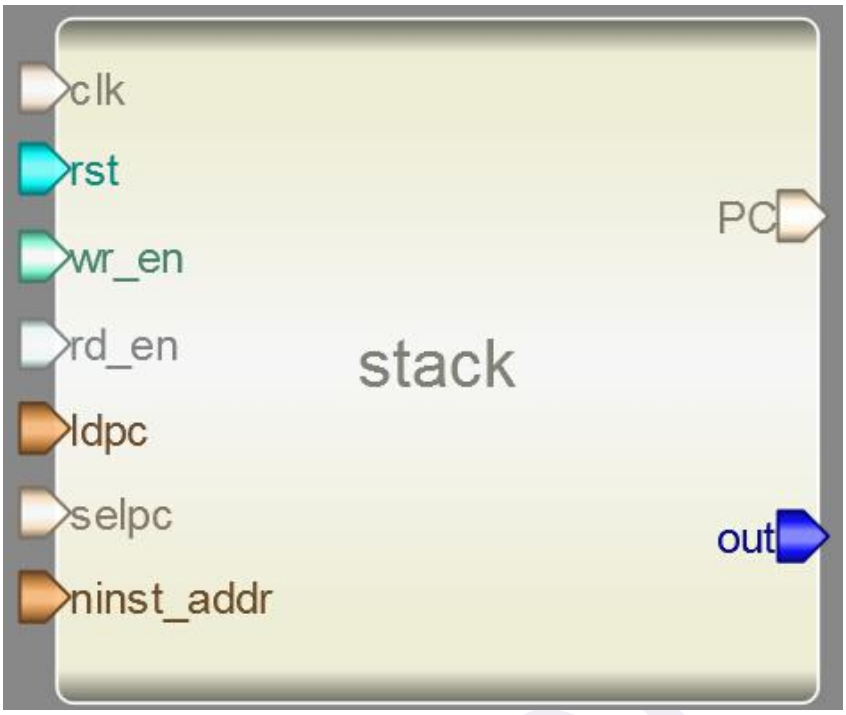


图 6-2-8 stack 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
reg [7:0] resu;
wire [7:0] result;
reg [3:0] addr;
reg [10:0] ram [15:0];
wire [10:0] dout;
wire [10:0] din;

always@(posedge clk or posedge rst)
begin
    if (rst)
        PC<=0;
    else
    if (ldpc)
        if(selpc)
            PC<=ninst_addr;
        else
            PC<=PC+1;
end

assign din = PC;

always@(posedge clk)
begin
    if (rst)
```

Robei

```
            addr<=0;
        else
        begin
            if (wr_en==0 && rd_en==1)
                if (addr>0)
                    addr<=addr-1;
            if (wr_en==1 && rd_en==0)
                if (addr<15)
                    addr<=addr+1;
        end
    end

    always @(posedge clk)
    if (wr_en)
        ram[addr] <= din;

    assign dout = ram[addr];
    assign out = dout + 1;
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

3. data_supply 模型设计

（1）新建一个模型命名为 data_supply，类型为 module，同时具备 12 输入 2 输出，每个引脚的属性和名称参照下图 6-2-9 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
we	input	wire	1	
wa	input	wire	3	
raa	input	wire	3	
rab	input	wire	3	
shiftout	input	wire	8	
insel	input	wire	1	
selk	input	wire	1	
data_in	input	wire	8	
kte	input	wire	8	
selimm	input	wire	1	
imm	input	wire	8	
portA	output	wire	8	
muximm	output	wire	8	

