《EDA 技术实用教程》 第 3 版

科学出版社
潘松　　黄继业

目　录

第 1 章　概述

1.1　EDA 技术及其发展
1.2　EDA 技术实现目标
1.3　硬件描述语言 VHDL
1.4　VHDL 综合
1.5　基于 VHDL 的自顶向下设计方法
1.3　EDA 技术的优势
1.3　EDA 的发展趋势
【习题】

第 2 章　EDA 设计流程及其工具

2.1　设计流程
2.1.1　设计输入(原理图 / HDL 文本编辑)
2.1.2　综合
2.1.3　适配
2.1.4　时序仿真与功能仿真
2.1.5　编程下载
2.1.6　硬件测试
2.2　ASIC 及其设计流程
2.2.1　ASIC 设计方法
2.2.2　一般 ASIC 设计的流程
2.3　常用 EDA 工具
2.3.1　设计输入编辑器
2.3.2　HDL 综合器
2.3.3　仿真器
2.3.4　适配器
2.3.5　下载器
2.4　QuartusII 简介
2.5　IP 核简介
【习题】

第 3 章　FPGA/CPLD 结构与应用

3.1　概述
3.1.1　可编程逻辑器件的发展历程
3.1.2　可编程逻辑器件的分类
3.2　简单可编程逻辑器件原理
3.2.1　电路符号表示
3.2.2　PROM
3.2.3　PLA

3.2.4 PAL
3.2.5 GAL
3.3 CPLD 的结构与工作原理
3.4 FPGA 的结构与工作原理
3.4.1 查找表逻辑结构
3.4.2 Cyclone/CycloneII 系列器件的结构与原理
3.5 硬件测试技术
3.5.1 内部逻辑测试
3.5.2 JTAG 边界扫描测试
3.5.3 嵌入式逻辑分析仪
3.6 FPGA/CPLD 产品概述
3.6.1 Lattice 公司 CPLD 器件系列
3.6.2 Xilinx 公司的 FPGA 和 CPLD 器件系列
3.6.3 Altera 公司 FPGA 和 CPLD 器件系列
3.6.4 Actel 公司的 FPGA 器件
3.6.5 Altera 公司的 FPGA 配置方式与配置器件
3.7 编程与配置
3.7.1 JTAG 方式的在系统编程
3.7.2 使用 PC 并行口配置 FPGA
3.7.3 FPGA 专用配置器件
3.7.4 使用单片机配置 FPGA
3.7.5 使用 CPLD 配置 FPGA
【习题】
第 4 章 VHDL 设计初步
4.1 多路选择器的 VHDL 描述
4.1.1 2 选 1 多路选择器的 VHDL 描述
4.1.2 相关语句结构和语法说明
4.2 寄存器描述及其 VHDL 语言现象
4.2.1 D 触发器的 VHDL 描述
4.2.2 VHDL 描述的语言现象说明
4.2.3 实现时序电路的 VHDL 不同表述
4.2.4 异步时序电路设计
4.3 1 位二进制全加器的 VHDL 描述
4.3.1 半加器描述
4.3.2 CASE 语句
4.3.3 全加器描述和例化语句
4.4 计数器设计
4.4.1 4 位二进制加法计数器设计
4.4.2 整数类型
4.4.3 计数器设计的其他表述方法
4.5 一般加法计数器设计
4.5.1 相关语法说明
4.5.2 程序分析
4.5.3 含并行置位的移位寄存器设计

【习题】
第 5 章　QuartusII 应用向导
5.1　基本设计流程
5.1.1　建立工作库文件夹和编辑设计文件
5.1.2　创建工程
5.1.3　编译前设置
5.1.4　全程编译
5.1.5　时序仿真
5.1.6　应用 RTL 电路图观察器
5.2　引脚设置和下载
5.2.1　引脚锁定
5.2.2　配置文件下载
5.2.3　AS 模式编程配置器件
5.2.3　JTAG 间接模式编程配置器件
5.2.3　USB Blaster 编程配置器件使用方法
5.3　嵌入式逻辑分析仪使用方法
5.4　原理图输入设计方法
5.4.1　设计流程
5.4.2　应用宏模块的原理图设计
【习题】
【实验与设计】
5-1. 组合电路的设计
5-2. 时序电路的设计
5-3. 设计含异步清 0 和同步时钟使能的加法计数器
5-4. 用原理图输入法设计 8 位全加器
5-5. 用原理图输入法设计较复杂数字系统
第 6 章　VHDL 设计进阶
6.1　数据对象
6.1.1　常数
6.1.2　变量
6.1.3　信号
6.1.4　进程中的信号与变量赋值
6.2　双向和三态电路信号赋值例解
6.2.1　三态门设计
6.2.2　双向端口设计
6.2.3　三态总线电路设计
6.3　IF 语句概述
6.4　进程语句归纳
6.4.1　进程语句格式
6.4.2　进程结构组成
6.4.3　进程要点
6.5　并行语句例解
6.6　仿真延时
6.6.1　固有延时

6.6.2 传输延时
6.6.3 仿真 δ
【习题】
【实验与设计】
6-1. 七段数码显示译码器设计
6-2. 八位数码扫描显示电路设计
6-3. 数控分频器的设计
6-4. 32 位并进/并出移位寄存器设计
第 7 章 宏功能模块与 IP 应用
7.1 宏功能模块概述
7.1.1 知识产权核的应用
7.1.2 使用 MegaWizard Plug-In Manager
7.1.3 在 **QuartusII** 中对宏功能模块进行例化
7.2 宏模块应用实例
7.2.1 工作原理
7.2.2 定制初始化数据文件
7.2.3 定制 LPM_ROM 元件
7.2.4 完成顶层设计
7.3 在系统存储器数据读写编辑器应用
7.4 编辑 SignalTapII 的触发信号
7.5 其它存储器模块的定制与应用
7.5.1 RAM 定制
7.5.2 FIFO 定制
7.6 流水线乘法累加器的混合输入设计
7.7 LPM 嵌入式锁相环调用
7.7.1 建立嵌入式锁相环元件
7.7.2 测试锁相环
7.8 IP 核 NCO 数控振荡器使用方法
7.9 8051 单片机 IP 核应用
【习题】
【实验与设计】
7-1. 正弦信号发生器设计
7-2. 八位 16 进制频率计设计
7-3. 利用 LPM_ROM 设计乘法器
7-4 IP 核应用实验
7-5 8051 单片机 IP 核应用实验
第 8 章 状态机设计
8.1 一般有限状态机设计
8.1.1 数据类型定义语句
8.1.2 为什么要使用状态机
8.1.3 一般有限状态机的设计
8.2 Moore 型有限状态机设计
8.2.1 多进程有限状态机
8.2.2 单进程 Moore 型有限状态机

8.3 Mealy 型有限状态机设计
8.4 状态编码
8.4.1 状态位直接输出型编码
8.4.2 顺序编码
8.4.3 一位热码编码
8.5 非法状态处理
【习题】
【实验与设计】
8-1. 序列检测器设计
8-2. ADC0809 采样控制电路实现
8-3. 数据采集电路和简易存储示波器设计
8-4. 比较器和 D/A 器件实现 A/D 转换功能的电路设计
8-5. 通用异步收发器设计
第 9 章 VHDL 结构与要素
9.1 实体
9.1.1 实体语句结构
9.1.2 参数传递说明语句
9.1.3 参数传递映射语句
9.1.4 端口说明语句
9.2 结构体
9.3 子程序
9.3.1 函数
9.3.2 重载函数
9.3.3 转换函数
9.3.4 决断函数
9.3.5 过程
9.3.6 重载过程
9.4 VHDL 库
9.4.1 库的种类
9.4.2 库的用法
9.5 程序包
9.6 配置
9.7 VHDL 文字规则
9.7.1 数字
9.7.2 字符串
9.7.3 标识符
9.7.4 下标名
9.8 数据类型
9.8.1 预定义数据类型
9.8.2 IEEE 预定义标准逻辑位与矢量
9.8.3 其他预定义标准数据类型
9.8.4 数组类型
9.9 操作符
9.9.1 逻辑操作符

9.9.2　关系操作符
9.9.3　算术操作符
【习题】
【实验与设计】
9-1　乐曲硬件演奏电路设计
9-2　乒乓球游戏电路设计
9-3　采用高速 A/D 的存储示波器设计
9-4　循环冗余校验(CRC)模块设计
第 10 章　VHDL 基本语句
10.1　顺序语句
10.1.1　赋值语句
10.1.2　IF 语句
10.1.3　CASE 语句
10.1.4　LOOP 语句
10.1.5　NEXT 语句
10.1.6　EXIT 语句
10.1.7　WAIT 语句
10.1.8　子程序调用语句
10.1.9　RETURN 语句
10.1.10　空操作语句
10.2　并行语句
10.2.1　并行信号赋值语句
10.2.2　块语句结构
10.2.3　并行过程调用语句
10.2.4　元件例化语句
10.2.5　生成语句
10.2.6　REPORT 语句
10.2.7　断言语句
10.3　属性描述与定义语句
【习题】
【实验与设计】
10-1　移位相加硬件乘法器设计
10-2　等精度频率计/相位计设计
10-3　基于 8051 单片机 IP 核的等精度频率计单片系统设计（LCD 显示）
10-4　基于 8051 单片机 IP 核的等精度频率计单片系统设计（LED 显示）
第 11 章　优化和时序分析
11.1　资源优化
11.1.1　资源共享
11.1.2　逻辑优化
11.1.3　串行化
11.2　速度优化
11.2.1　流水线设计
11.2.2　寄存器配平
11.2.3　关键路径法

11.3 优化设置与时序分析
11.3.1 Settings 设置
11.3.2 HDL 版本设置及 Analysis & Synthesis 功能
11.3.3 Analysis & Synthesis 的优化设置
11.3.4 适配器 Fitter 设置
11.3.5 增量布局布线控制设置
11.3.6 使用 **Design Assistant** 检查设计可靠性
11.3.7 时序设置与分析
11.3.8 查看时序分析结果
11.3.9 适配优化设置示例
11.3.10 Slow Slew Rate 设置
11.3.11 LogicLock 优化技术
11.4 Chip Editor 应用
11.4.1 Chip Editor 应用实例
11.4.2 Chip Editor 功能说明
11.4.3 利用 Change Manager 检测底层逻辑
【习题】
【实验与设计】
11-1 采用流水线技术设计高速数字相关器
11-2 线性反馈移位寄存器设计
11-3 直接数字式频率合成器(DDS)设计
11-4 基于 DDS 的数字移相信号发生器设计
11-5 基于 DDS 的幅度调制 AM 信号发生器设计
11-6 频率调制 FM 信号发生器设计
第 12 章 系统仿真
12.1 仿真
12.2 VHDL 源程序仿真
12.3 仿真激励信号的产生
12.4 VHDL 测试基准
12.5 VHDL 系统级仿真
12.6 使用 ModelSim 进行仿真
12.7 VHDL 的 RTL 表述
12.7.1 行为描述
12.7.2 数据流描述
12.7.3 结构描述
第 13 章 电子系统设计实践
13.1 VGA 彩条信号显示控制器设计
13.2 VGA 图象显示控制器设计
13.3 步进电机细分驱动控制
13.4 直流电机的 PWM 控制
【习题】
【实验与设计】
13-1. VGA 彩条信号显示控制器设计
13-2. VGA 图像显示控制器设计

13-3. 步进电机细分驱动控制实验
13-4. 直流电机 PWM 控制实验
参考文献
附录

EDA 技术实用教程

第 1 章　概　述

1.1 EDA技术及其发展

EDA (Electronic Design Automation)

EDA技术发展的三个阶段

20世纪70年代	MOS工艺　CAD概念
20世纪80年代	CMOS时代　出现 FPGA
20世纪90年代	ASIC设计技术　EDA技术

1.1 EDA技术及其发展

EDA技术在进入21世纪后，得到了更大的发展：

- 电子设计成果　自主知识产权
- 仿真和设计　EDA软件不断推出
- 电子技术全方位纳入EDA领域　传统设计建模理念发生重大变化
- EDA使得电子领域各学科的界限更加模糊　更加互为包容
- 更大规模的FPGA和CPLD器件的不断推出
- EDA工具　ASIC设计　涵盖大规模电子系统及复杂IP核模块
- 软硬件IP核在电子行业广泛应用　IP－Intellectual Property
- SoC高效低成本设计技术的成熟
- 硬件描述语言出现（如System C）　设计和验证趋于简单

1.2 EDA技术实现目标

目标：是完成专用集成电路ASIC的设计和实现

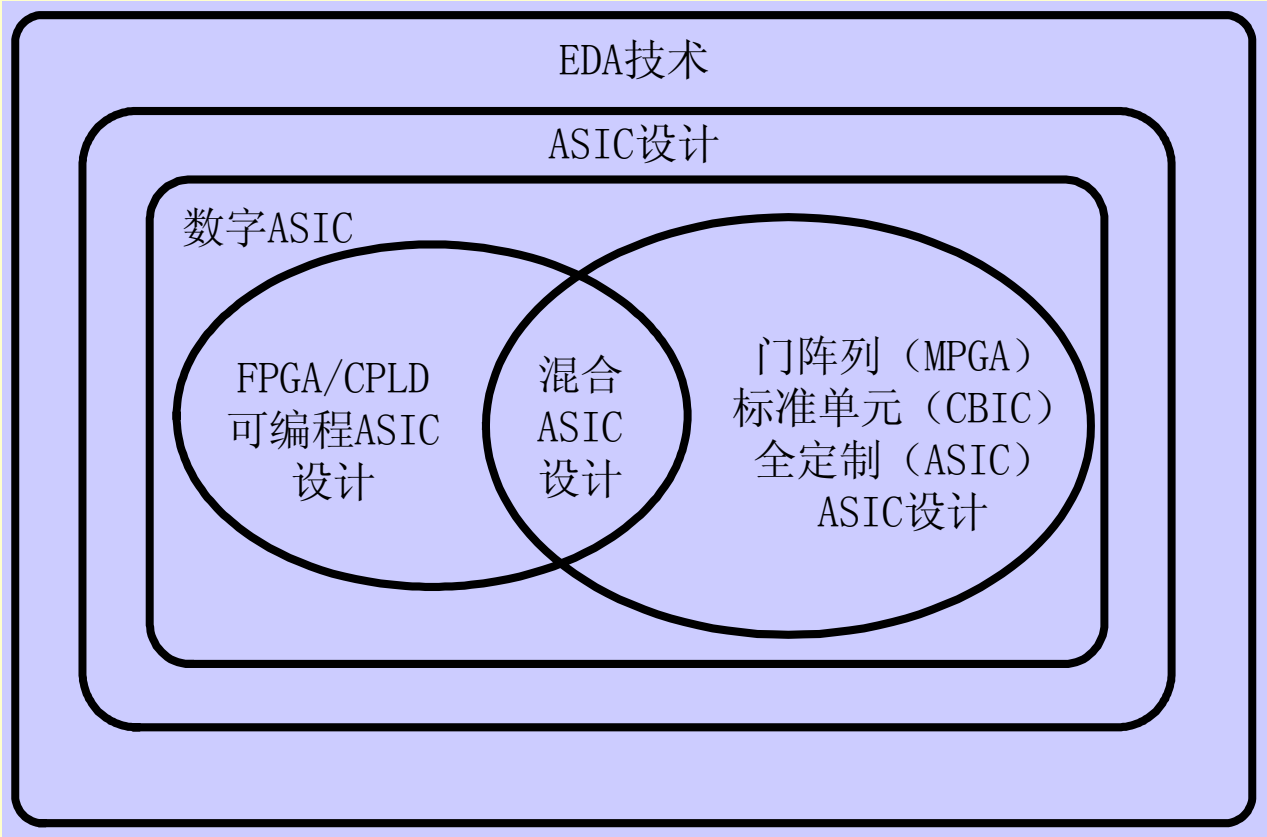


图1-1 EDA技术实现目标

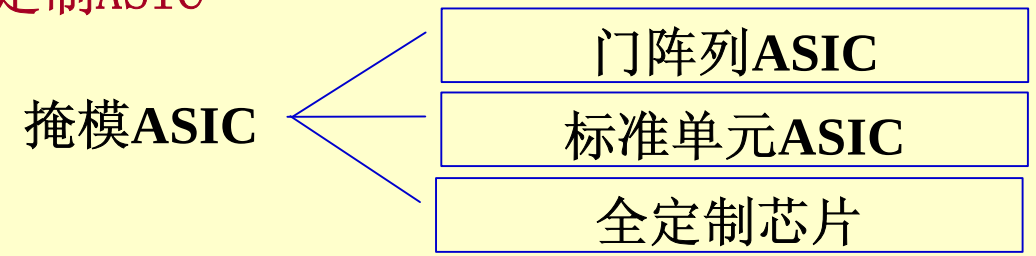
1.2 EDA技术实现目标

1. 超大规模可编程逻辑器件

FPGA(Field Programmable Gate Array)

CPLD(Complex Programmable Logic Device)