图 6-2-9　data_supply 引脚的属性

Copyright © Robei

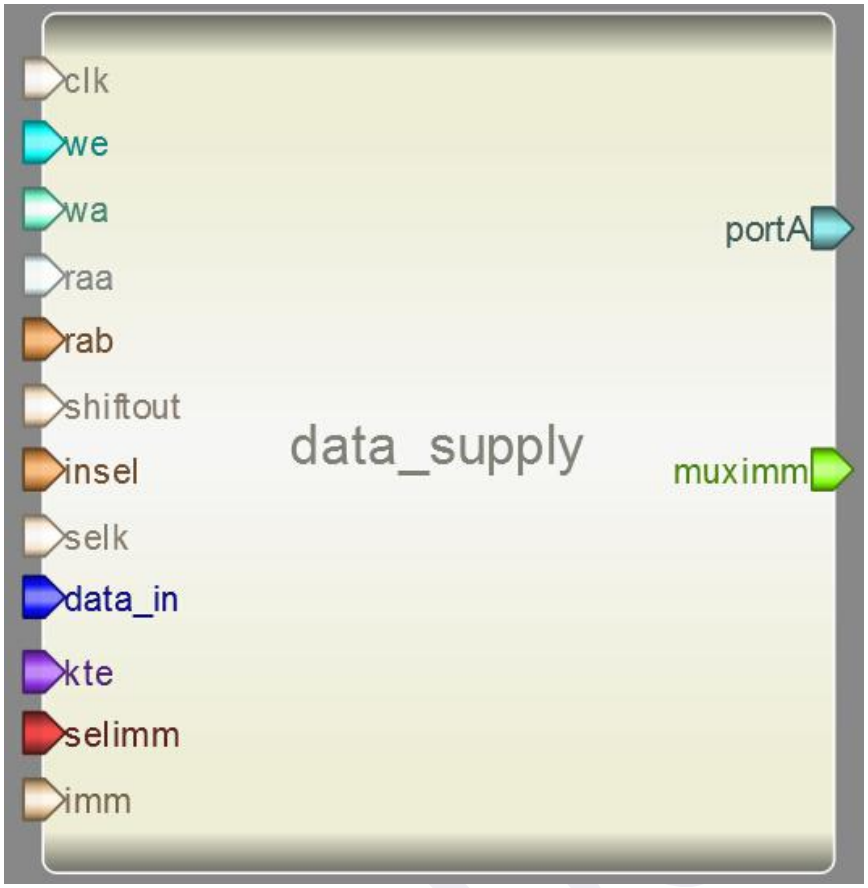


图 6-2-10　data_supply 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
wire [7:0] portB;
wire [7:0] regmux, muxkte;
reg [7:0] mem [7:0];

always@(posedge clk)
begin
      mem[0]<=0;
      if(we)
            mem[wa]<=regmux;
end

assign portA=mem[raa];
assign portB=mem[rab];
assign regmux=insel? shiftout : muxkte;
assign muxkte=selk? kte : data_in;
assign muximm=selimm? imm : portB;
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

4. zc_control 模型设计

（1）新建一个模型命名为 zc_control，类型为 module，同时具备 5 输入 2 输出，每个引脚的属性和名称参照下图 6-2-11 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
rst	input	wire	1	
ldflag	input	wire	1	
zero	input	wire	1	
carry	input	wire	1	
z	output	reg	1	
c	output	reg	1	

图 6-2-11 zc_control 引脚的属性

图 6-2-12 zc_control 界面图

（2）添加代码。点击模型下方的 Code 添加代码。代码：

```
always @ (posedge clk or posedge rst)
begin
    if (rst)
    begin
        z<=0;
        c<=0;
    end
    else
    if (ldflag)
    begin
        z<=zero;
        c<=carry;
    end
end
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

Copyright © Robei

5. data path 模型设计

（1）新建一个模型命名为 data_path，类型为 module，同时具备 20 输入 5 输出，每个引脚的属性和名称参照下图 6-2-13 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
rst	input	wire	1	
data_in	input	wire	8	
insel	input	wire	1	
we	input	wire	1	
raa	input	wire	3	
rab	input	wire	3	
wa	input	wire	3	
opalu	input	wire	3	
sh	input	wire	3	
selpc	input	wire	1	
selk	input	wire	1	
ldpc	input	wire	1	
ldflag	input	wire	1	
wr_en	input	wire	1	
rd_en	input	wire	1	
ninst_addr	input	wire	11	
kte	input	wire	8	
imm	input	wire	8	
selimm	input	wire	1	
data_out	output	wire	8	
inst_addr	output	wire	11	
stack_addr	output	wire	11	
z	output	wire	1	
c	output	wire	1	

图 6-2-13 data_path 引脚的属性

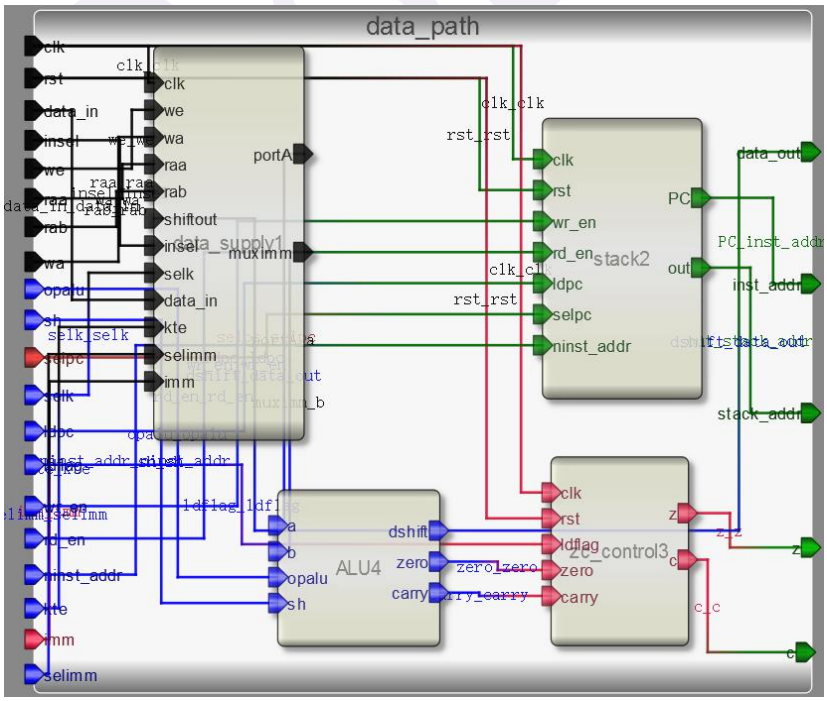


图 6-2-14 data_path 界面图

注意：data_supply 的 muximm 连接到 ALU 的 b 端口；ALU 的 dshift 同时连接到 data_supply

的 shiftout 和 data_path 的 data_out。

（2）添加代码。由于该模块仅仅是把 ALU，stack，data supply 和 zc control 四个模块整合连接起来，所以并无代码。

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

6. instruction memory 模型设计

（1）新建一个模型命名为 instruction_memory，类型为 module，同时具备 2 输入 1 输出，每个引脚的属性和名称参照下图 6-2-15 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
address	input	wire	11	
instruction	output	reg	16	

图 6-2-15 instruction_memory 引脚的属性

图 6-2-16 instruction_memory 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

代码：由于 Robei 软件目前尚不支持$readmemh()函数，故代码中有两行被注释掉了。后面使用 Modelsim 进行仿真的时候需要将这两行代码还原。

```
reg [15:0] rom [2047:0];
wire we;
//initial
//$readmemh("instructions.mem", rom, 0, 2047);
assign we=0;