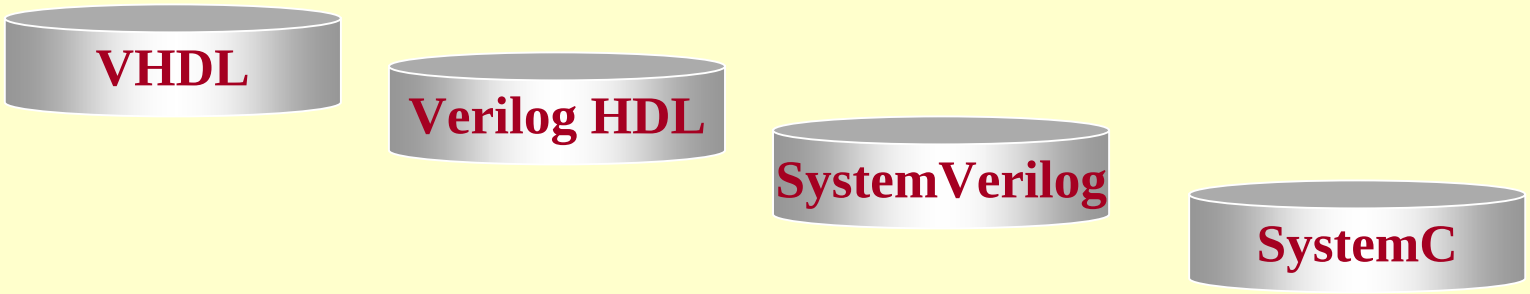
2. 半定制或全定制ASIC



3. 混合ASIC

CPU、RAM、ROM、硬件加法器、乘法器、锁相环

1.3 硬件描述语言VHDL



VHDL－

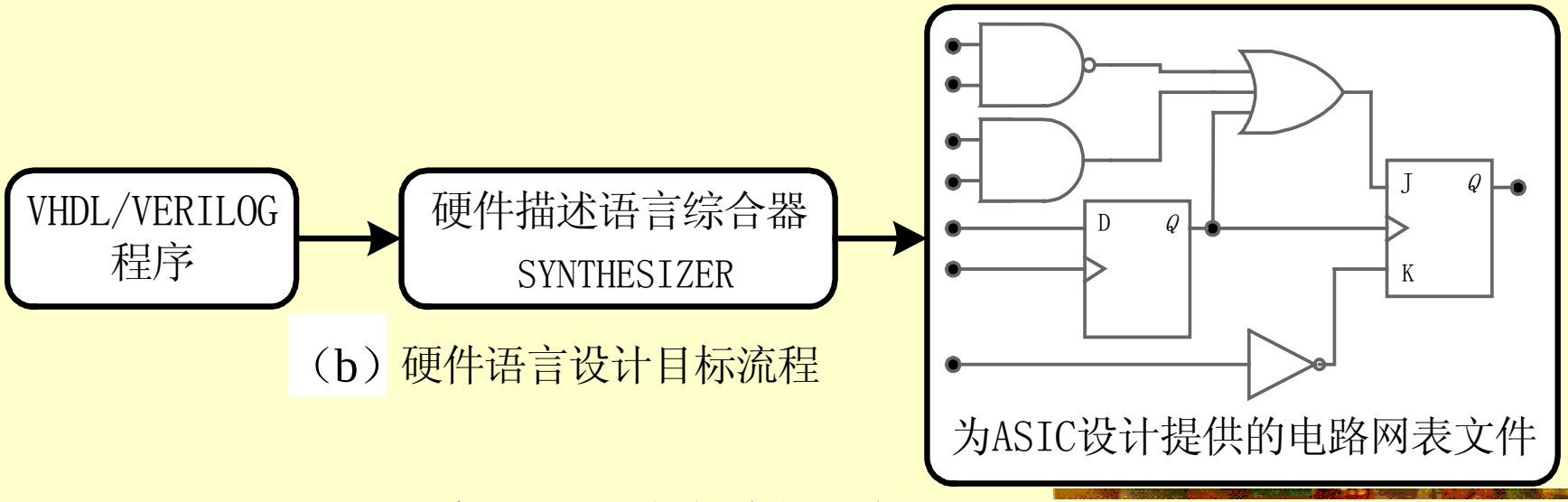
VHSIC(Very High Speed Integrated Circuit)Hardware Description Language

- 具有很强的电路描述和建模能力
- 具有与具体硬件电路无关和与设计平台无关的特性
- 具有良好的电路行为描述和系统描述的能力

1.4 VHDL综合

把抽象的实体结合成单个或统一的实体。

（a）软件语言设计目标流程



（b）硬件语言设计目标流程

图1-2 编译器和综合功能比较

1.4 VHDL综合

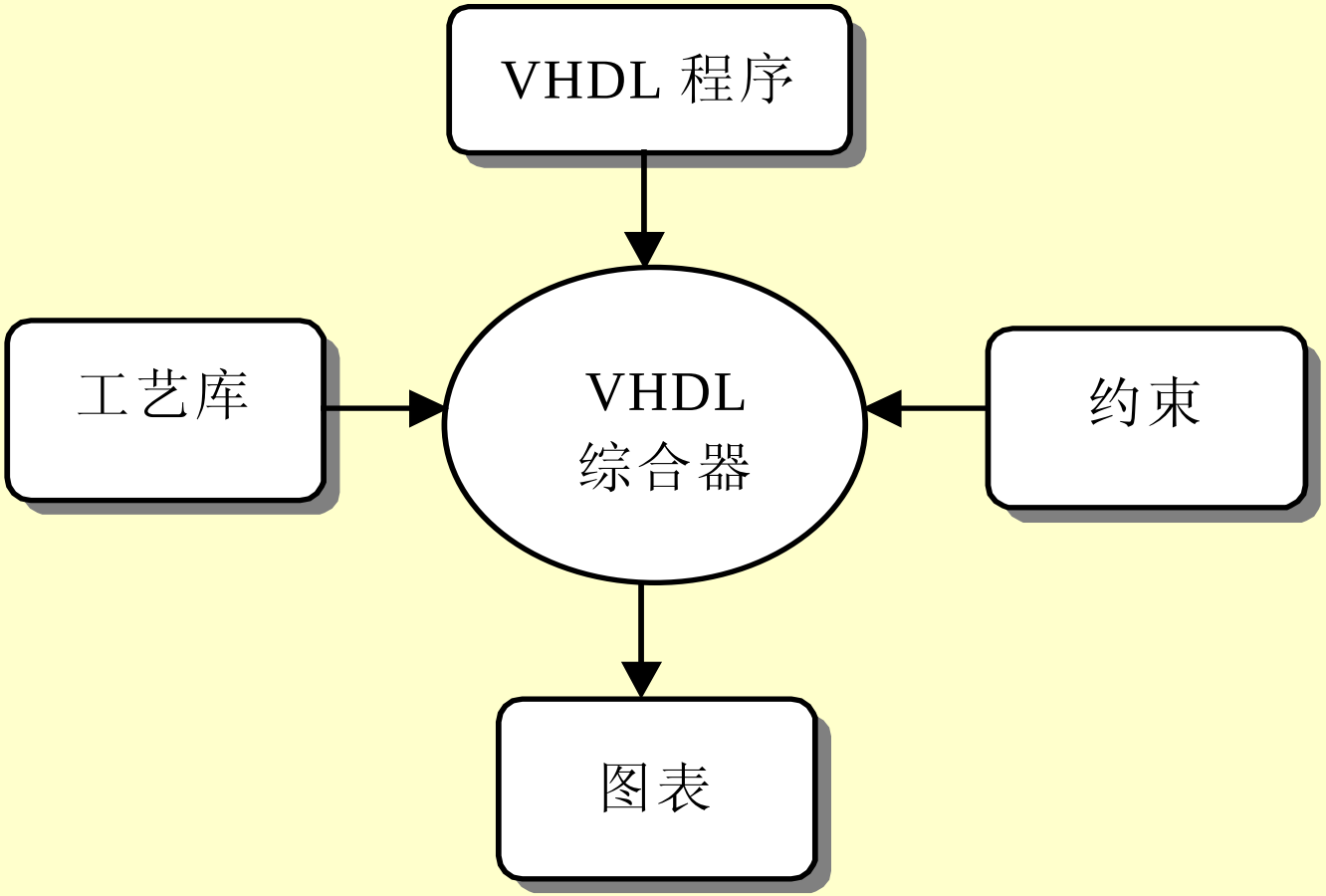


图1-3 VHDL综合器运行流程

1.5 基于VHDL的自顶向下设计方法

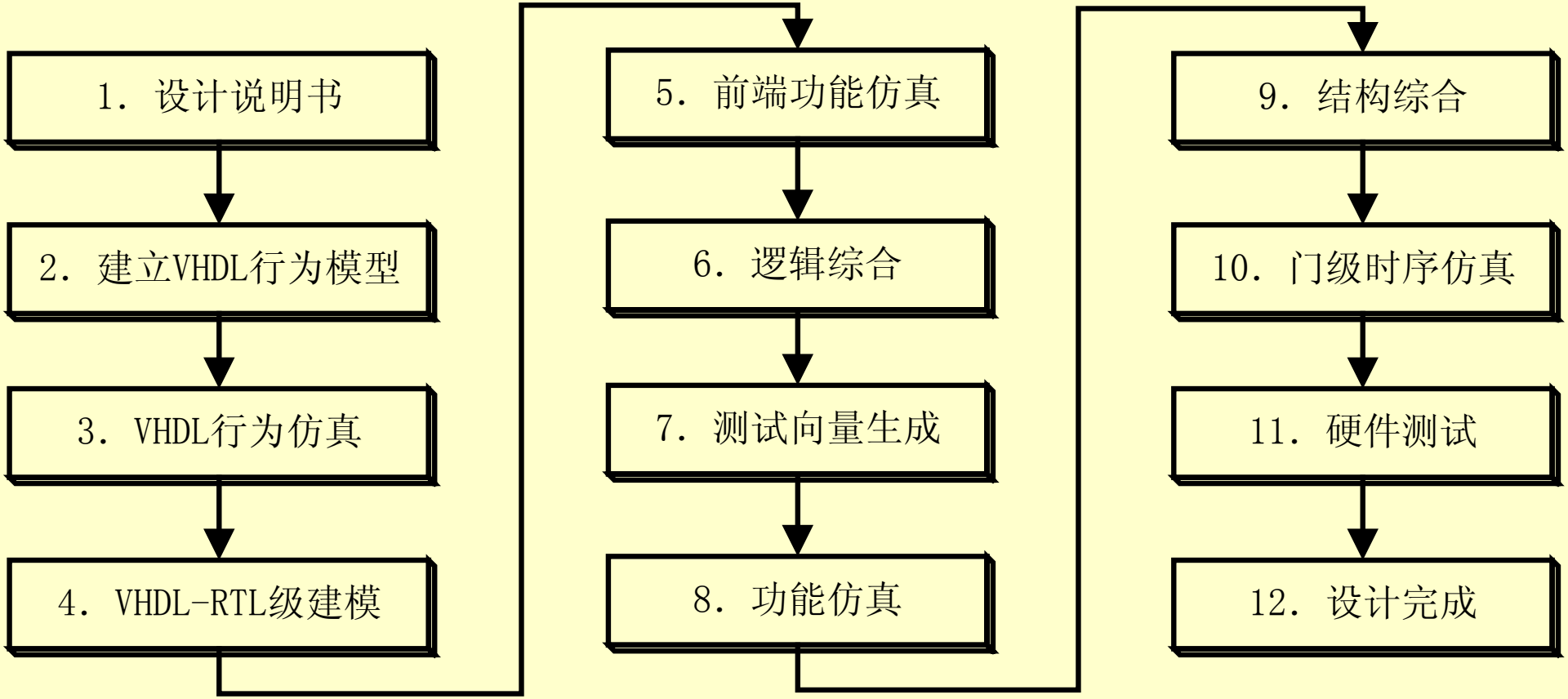


图1-4 自顶向下的设计流程

KX 康芯科技

1.6 EDA技术的优势

- 可以在电子设计的各个阶段、各个层次进行计算机模拟验证
- 有各类库的支持
- 某些HDL语言也是文档型的语言(如VHDL)
- 日益强大的逻辑设计仿真测试技术
- 设计者拥有完全的自主权，再无受制于人之虞
- 良好的可移植与可测试性，为系统开发提供了可靠的保证
- 能将所有设计环节纳入统一的自顶向下的设计方案中
- 自动设计能力、不同内容的仿真模拟、完整的测试

1.7 EDA的发展趋势

- 在一个芯片上完成的系统级的集成已成为可能
- 可编程逻辑器件开始进入传统的**ASIC**市场
- **EDA**工具和**IP**核应用更为广泛
- 高性能的**EDA**工具得到长足的发展
- 计算机硬件平台性能大幅度提高，为复杂的**SoC**设计提供了物理基础。

习 题

1-1 **EDA技术与ASIC设计和FPGA开发有什么关系？**

1-2 与软件描述语言相比，**VHDL**有什么特点？

1-3 什么是综合？有那些类型？综合在电子设计自动化中的地位是什么？

1-4 在**EDA**技术中，自顶向下的设计方法的重要意义是什么？

1-5 **IP**在**EDA**技术的应用和发展中的意义是什么？

EDA 技术实用教程

第 2 章

EDA设计流程及其工具

2.1 设计流程

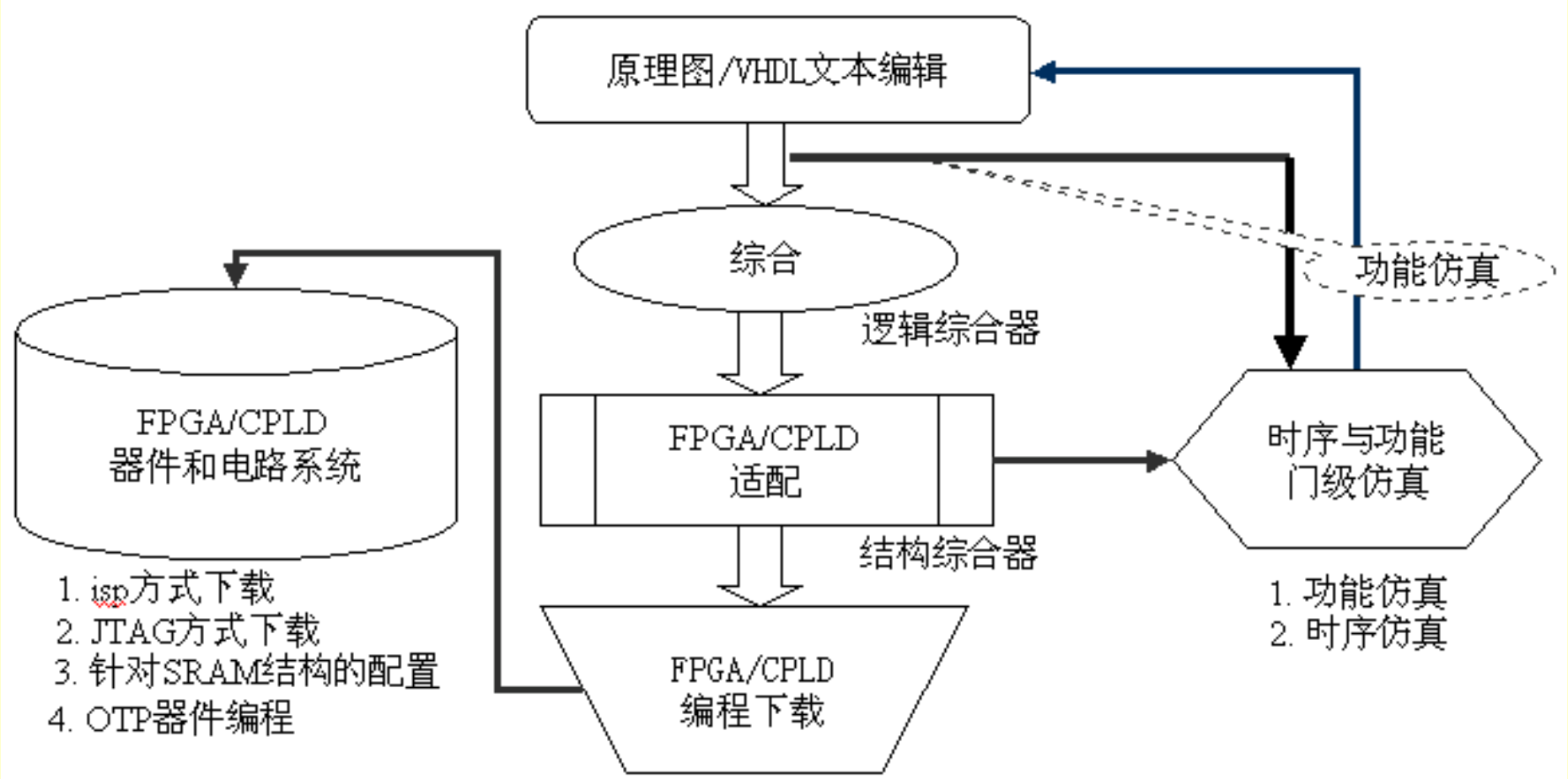


图2-1 应用于FPGA/CPLD的EDA开发流程

KX 康芯科技

2.1 设计流程

2.1.1 设计输入(原理图 / HDL文本编辑)

1. 图形输入

状态图输入　　波形图输入　　原理图输入

在EDA软件的图形编辑界面上绘制能完成特定功能的电路原理图

2. HDL文本输入

将使用了某种硬件描述语言(HDL)的电路设计文本，

如VHDL或Verilog的源程序，进行编辑输入。

2.1 设计流程

2.1.2 综合

整个综合过程就是将设计者在EDA平台上编辑输入的HDL文本、原理图或状态图形描述，依据给定的硬件结构组件和约束控制条件进行编译、优化、转换和综合，最终获得门级电路甚至更底层的电路描述网表文件。