always @(posedge clk)
if(we)
    rom[address]<=0;
else
    instruction <= rom[address];
```

（3）保存模型(存储文件夹路径不能有空格和中文)，运行并检查有无错误输出。

7. control unit 模型设计

（1）新建一个模型命名为 control_unit，类型为 module，同时具备 6 输入 20 输出，每个引脚的属性和名称参照下图进行对应的修改。

 Copyright © Robei

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
rst	input	wire	1	
instruction	input	wire	16	
z	input	wire	1	
c	input	wire	1	
stack_addr	input	wire	11	
port_addr	output	reg	8	
write_e	output	reg	1	
read_e	output	reg	1	
insel	output	reg	1	
we	output	reg	1	
raa	output	reg	3	
rab	output	reg	3	
wa	output	reg	3	
opalu	output	reg	3	
sh	output	reg	3	
selpc	output	reg	1	
ldpc	output	reg	1	
naddress	output	reg	11	
selk	output	reg	1	
kte	output	reg	8	
wr_en	output	reg	1	
rd_en	output	reg	1	
imm	output	reg	8	
selimm	output	reg	1	
ldflag	output	reg	1	

图 6-2-17 control_unit 引脚的属性

Copyright © Robei

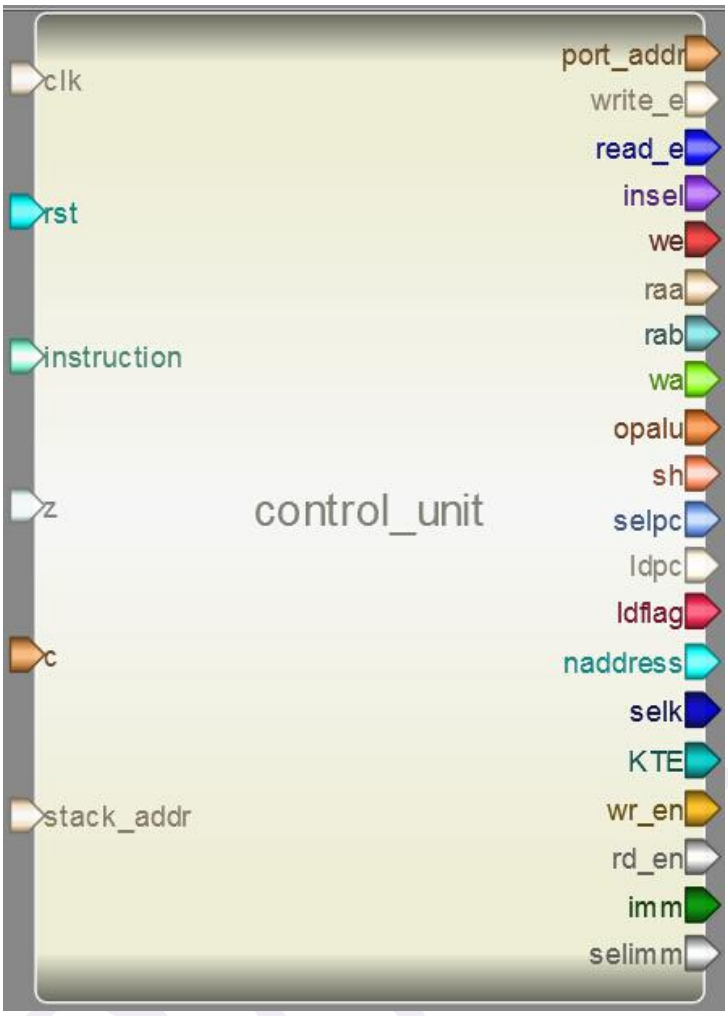


图 6-2-18 control_unit 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

```
parameter fetch=5'd0;
parameter decode=5'd1;
parameter ldi=5'd2;
parameter ldm=5'd3;
parameter stm=5'd4;
parameter cmp=5'd5;
parameter add=5'd6;
parameter sub=5'd7;
parameter andi=5'd8;
parameter oor=5'd9;
parameter xori=5'd10;
parameter jmp=5'd11;
parameter jpz=5'd12;
parameter jnz=5'd13;
parameter jpc=5'd14;
parameter jnc=5'd15;
```


Copyright © Robei

```
parameter csr=5'd16;
parameter ret=5'd17;
parameter adi=5'd18;
parameter csz=5'd19;
parameter cnz=5'd20;
parameter csc=5'd21;
parameter cnc=5'd22;
parameter sl0=5'd23;
parameter sl1=5'd24;
parameter sr0=5'd25;
parameter sr1=5'd26;
parameter rrl=5'd27;
parameter rrr=5'd28;
parameter noti=5'd29;
parameter nop=5'd30;

wire [4:0] opcode;
reg [4:0] state;
assign opcode=instruction[15:11];

always@(posedge clk or posedge rst)
begin
    if (rst)
        state<=decode;
    else
        case (state)
        fetch: state<=decode;
        decode:
            case (opcode)
            2: state<=ldi;
            3: state<=ldm;
            4: state<=stm;
            5: state<=cmp;
            6: state<=add;
            7: state<=sub;
            8: state<=andi;
            9: state<=oor;
            10: state<=xori;
            11: state<=jmp;
            12: state<=jpz;
            13: state<=jnz;
            14: state<=jpc;
```

Robei

```
            15: state<=jnc;
            16: state<=csr;
            17: state<=ret;
            18: state<=adi;
            19: state<=csz;
            20: state<=cnz;
            21: state<=csc;
            22: state<=cnc;
            23: state<=sl0;
            24: state<=sl1;
            25: state<=sr0;
            26: state<=sr1;
            27: state<=rrl;
            28: state<=rrr;
            29: state<=noti;
            default: state<=nop;
            endcase
        ldi:state<=fetch;
        ldm:state<=fetch;
        stm:state<=fetch;
        cmp:state<=fetch;
        add:state<=fetch;
        sub:state<=fetch;
        andi:state<=fetch;
        oor:state<=fetch;
        xori:state<=fetch;
        jmp:state<=fetch;
        jpz:state<=fetch;
        jnz:state<=fetch;
        jpc:state<=fetch;
        jnc:state<=fetch;
        csr:state<=fetch;
        ret:state<=fetch;
        adi:state<=fetch;
        csz:state<=fetch;
        cnz:state<=fetch;
        csc:state<=fetch;
        cnc:state<=fetch;
        sl0:state<=fetch;
        sl1:state<=fetch;
        sr0:state<=fetch;
        sr1:state<=fetch;
```


Copyright © Robei

```
            rrl:state<=fetch;
            rrr:state<=fetch;
            noti:state<=fetch;
            nop:state<=fetch;
            endcase
    end

    always @ (state)
    begin
        port_addr<=0;
        write_e<=0;
        read_e<=0;
        insel<=0;
        we<=0;
        raa<=0;
        rab<=0;
        wa<=0;
        opalu<=4;
        sh<=4;
        selpc<=0;
        ldpc<=1;
        ldflag<=0;
        naddress<=0;
        selk<=0;
        KTE<=0;
        wr_en<=0;
        rd_en<=0;
        imm<=0;
        selimm<=0;
        case (state)
            fetch: ldpc<=0;
            decode:
            begin
                ldpc<=0;
                if (opcode==stm)
                begin
                    raa<=instruction[10:8];
                    port_addr<=instruction[7:0];
                end
                else if (opcode==ldm)
                begin
                    wa<=instruction[10:8];
```

Robei

```
            port_addr<=instruction[7:0];
        end
        else if (opcode==ret)
        begin
            rd_en<=1;
        end
    end
    ldi:
    begin
        selk<=1;
        KTE<=instruction[7:0];
        we<=1;
        wa<=instruction[10:8];
    end
    ldm:
    begin
        wa<=instruction[10:8];
        we<=1;
        read_e<=1;
        port_addr<=instruction[7:0];
    end
    stm:
    begin
        raa<=instruction[10:8];
        write_e<=1;
        port_addr<=instruction[7:0];
    end
    cmp:
    begin
        ldflag<=1;
        raa<=instruction[10:8];
        rab<=instruction[7:5];
        opalu<=6;
    end
    add:
    begin
        raa<=instruction[10:8];
        rab<=instruction[7:5];
        wa<=instruction[10:8];
        insel<=1;
        opalu<=5;
        we<=1;
```


Copyright © Robei

```
end
sub:
begin
    raa<=instruction[10:8];
    rab<=instruction[7:5];
    wa<=instruction[10:8];
    insel<=1;
    opalu<=6;
    we<=1;
end
andi:
begin
    raa<=instruction[10:8];
    rab<=instruction[7:5];
    wa<=instruction[10:8];
    insel<=1;
    opalu<=1;
    we<=1;
end
oor:
begin
    raa<=instruction[10:8];
    rab<=instruction[7:5];
    wa<=instruction[10:8];
    insel<=1;
    opalu<=3;
    we<=1;
end
xori:
begin
    raa<=instruction[10:8];
    rab<=instruction[7:5];
    wa<=instruction[10:8];
    insel<=1;
    opalu<=2;
    we<=1;
end
jmp:
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
```

Robei

```
end
jpz:
if (z)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
end
jnz:
if (!z)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
end
jpc:
if (c)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
end
jnc:
if (!c)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
end
csr:
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
    wr_en<=1;
end
ret:
begin
    naddress<=stack_addr;
    selpc<=1;
    ldpc<=1;
end
```


Copyright © Robei

```
adi:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    imm<=instruction[7:0];
    selimm<=1;
    insel<=1;
    opalu<=5;
    we<=1;
end
csz:
if (z)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
    wr_en<=1;
end
cnz:
if (!z)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
    wr_en<=1;
end
csc:
if (c)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
    wr_en<=1;
end
cnc:
if (!c)
begin
    naddress<=instruction[10:0];
    selpc<=1;
    ldpc<=1;
    wr_en<=1;
end
```

Robei