2.1.3 适配

将由综合器产生的网表文件配置于指定的目标器件中，使之产生最终的下载文件，如JEDEC、Jam格式的文件。

2.1 设计流程

2.1.4 时序仿真与功能仿真

时序仿真

接近真实器件运行特性的仿真

功能仿真

直接对VHDL、原理图描述或其他描述形式的逻辑功能进行测试模拟

2.1.5 编程下载

2.1.6 硬件测试

2.2 ASIC及其设计流程

ASIC(Application Specific Integrated Circuits，专用集成电路)

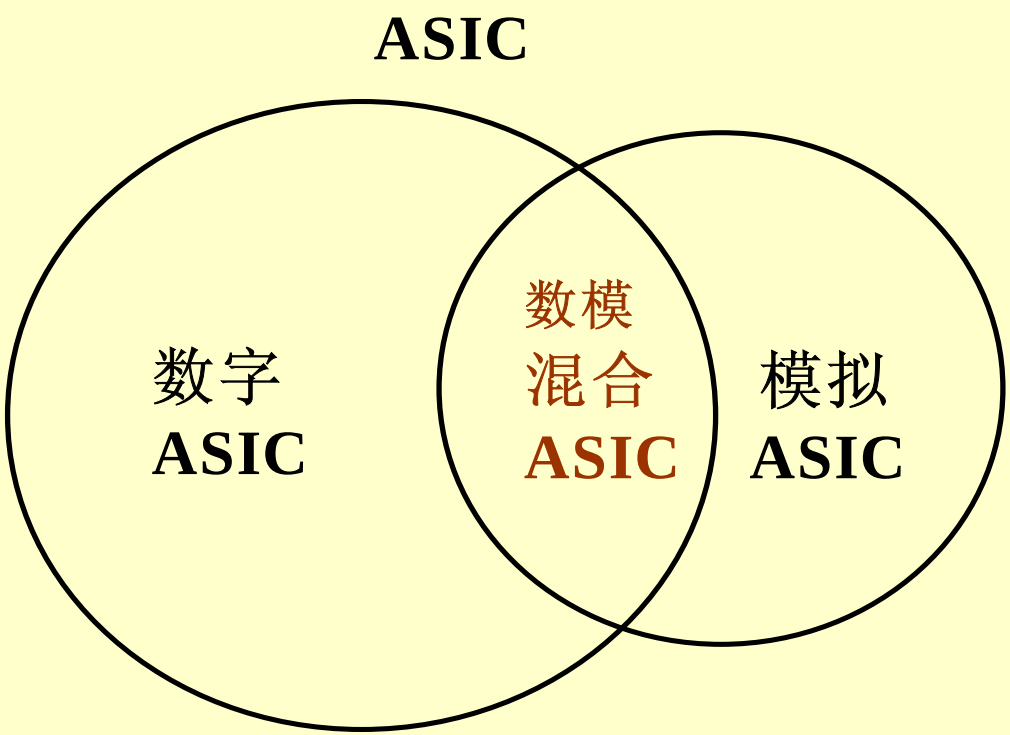


图2-2 ASIC分类

2.2 ASIC及其设计流程

2.2.1 ASIC设计方法

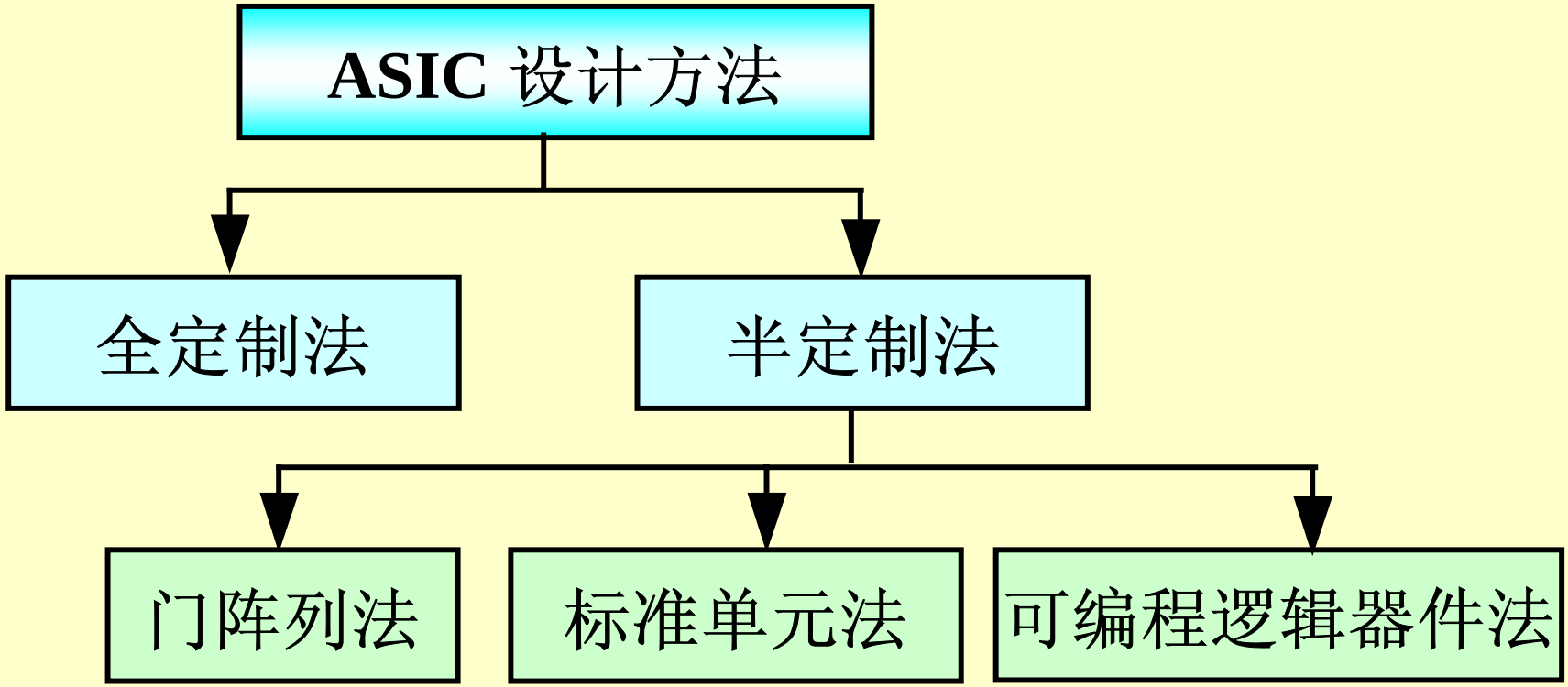


图2-3 ASIC实现方法

2.2.2 一般ASIC设计的流程

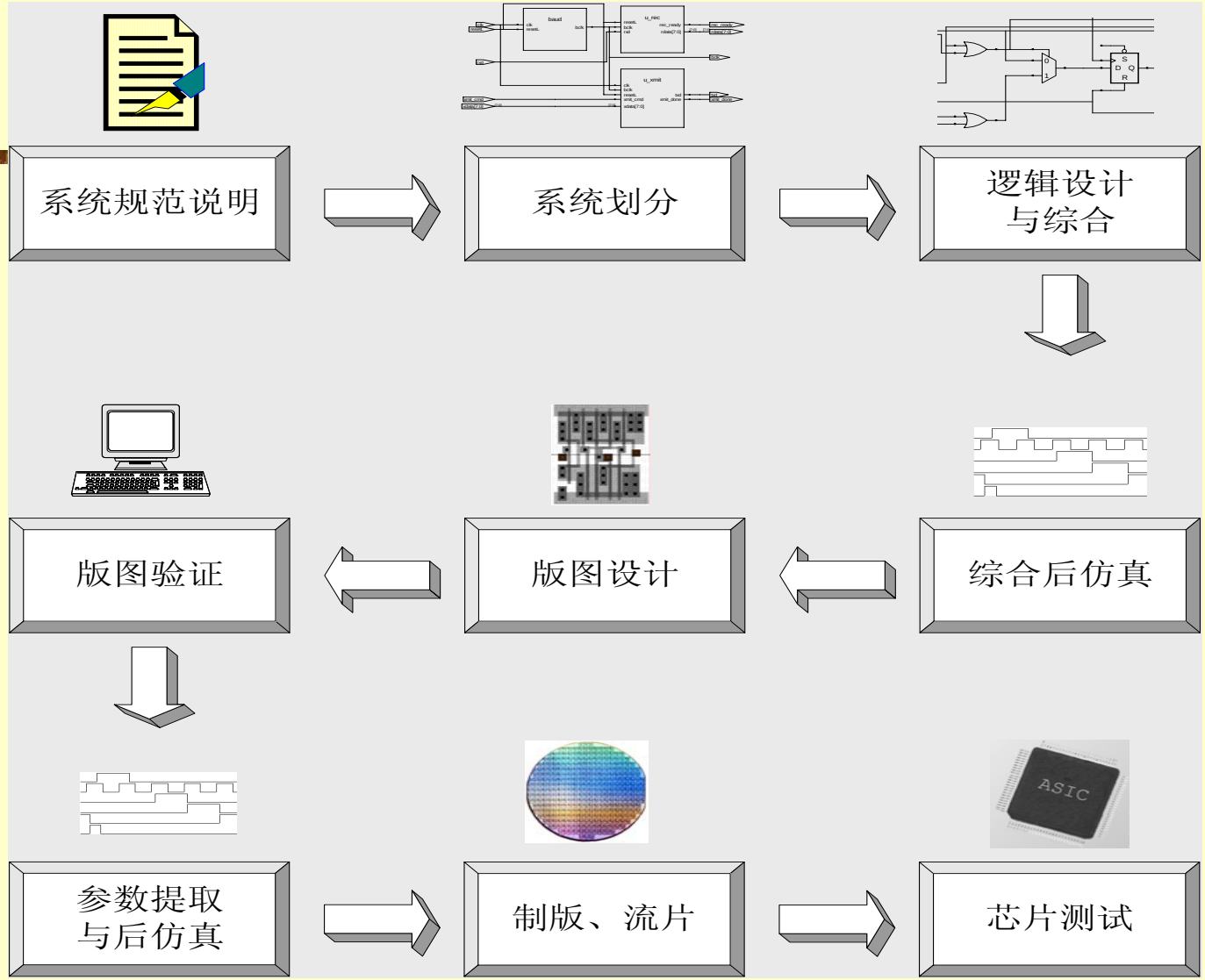


图2-4 ASIC设计流程

2.3 常用EDA工具

2.3.1 设计输入编辑器

2.3.2 HDL综合器

FPGA Compiler II、DC-FPGA综合器、Synplify Pro综合器、LeonardoSpectrum综合器和Precision RTL Synthesis综合器

2.3.3 仿真器

VHDL仿真器　Verilog仿真器

Mixed HDL仿真器　其他HDL仿真器

2.3.4 适配器

2.3.5 下载器

2.4 QuartusII 简介

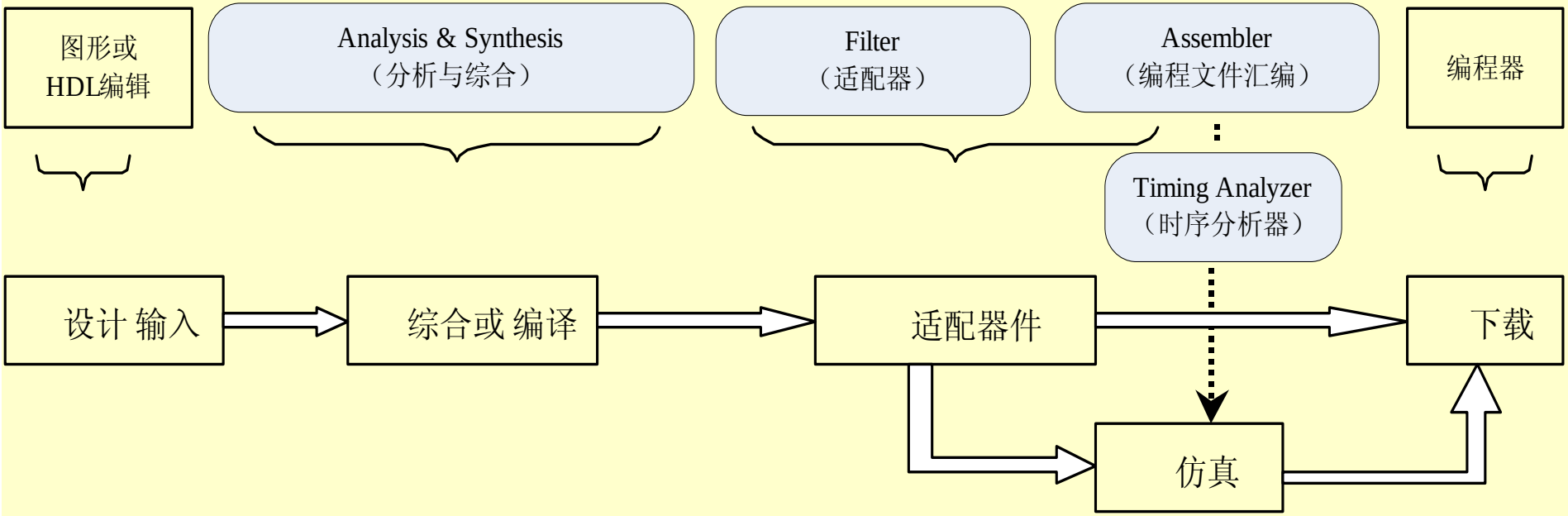


图1-9 Quartus II设计流程

2.5 IP核简介

IP (Intellectual Property)

习 题

1-1 叙述EDA的FPGA/CPLD设计流程。

1-2 IP是什么？IP与EDA技术的关系是什么？

1-3 叙述ASIC的设计方法。

1-4 FPGA/CPLD在ASIC设计中有什么用处？

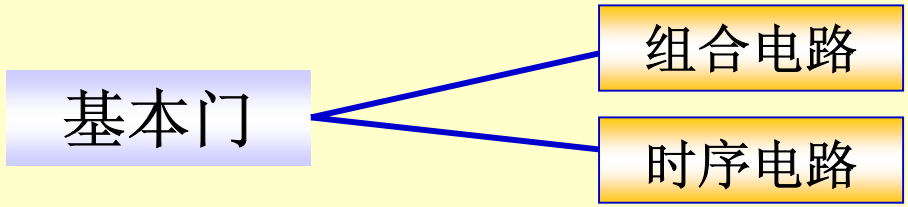
1-5 简述在基于FPGA/CPLD的EDA设计流程中所涉及的EDA工具，及其在整个流程中的作用。

EDA 技术实用教程

第 3 章

FPGA/CPLD 结构与应用

3.1 概　述



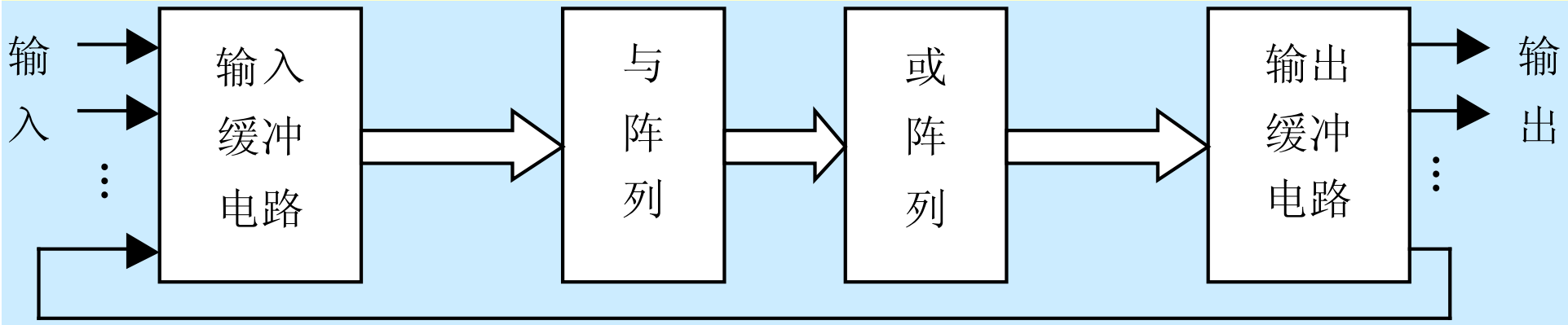
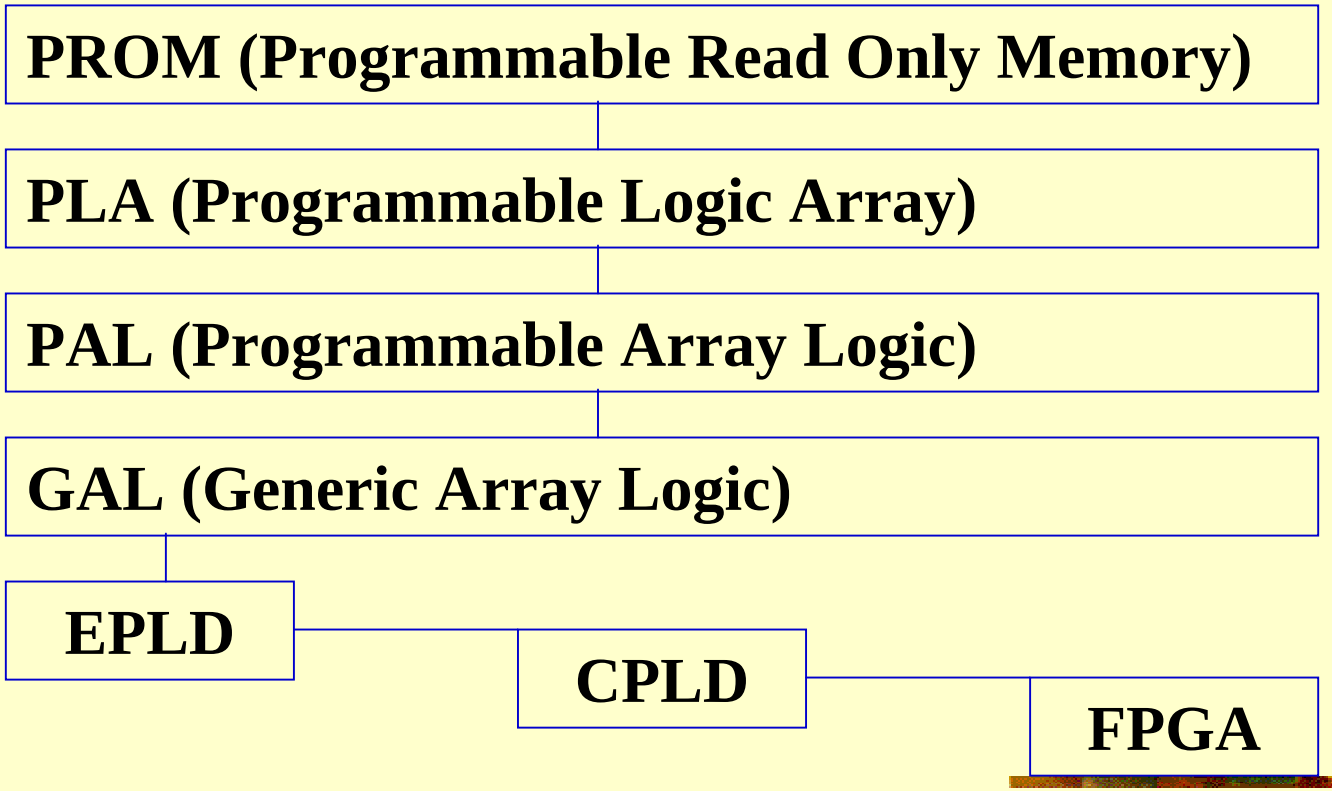


图3-1 基本PLD器件的原理结构图

3.1 概　述

3.1.1 可编程逻辑器件的发展历程



3.1 概　述

3.1.2 可编程逻辑器件的分类

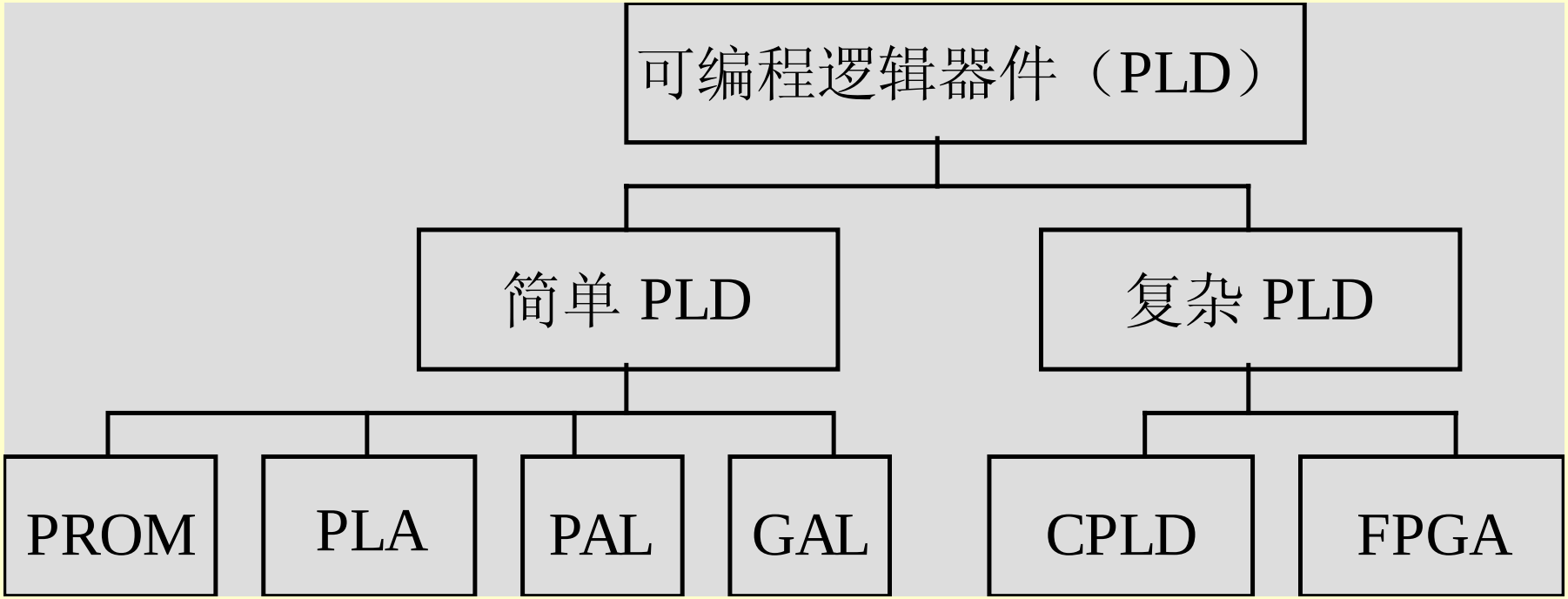


图3-2 PLD按集成度分类

3.2 简单可编程逻辑器件原理

3.2.1 电路符号表示

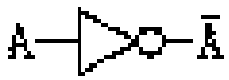
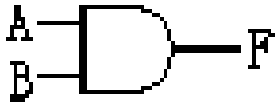
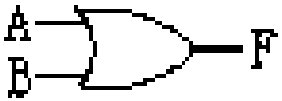
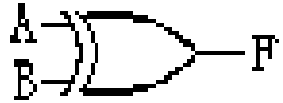
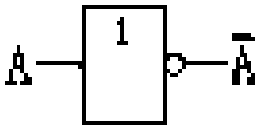
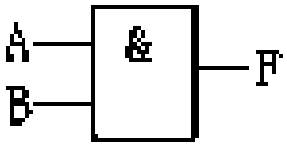
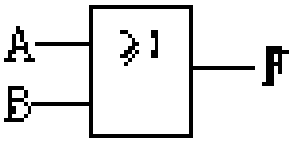
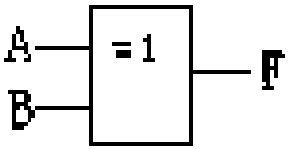
	非门	与门	或门	异或门
常用符号				
国标符号				
逻辑表达式	$\bar{A}$= NOT A	F=A•B	F=A+B	F=A⊕B