```
sl0:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    insel<=1;
    sh<=0;
    we<=1;
end
sl1:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    insel<=1;
    sh<=5;
    we<=1;
end
sr0:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    insel<=1;
    sh<=2;
    we<=1;
end
sr1:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    insel<=1;
    sh<=6;
    we<=1;
end
rrl:
begin
    raa<=instruction[10:8];
    wa<=instruction[10:8];
    insel<=1;
    sh<=1;
    we<=1;
end
rrr:
begin
```

Robei

```
                raa<=instruction[10:8];
                wa<=instruction[10:8];
                insel<=1;
                sh<=3;
                we<=1;
            end
            noti:
            begin
                raa<=instruction[10:8];
                wa<=instruction[10:8];
                insel<=1;
                opalu<=0;
                we<=1;
            end
            nop: opalu<=4;
        endcase
    end
```

（3）保存模型（存储文件夹路径不能有空格和中文），运行并检查有无错误输出。

8. Natalius processor 模型设计

（1）新建一个模型命名为 processor，类型为 module，同时具备 3 输入 4 输出，每个引脚的属性和名称参照下图 6-2-19 进行对应的修改。

Name	Inout	DataType	Datasize	Function
clk	input	wire	1	
rst	input	wire	1	
data_in	input	wire	8	
port_addr	output	wire	8	
read_e	output	wire	1	
write_e	output	wire	1	
data_out	output	wire	8	

图 6-2-19　processor 引脚的属性

Copyright © Robei

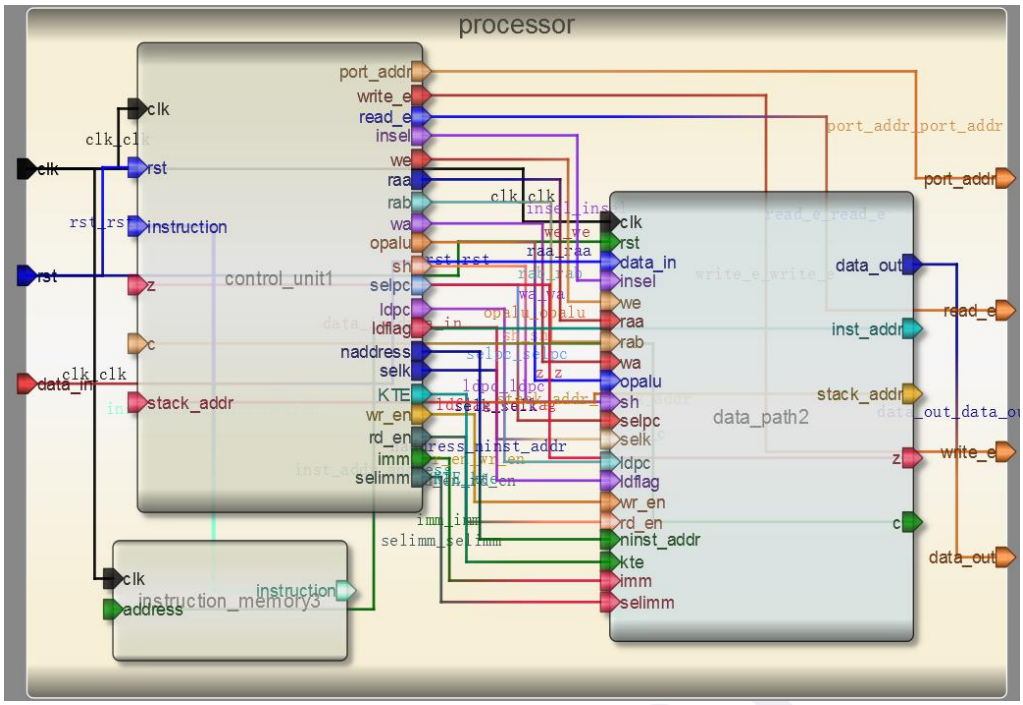


图 6-2-20 processor 界面图

（2）添加代码。点击模型下方的 Code 添加代码。

该模块和 data_path 模块类似，仅仅是把 control unit, data path 和 instruction memory 三个模块整合连接起来，所以并无代码。

（3）保存模型(存储文件夹路径不能有空格和中文)，运行并检查有无错误输出。

9. processor_test 测试文件的设计

（1）由于 Robei 软件目前尚不支持$readmemh()函数，故本设计的仿真测试在 Modelsim 中进行。

（2）将之前设计好的所有模块的源代码全部复制到一个文件夹中，并新建一个 Modelsim project。

（3）新建一个文件命名为“test_processor.v”，并将以下测试代码复制到文件中。