图3-3 常用逻辑门符号与现有国标符号的对照

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.1 电路符号表示

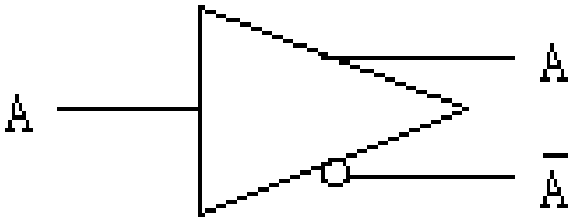


图3-4 PLD的互补缓冲器

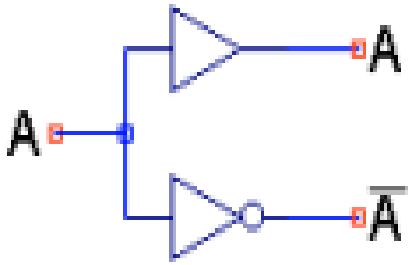


图3-5 PLD的互补输入

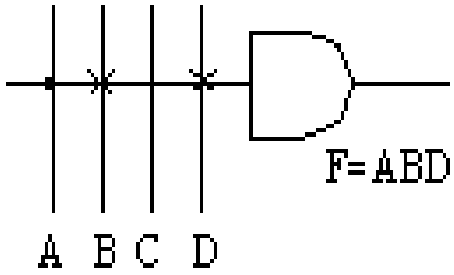


图3-6 PLD中与阵列表示

3.2 简单可编程逻辑器件原理

3.2.1 电路符号表示

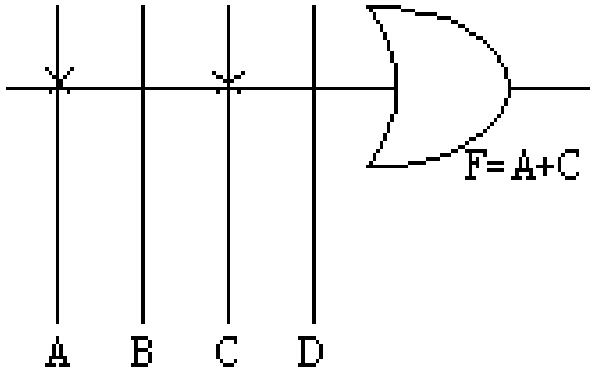


图3-7 PLD中或阵列的表示

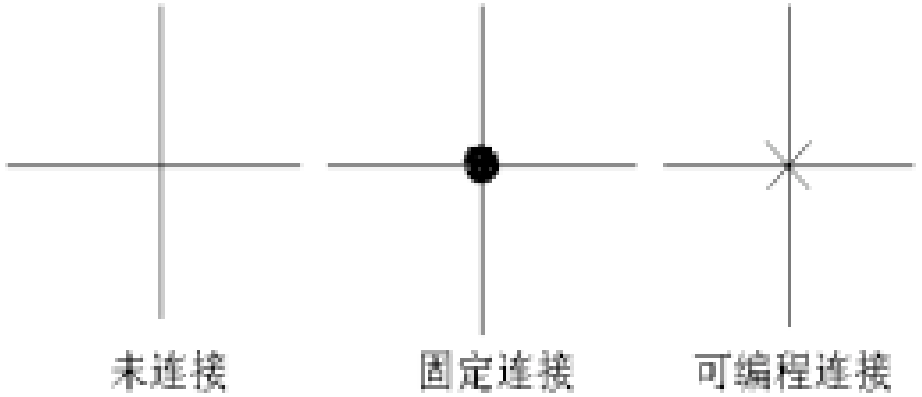


图3-8 阵列线连接表示

3.2 简单可编程逻辑器件原理

3.2.2 PROM

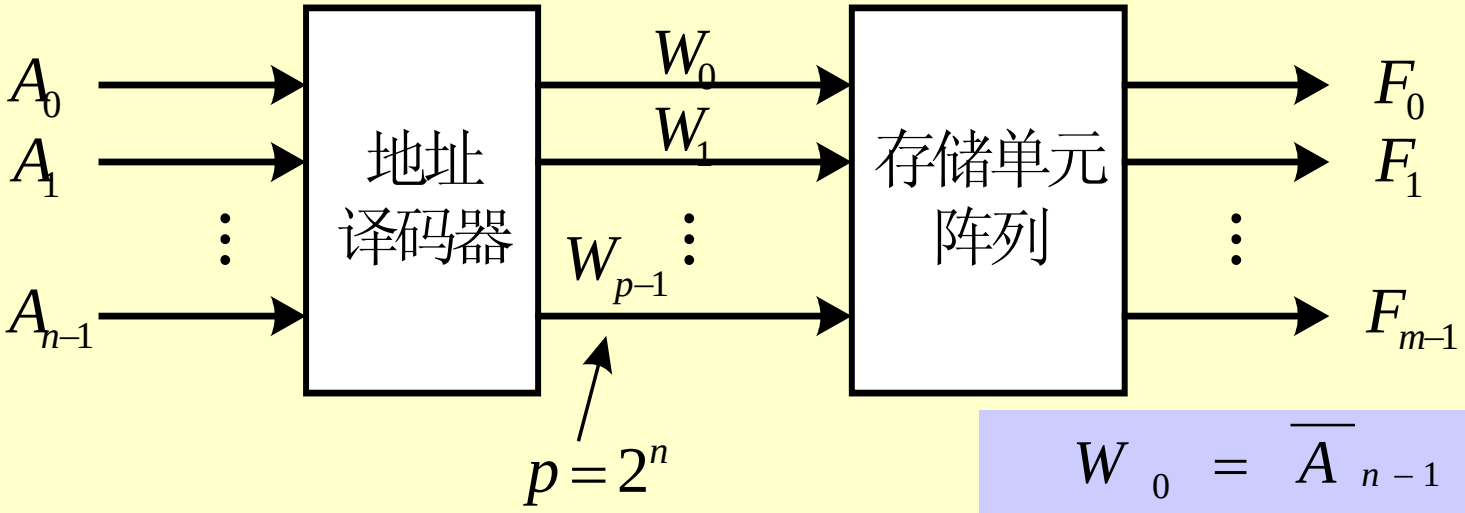


图3-9 PROM基本结构

$$W_0 = \overline{A}_{n-1} \cdots \overline{A}_1 \overline{A}_0$$

$$W_1 = \overline{A}_{n-1} \cdots \overline{A}_1 A_0$$

$$\cdots$$

$$W_{2^n-1} = A_{n-1} \cdots A_1 A_0$$

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.2 PROM

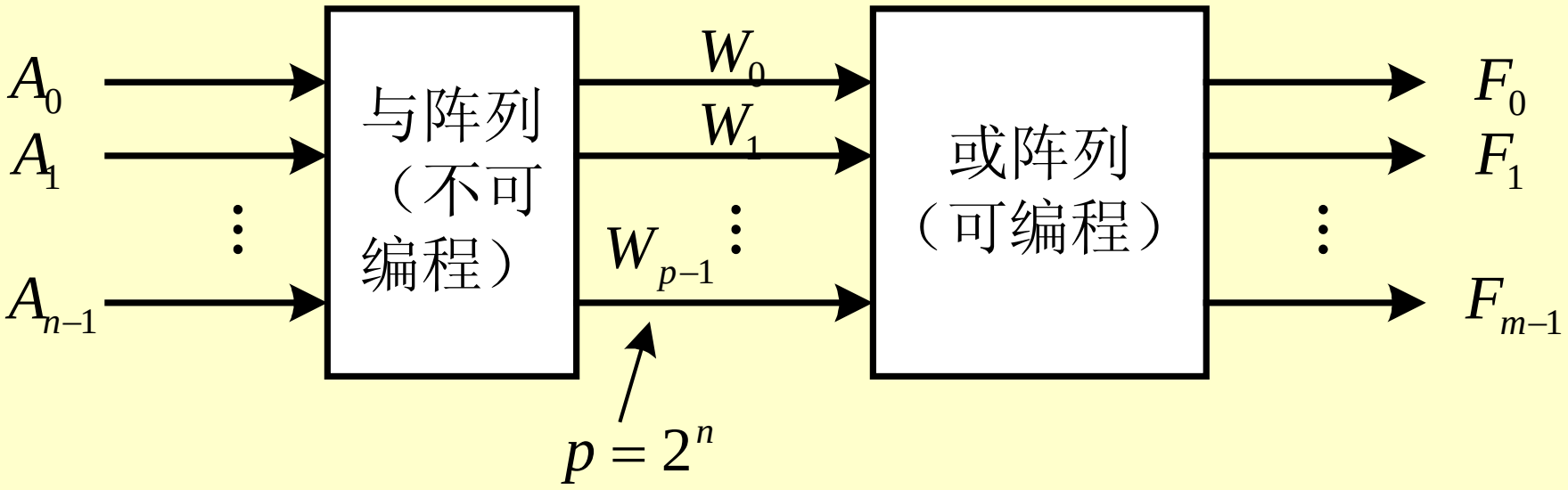


图3-10 PROM的逻辑阵列结构

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.2 PROM

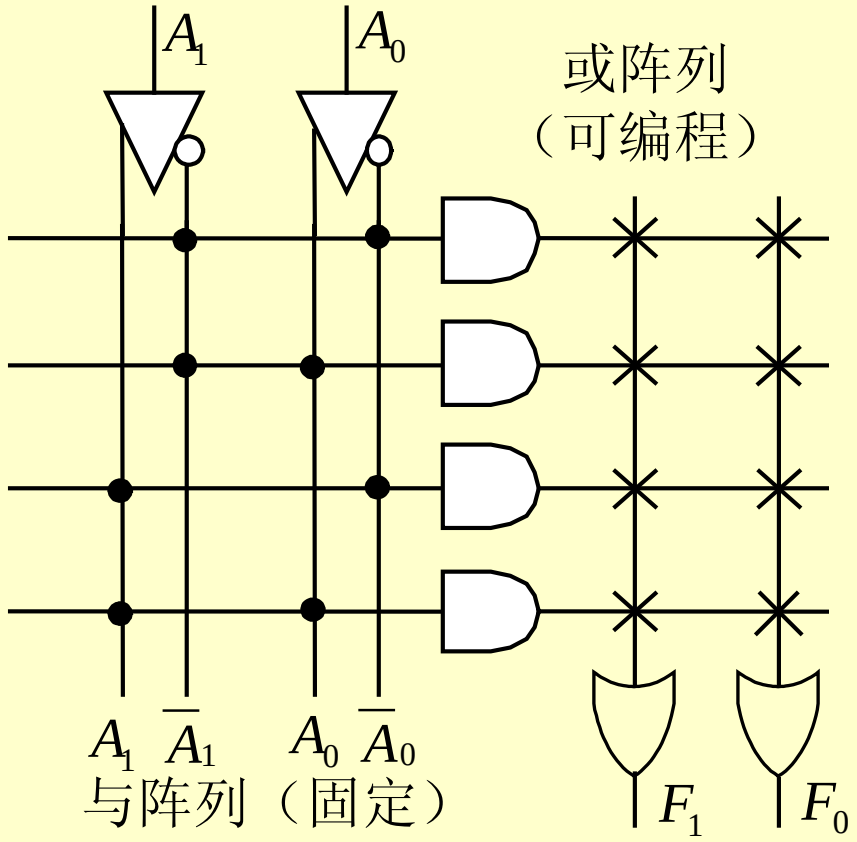


图3-11 PROM表达的PLD阵列图

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.2 PROM

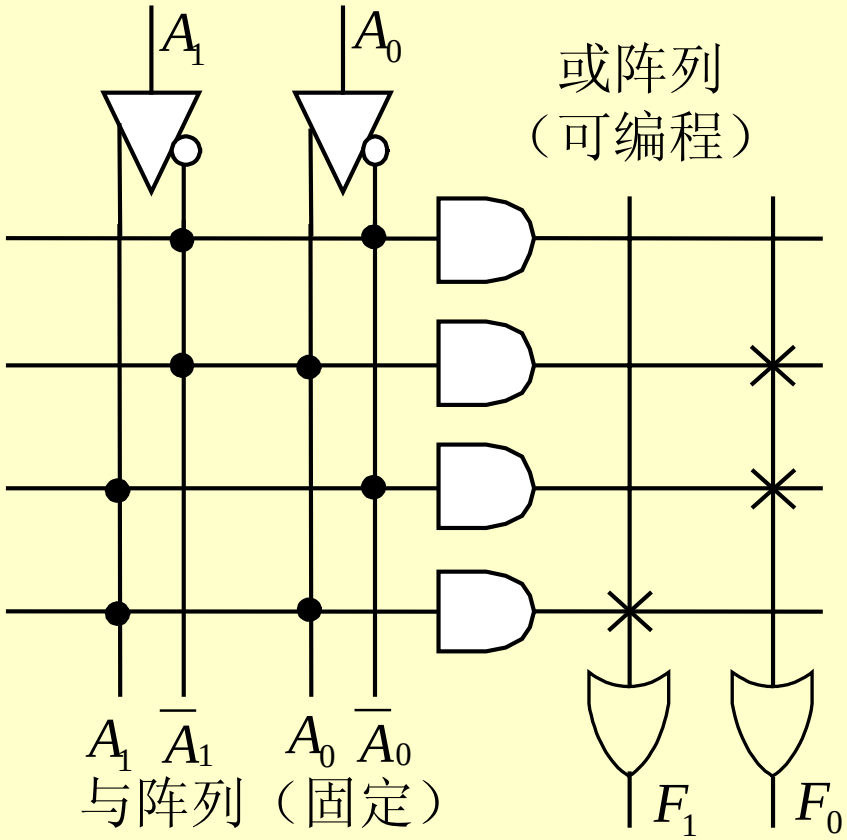


图3-12 用PROM完成半加器逻辑阵列

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.3 PLA

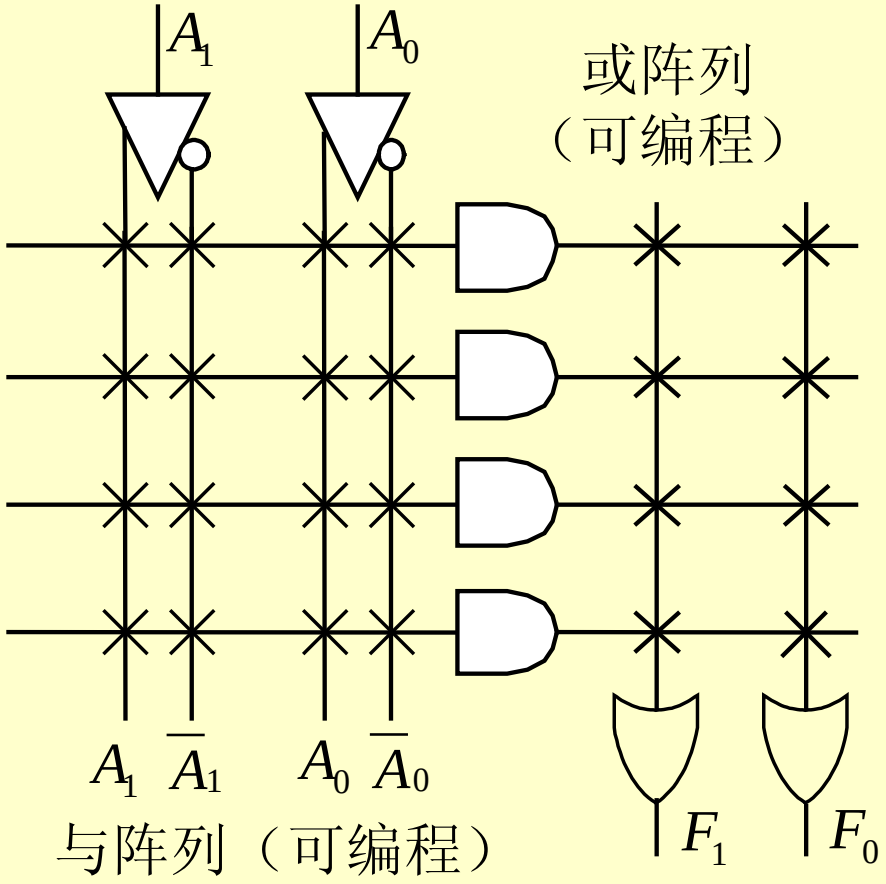
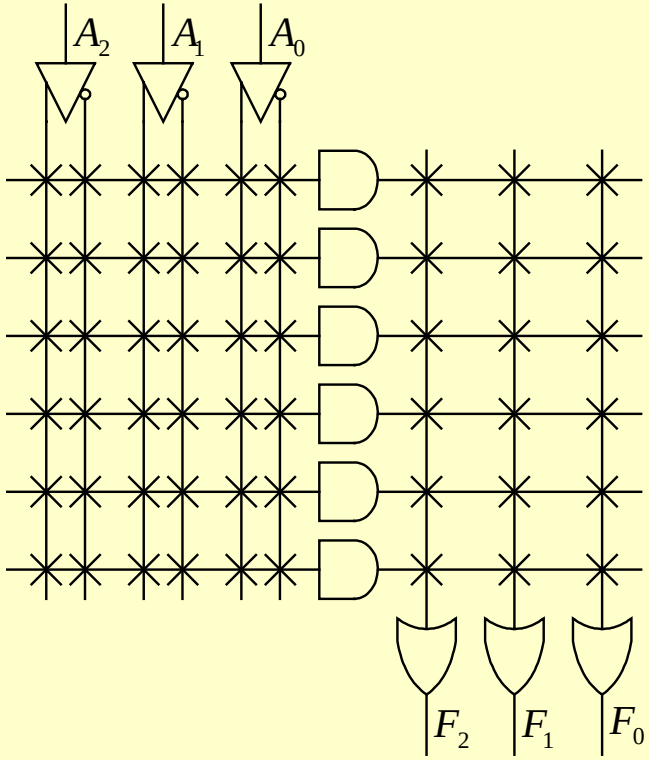