```
`timescale 1ns / 1ps
module testbench_processor();
reg clk_tb;
reg rst_tb;
reg [7:0] data_in_tb;
wire [7:0] port_addr_tb;
wire read_e_tb;
wire write_e_tb;
wire [7:0] data_out_tb;

processor processor_i(
    .clk(clk_tb),
    .rst(rst_tb),
    .port_addr(port_addr_tb),
```

 Copyright © Robei

```
        .read_e(read_e_tb),
        .write_e(write_e_tb),
        .data_in(data_in_tb),
        .data_out(data_out_tb)
        );
    initial begin
        clk_tb = 0;
        rst_tb = 1;
        data_in_tb = 0;
        #5 rst_tb = 0;
        #2000 $finish;
    end
    always #2 clk_tb = ~clk_tb;
    endmodule
```

（4）一定要将“2.4 汇编器脚本使用”中生成好的“instructions.mem”文件放置在该 project 所在的同一个目录中，然后才可以进行仿真。

（5）进行仿真并查看波形。根据之前设计的代码内容，查看分析仿真结果。

本次仿真使用的“test.asm”如下

```
    ldi r1, 22
    ldi r2, 80
    ldi r3, 36
    ldi r4, 45
    add r1, r2
    stm r1, 11
    add r3, r4
    stm r3, 25
    add r1, r3
    stm r1, 32
```

使用 Python 转化之后的结果如下图 6-2-21 所示

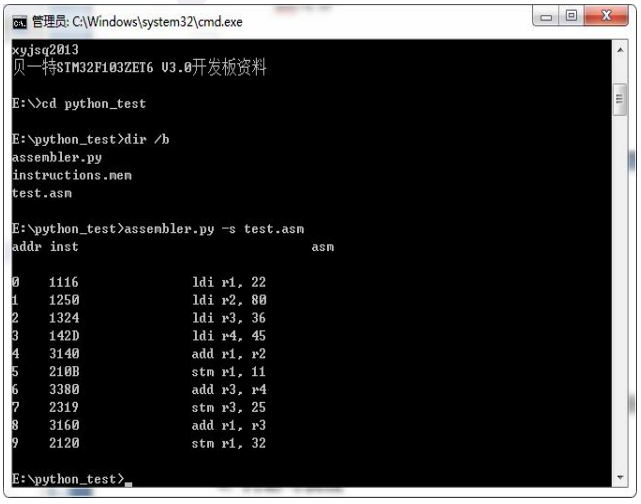


图 6-2-21 生成 imstruction.mem 文件

转换成“instructions.mem”之后，并进行仿真得到的波形如下图所示:

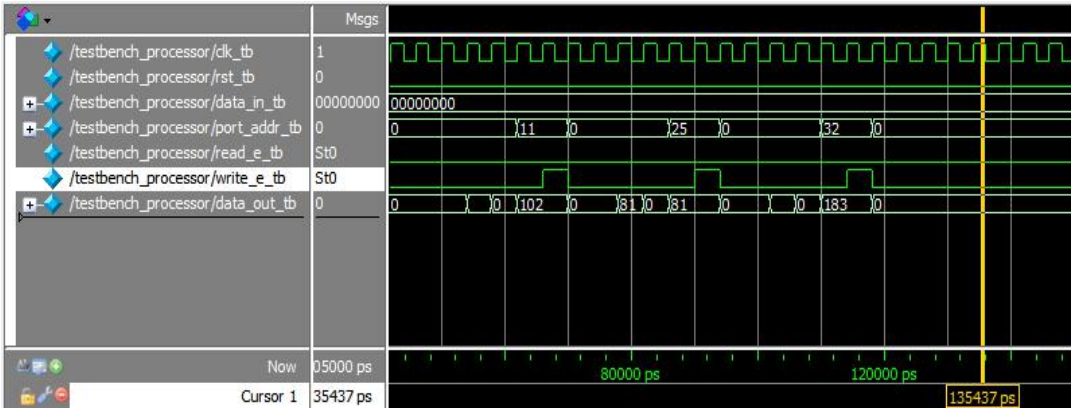


图 6-2-22 Modelsim 中的仿真结果

经验证，仿真结果正确无误。

6.2.3. 问题与挑战

可以尝试使用 Natalius 指令集中更多类型的指令，实现更加复杂的功能。

Copyright © Robei

第七章：总结反思，项目挑战

通过之前的学习，相信读者们已经对 FPGA 的设计有了完整的掌控和了解，但是，如果想要精通，就需要付出大量的时间和精力来尝试、实验、思考。今天的主要内容就是没有内容，我们留下一天的时间让读者来思考，纯粹的填鸭式学习是没有思考时间的，但是我们希望读者能够自己思考，查漏补缺，把之前的篇章重新复习一遍，认真理解自己在操作中的失误和代码编写中的语法错误。虽然我们书的篇章有限，但是学习是无止境的，读者可以到我们的官网查找更新的资料，也可以将自己的设计上传到 Robei 官方网站，与大家分享。Robei 是一个开放与共享的知识平台，书虽然结束了，Robei 却与你同在。