图3-13 PLA逻辑阵列示意图

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.3 PLA



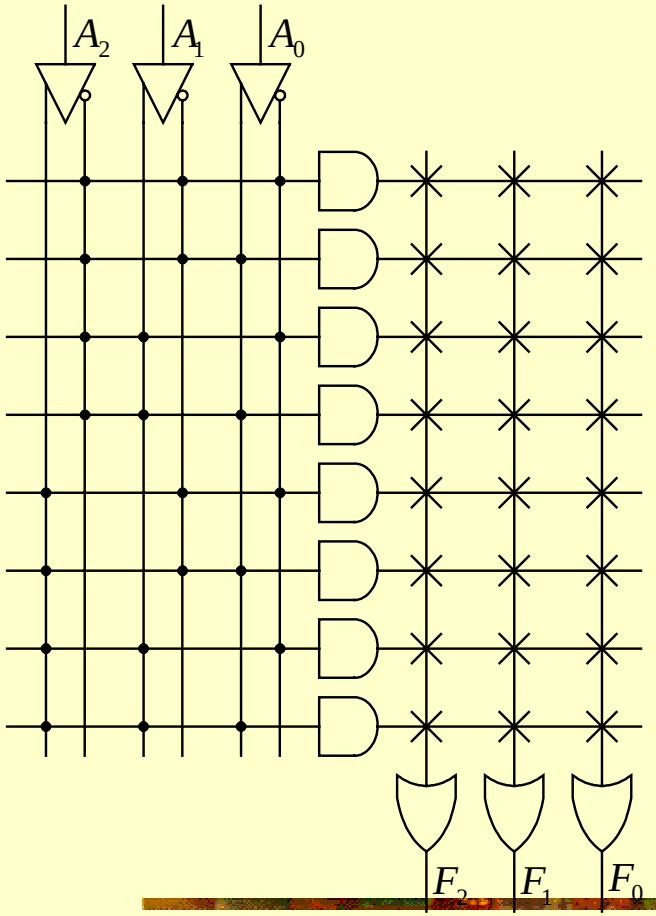


图3-14 PLA与 PROM的比较

KX 康芯科技

3.2 简单可编程逻辑器件原理

3.2.4 PAL

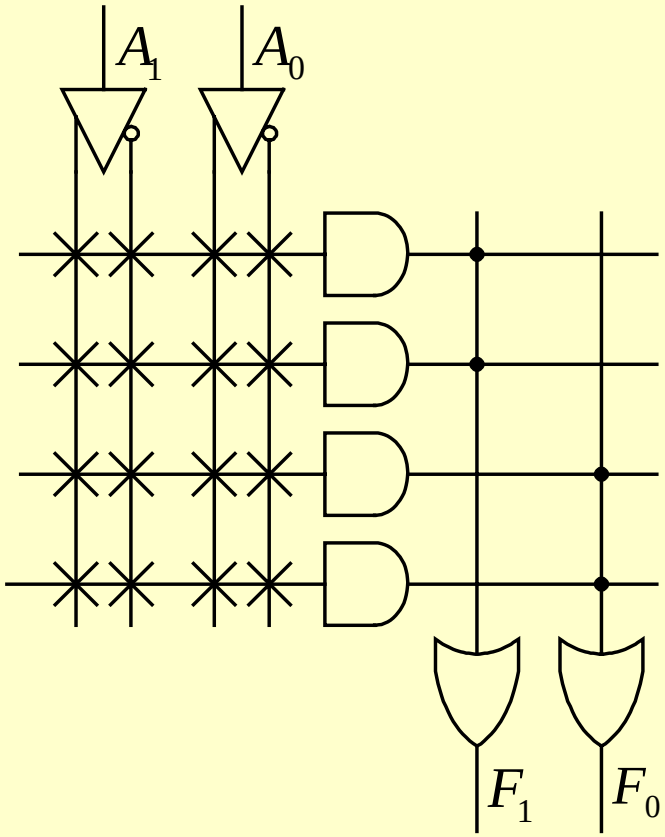


图3-15 PAL结构

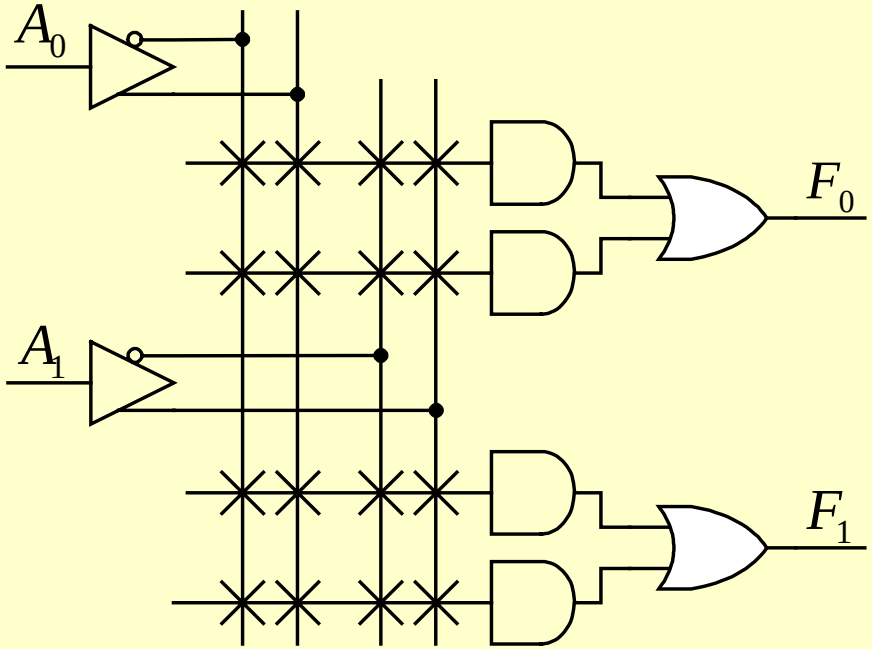


图3-16 PAL的常用表示

KX 康芯科技

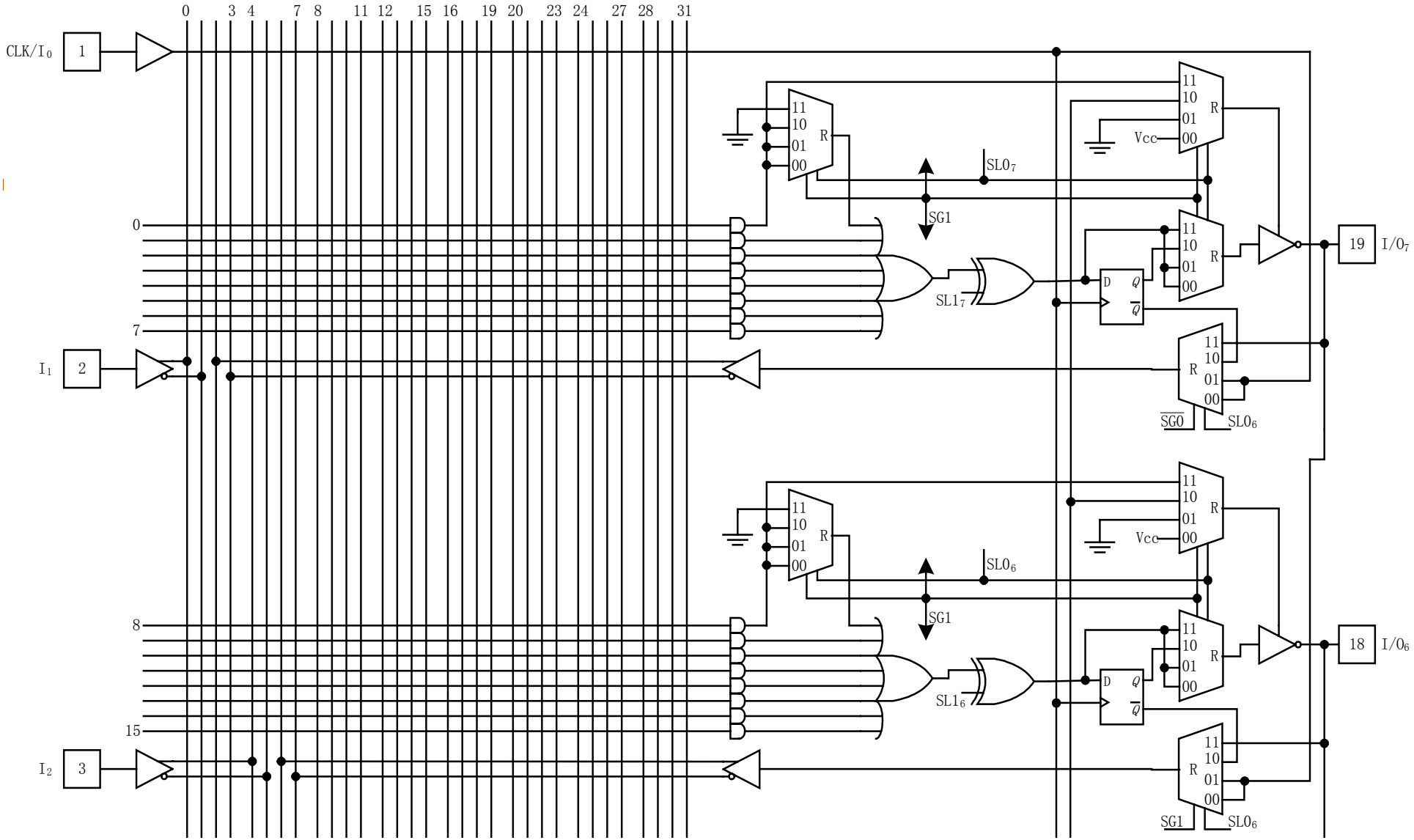


图3-17 一种PAL16V8的部分结构图

康芯科技

3.2.5 GAL

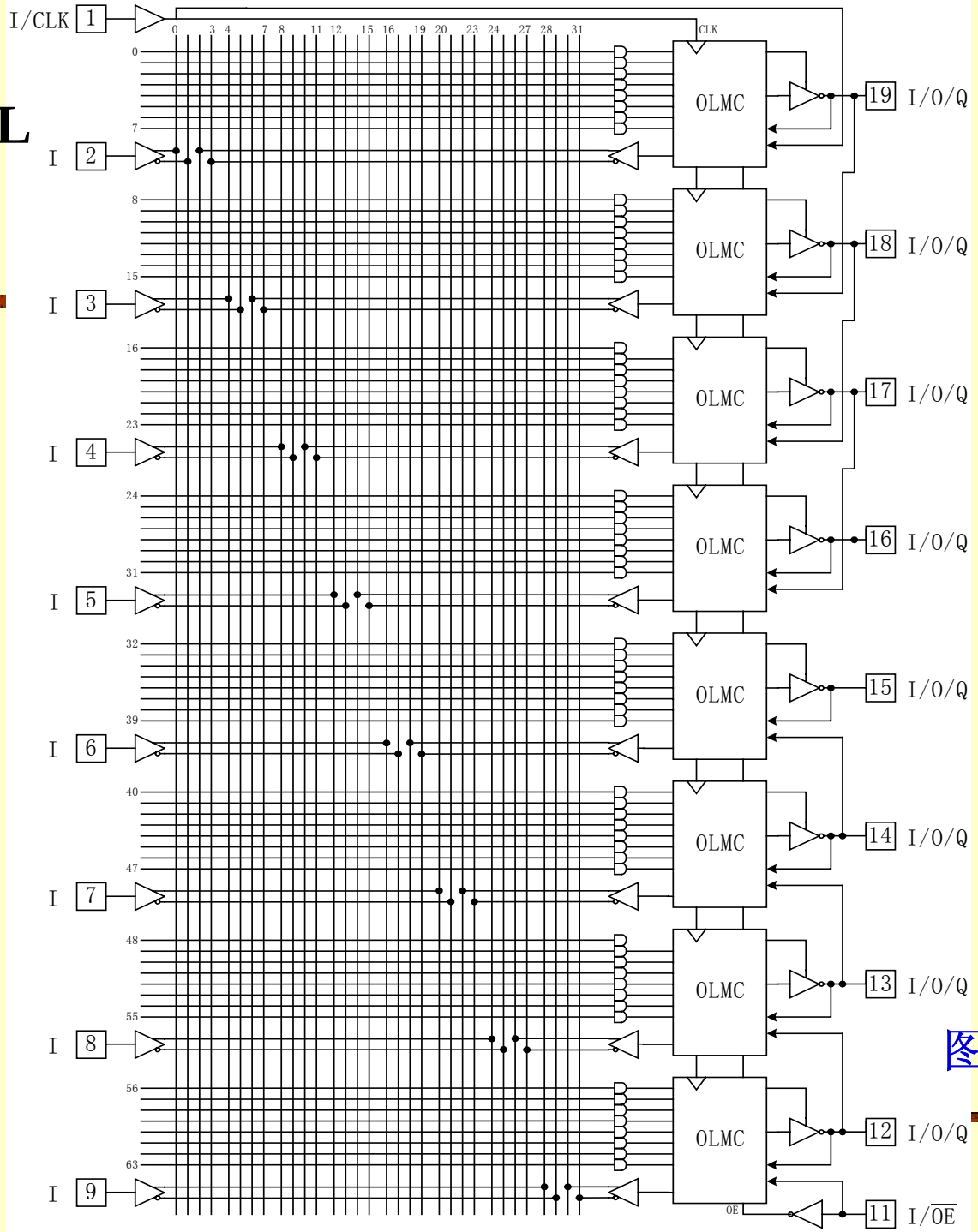


图3-15 PAL结构

康芯科技

3.2 简单可编程逻辑器件原理

3.2.5 GAL

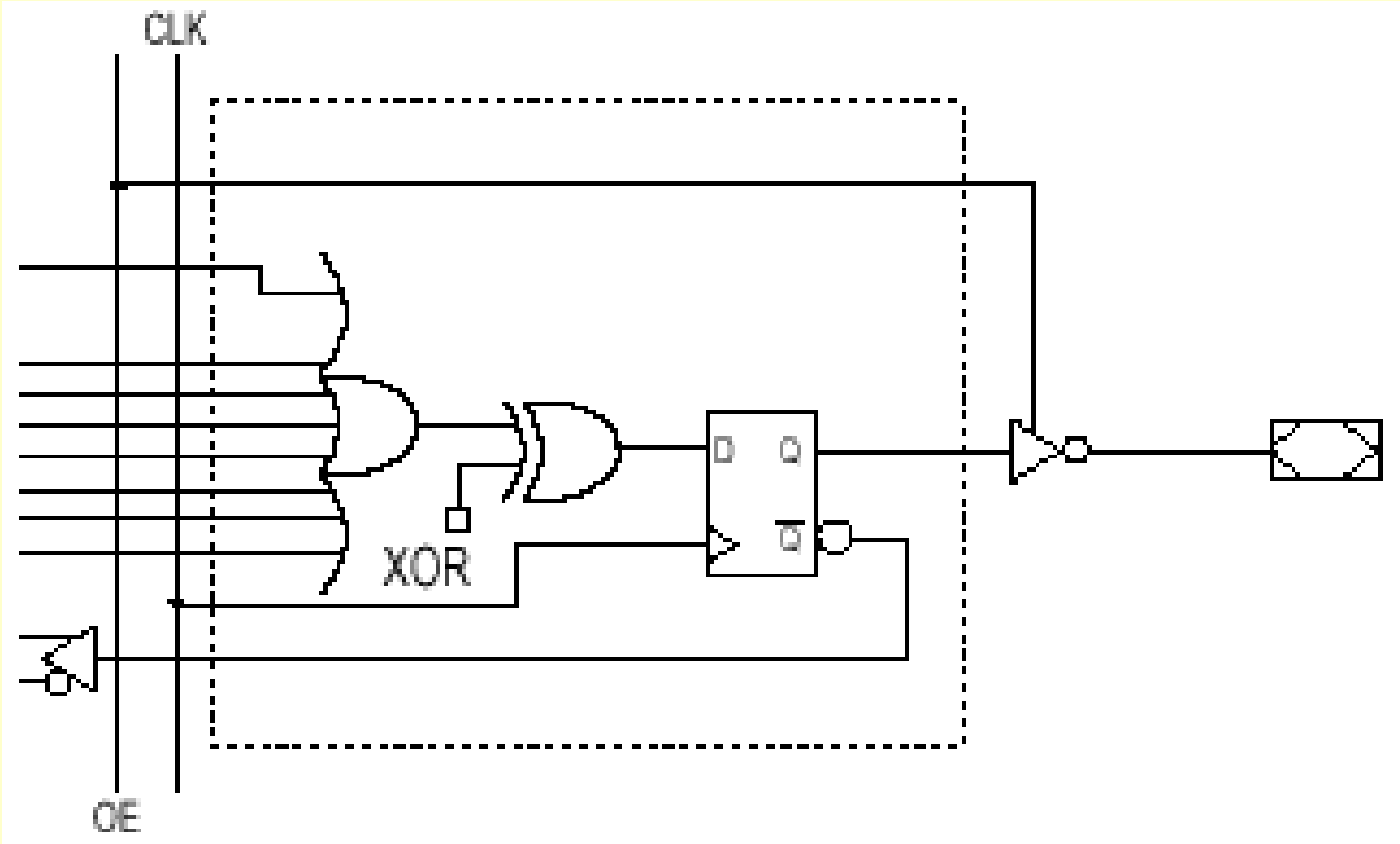


图3-15 PAL结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

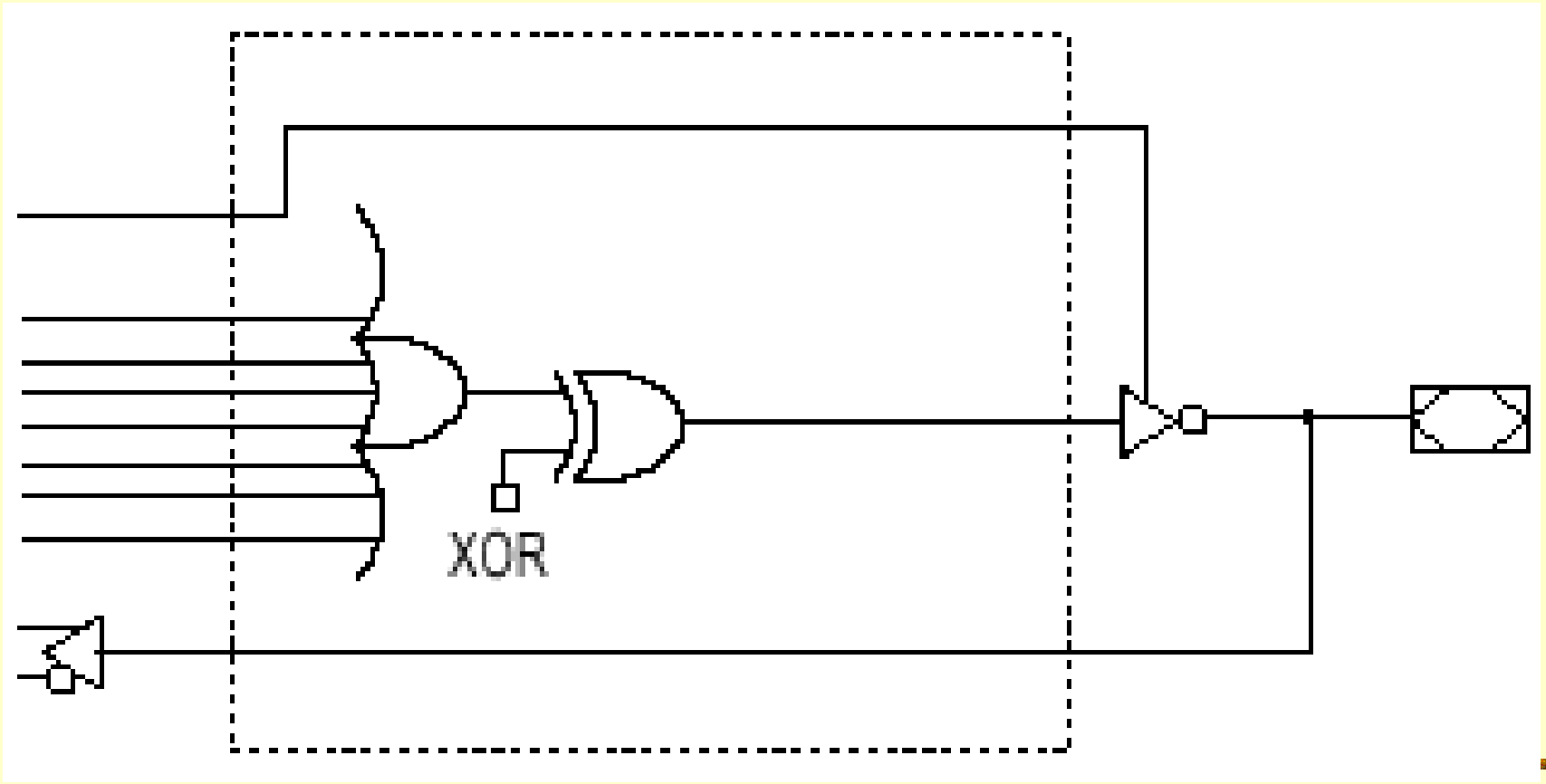