Robei

参考文献

[1] Simplified FPGA design with Robei, Guosheng Wu, Robei LLC, Henderson, NV, USA

[2] Verilog HDL 数字系统设计与验证, 乔庐峰, 北京：电子工业出版社，2009

[3] 基于 FPGA 的 VGA 控制器设计与实现, 杨杰，穆伟斌, 内蒙古：内蒙古出版社，2008

[4] Verilog HDL 高级数字设计(第二版). Micheal D.Ciletti 著, 李广军，林水生等译, 北京：电子工业出版社，2014

[5] Natalius 8 bit RISC Processor. Fabio Andres Guzman Figueroa. OpenCores，Jun 8，2012

[6] FPGA 数字逻辑设计教程——Verilog, Darrin M,Hanna, 北京：电子工业出版社，2010

[7] SPI 总线的 UART 扩展方法,《单片机与嵌入式系统应用》, 解书钢，马维华，吴术, 2008 年第 6 期

[8] FPGA 应用开发入门与典型案例, 姚远，李辰, 北京：人民邮电出版社，2010

[9] 单片机与 FPGA 逻辑接口的 Verilog 实现, 汲伟明，葛旭亮, 上海应用技术学院学报，2007 年 9 月，第三期，第七卷

[10] Verilog 数字系统设计教程, 夏宇闻, 北京：北京航空航天大学出版社，2003

[11]大型 RISC 处理器设计, 戈尔齐, 北京：北京航空航天大学出版社，2005.

Copyright © Robei

鸣　谢

非常感谢以下教授和专家在若贝的发展过程中给予的大力支持和帮助：

Paul Yih：美国 5CGroup 创始人，心理学博士，商业家。曾一人闯荡非洲、澳洲、南美洲、北美州等地，创造了一个又一个传奇，被称为真实版的印地安纳琼斯。Paul 也热衷于教育事业，用自己的亲身经历来引导学生不断超越自己。

于芳：中国科学院微电子研究所，硅器件与集成技术研发中心副总工程师，课题组长，博士士生导师，研究员，中科院微电子战略专家组成员，中国电子学会委员。长期从事 CMOS/SOI（SOS）加固器件、SRAM、FPGA 及数模混合集成电路设计、工艺及可靠性等技术研究，随着需要近些年也开展了实用化 FPGA 应用及开发软件的等技术研究。研究成果获中国科学院科技进步二等奖 2 项，三等奖 1 项。在国内外会议和刊物上发表多篇研究论文。

詹华群：江西科技师范大学通信与电子学院副院长。詹教授从事集成电路教育超过 12 年，有相当强的专业技术知识背景，对 FPGA 和 CPLD 有深层次的认识，并熟悉各种 EDA 设计工具。曾获江西省科技进步奖、南昌市科技进步奖、江西省教学成果奖，每年都指导学生参加全国及江西省电子设计比赛，并多次获奖，2013 年指导的学生获得 2 项全国电子设计比赛一等奖。

Robei