图3-20 寄存器模式组合双向输出结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

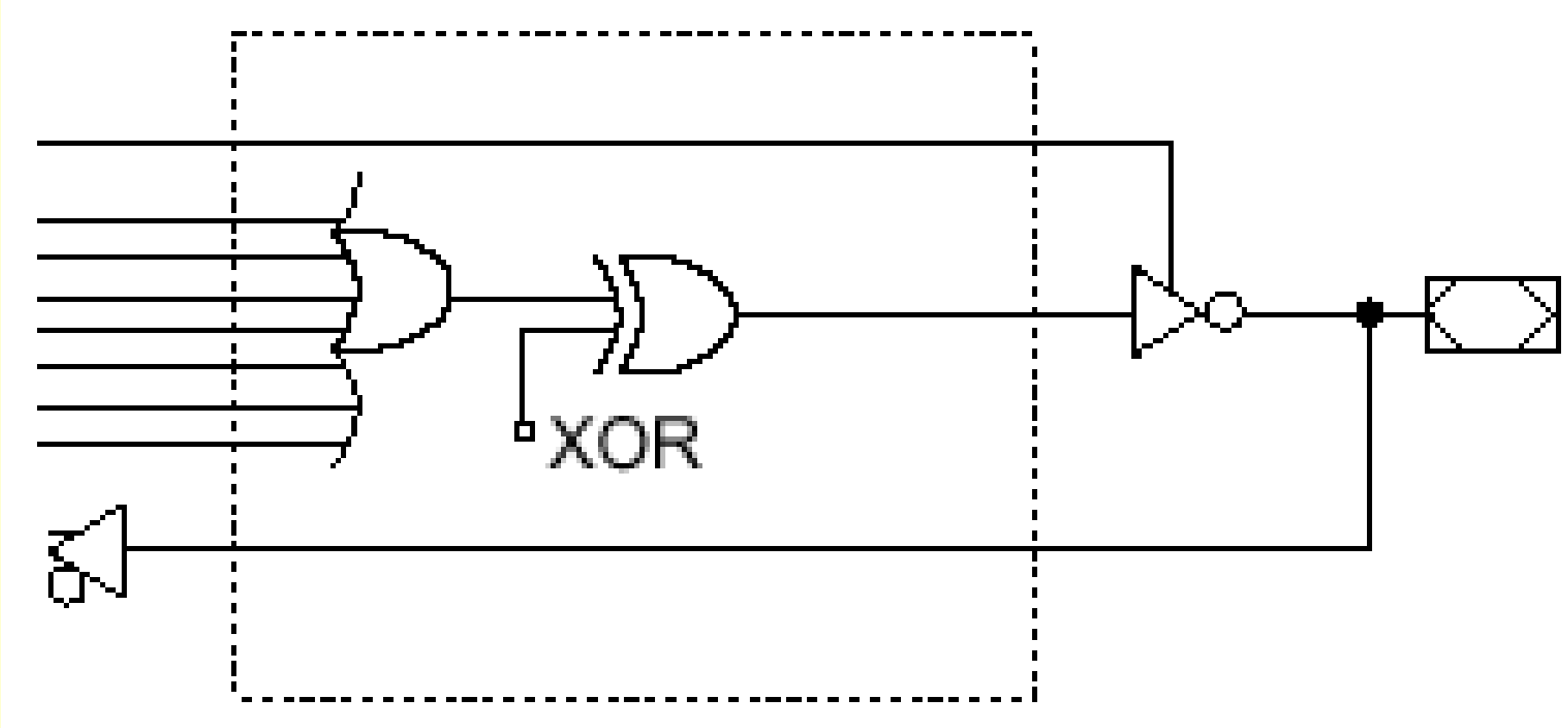


图3-21 组合输出双向结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

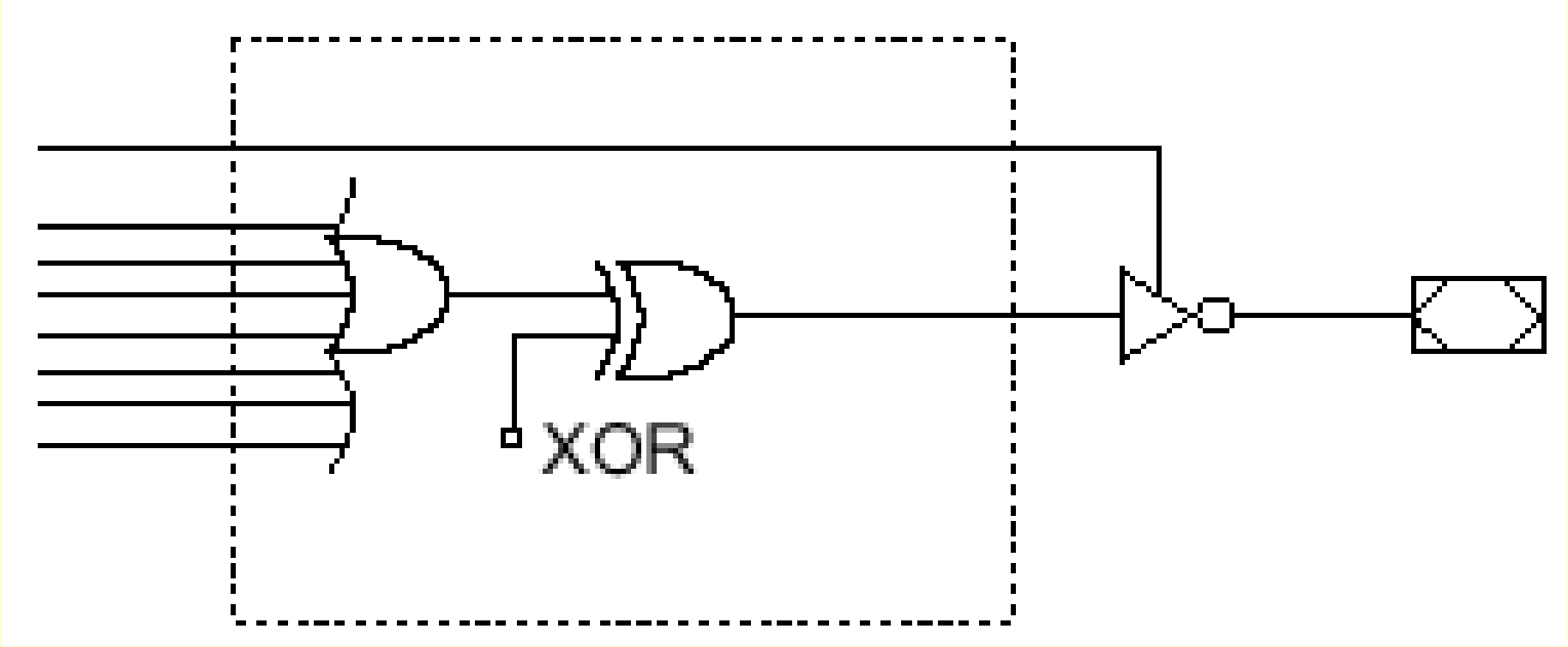


图3-22 复合型组合输出结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

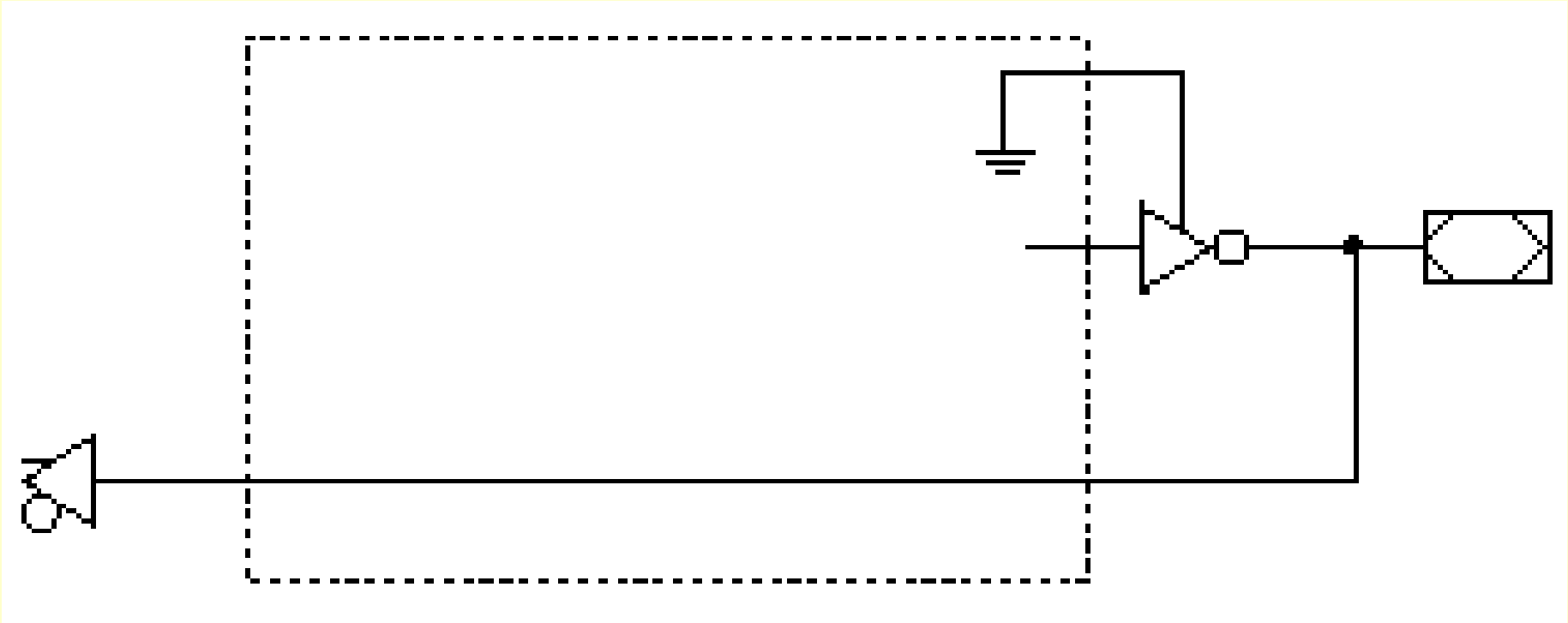


图3-23 反馈输入结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

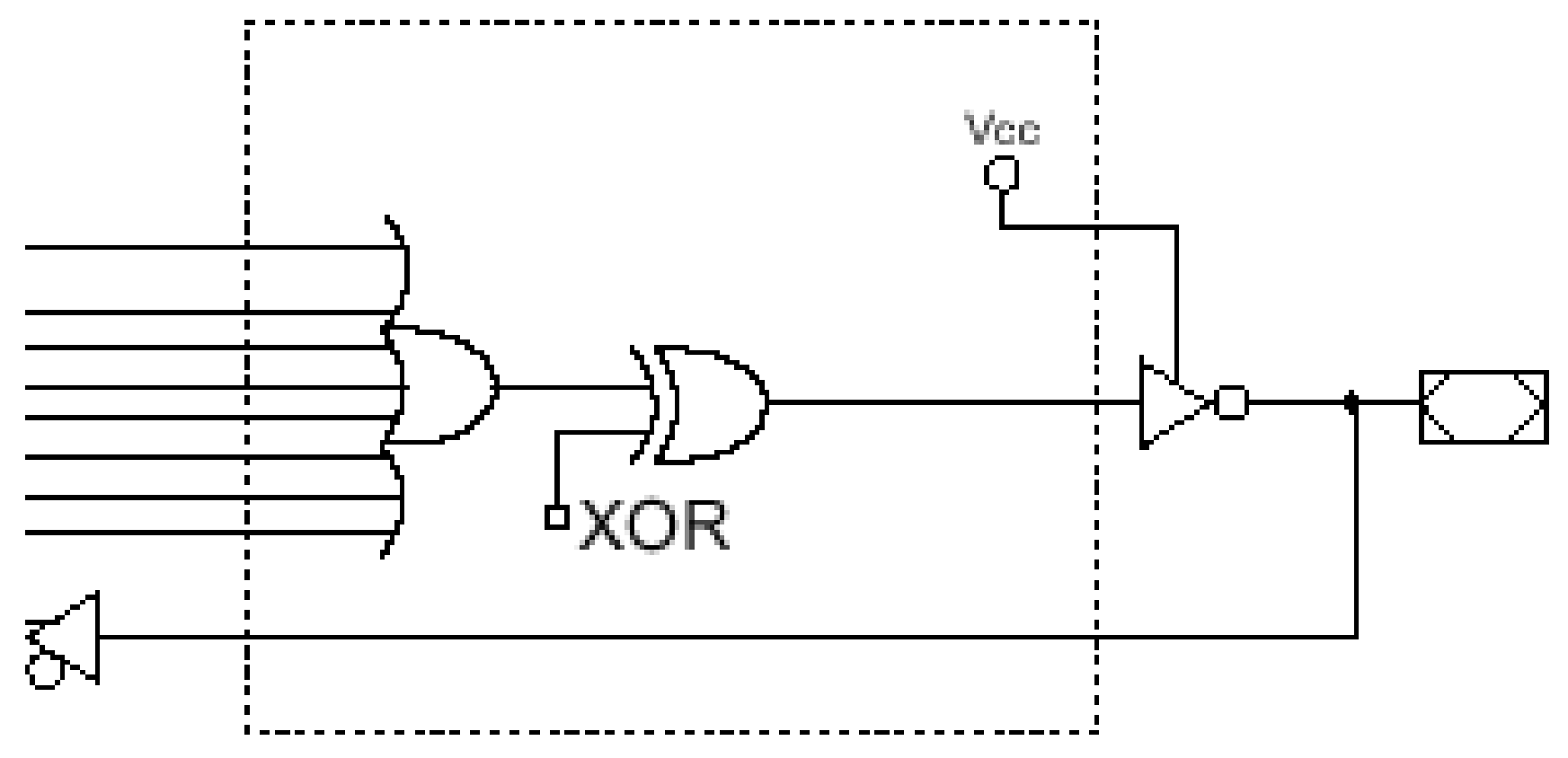


图3-24 输出反馈结构

3.2 简单可编程逻辑器件原理

3.2.5 GAL

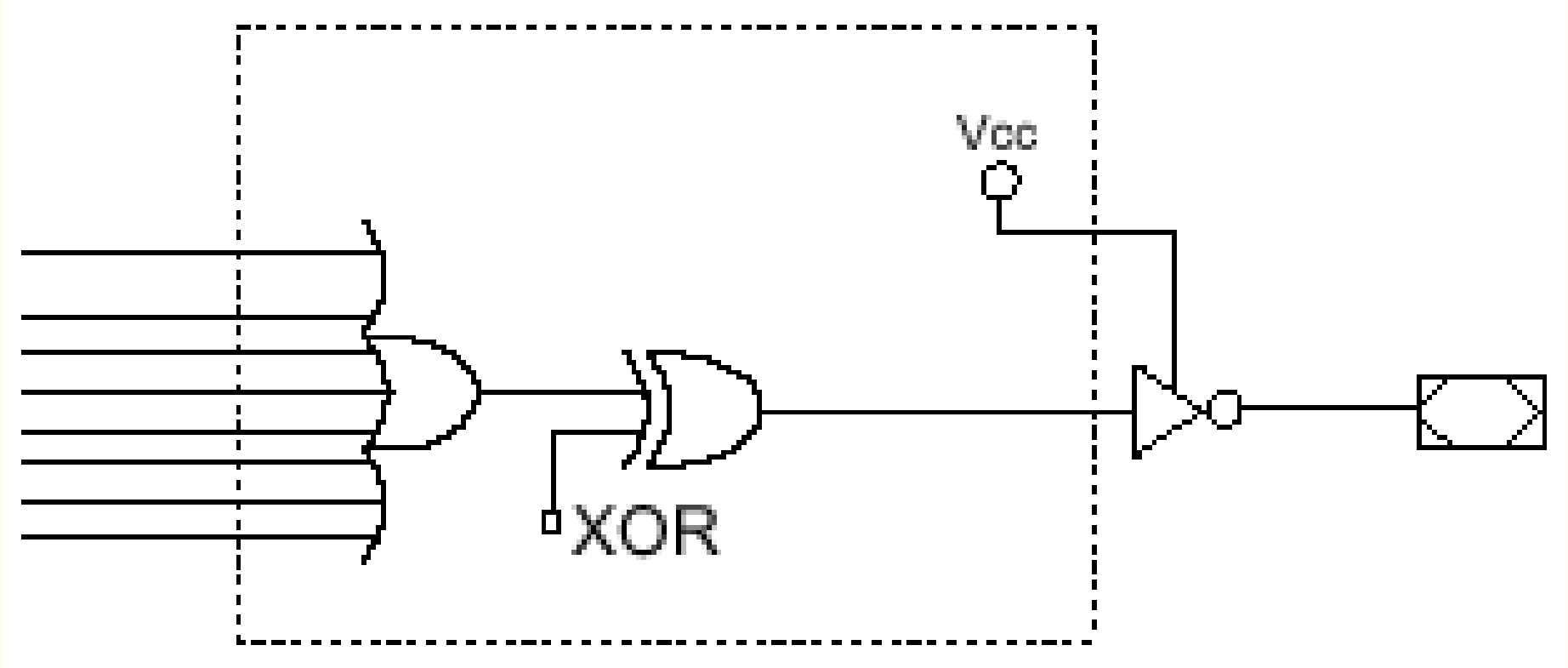


图3-25 简单模式输出结构

3.3 CPLD的结构与工作原理

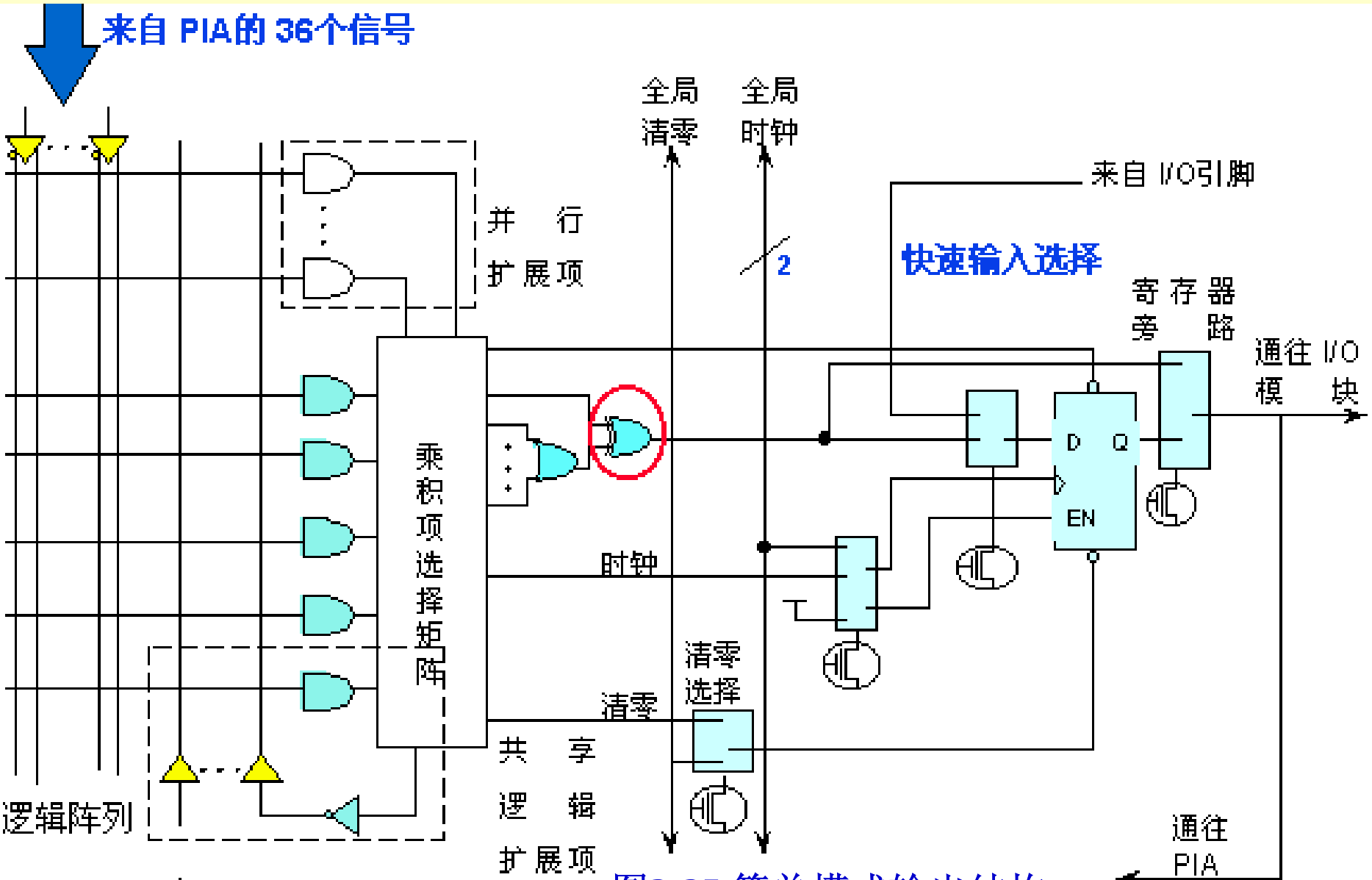


图3-25 简单模式输出结构

3.3 CPLD的结构与工作原理

1．逻辑阵列块(LAB)

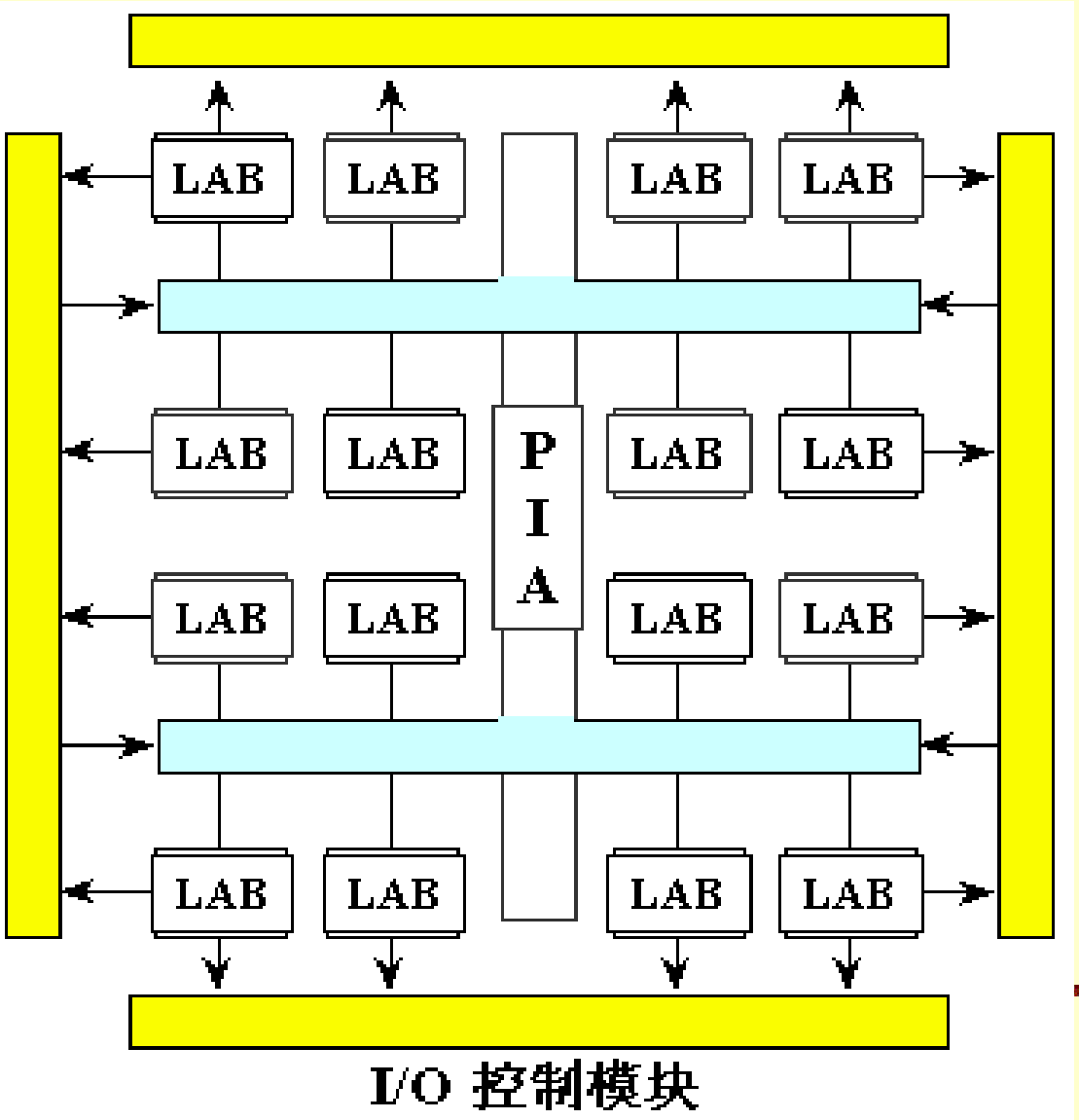
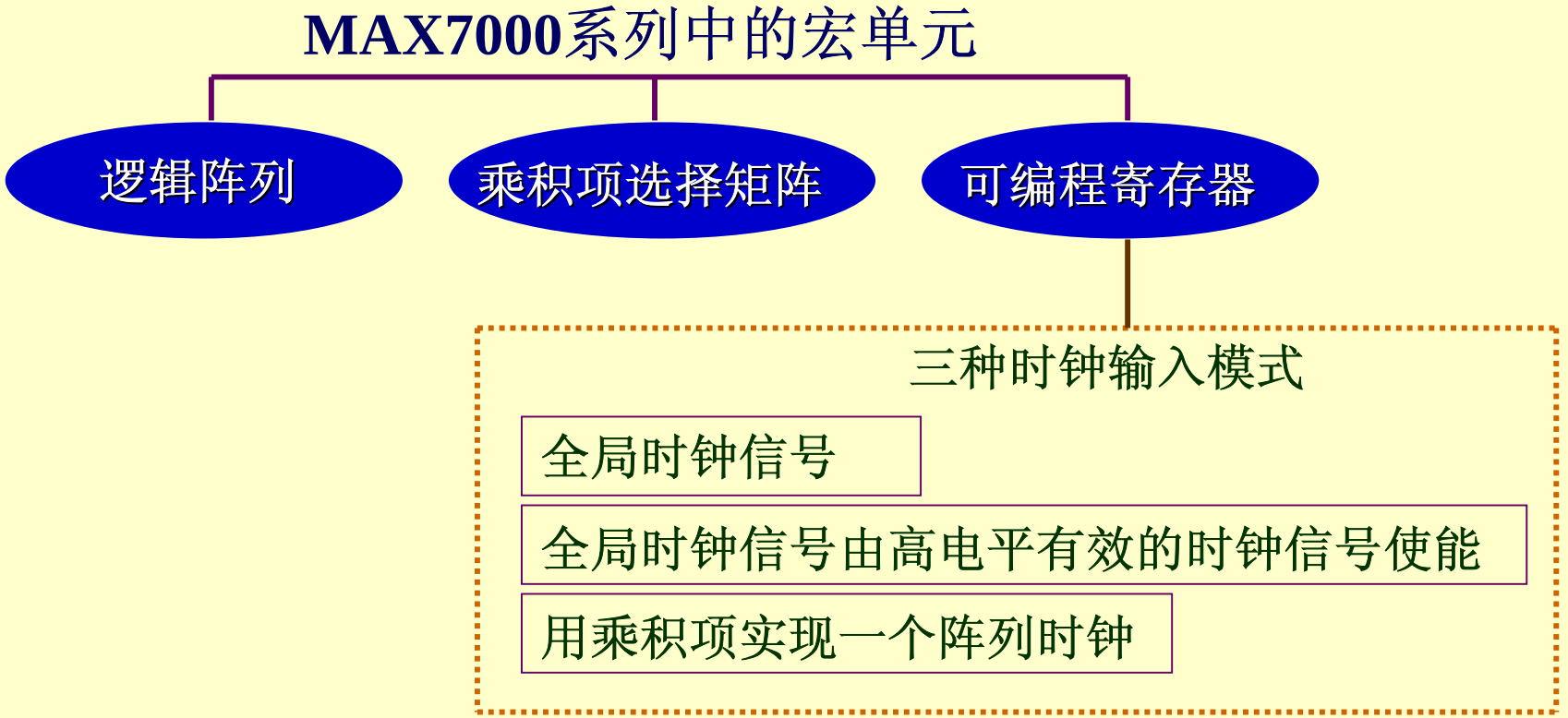


图3-27 MAX7128S的结构

3.3 CPLD的结构与工作原理

2．宏单元



3.3 CPLD的结构与工作原理

3．扩展乘积项

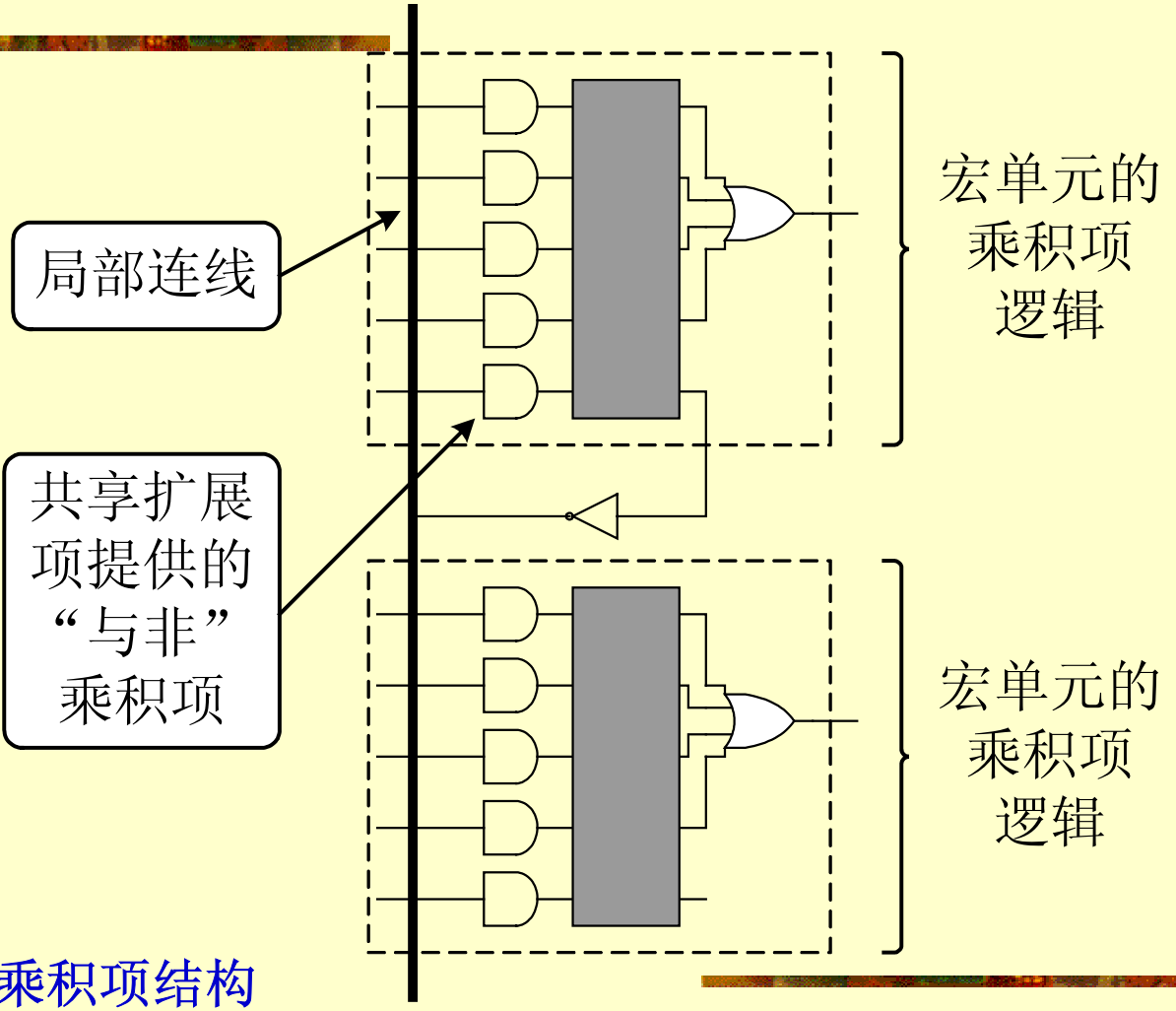


图3-28 共享扩展乘积项结构

3.3 CPLD的结构与工作原理

3．扩展乘积项

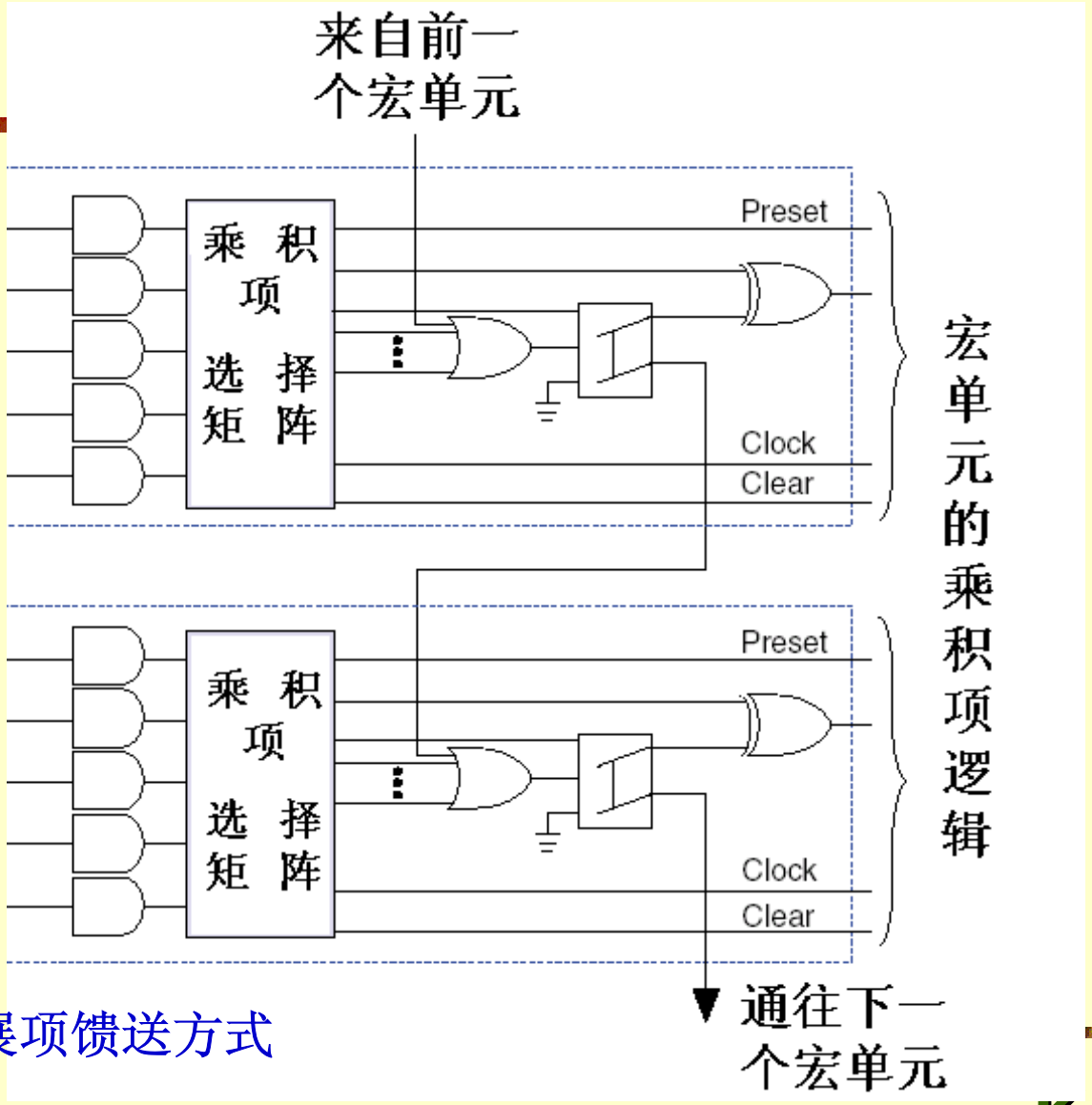


图3-29 并联扩展项馈送方式

3.3 CPLD的结构与工作原理

4. 可编程连线阵列(PIA)

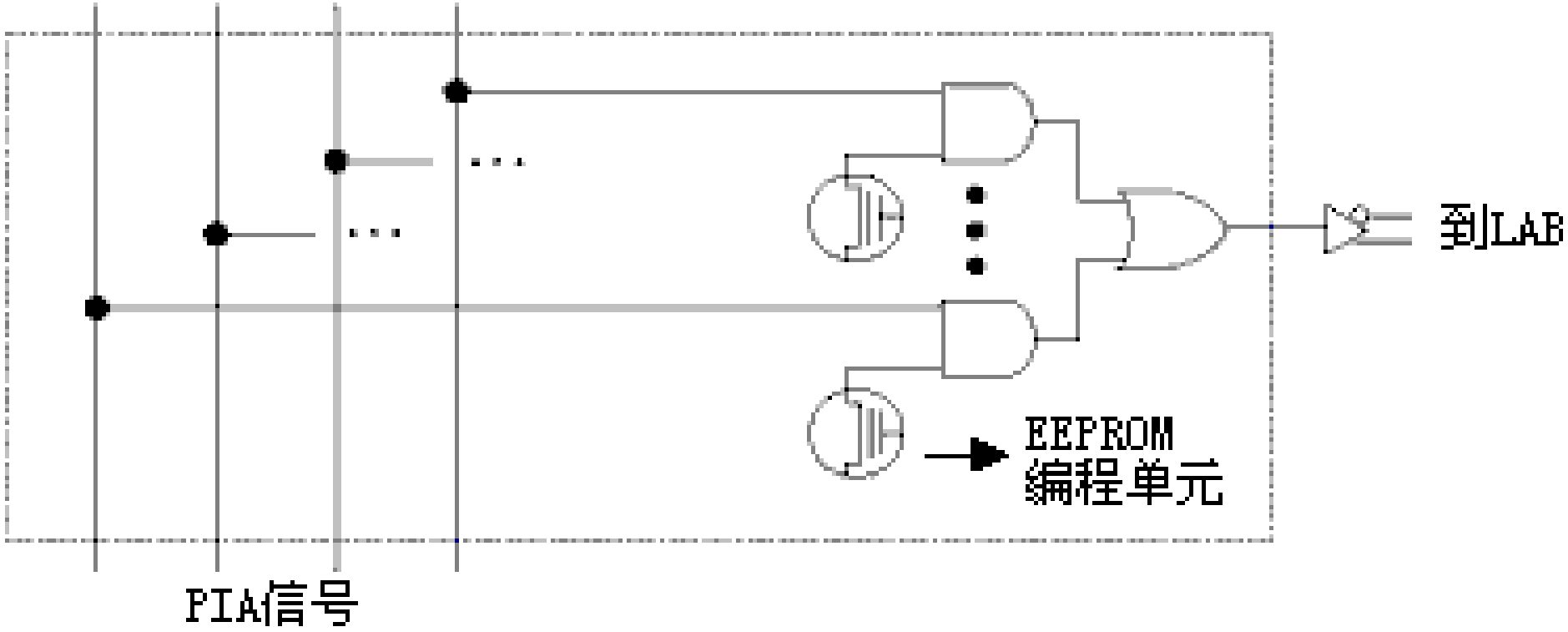


图3-30 PIA信号布线到LAB的方式

3.3 CPLD的结构与工作原理

5．I/O控制块

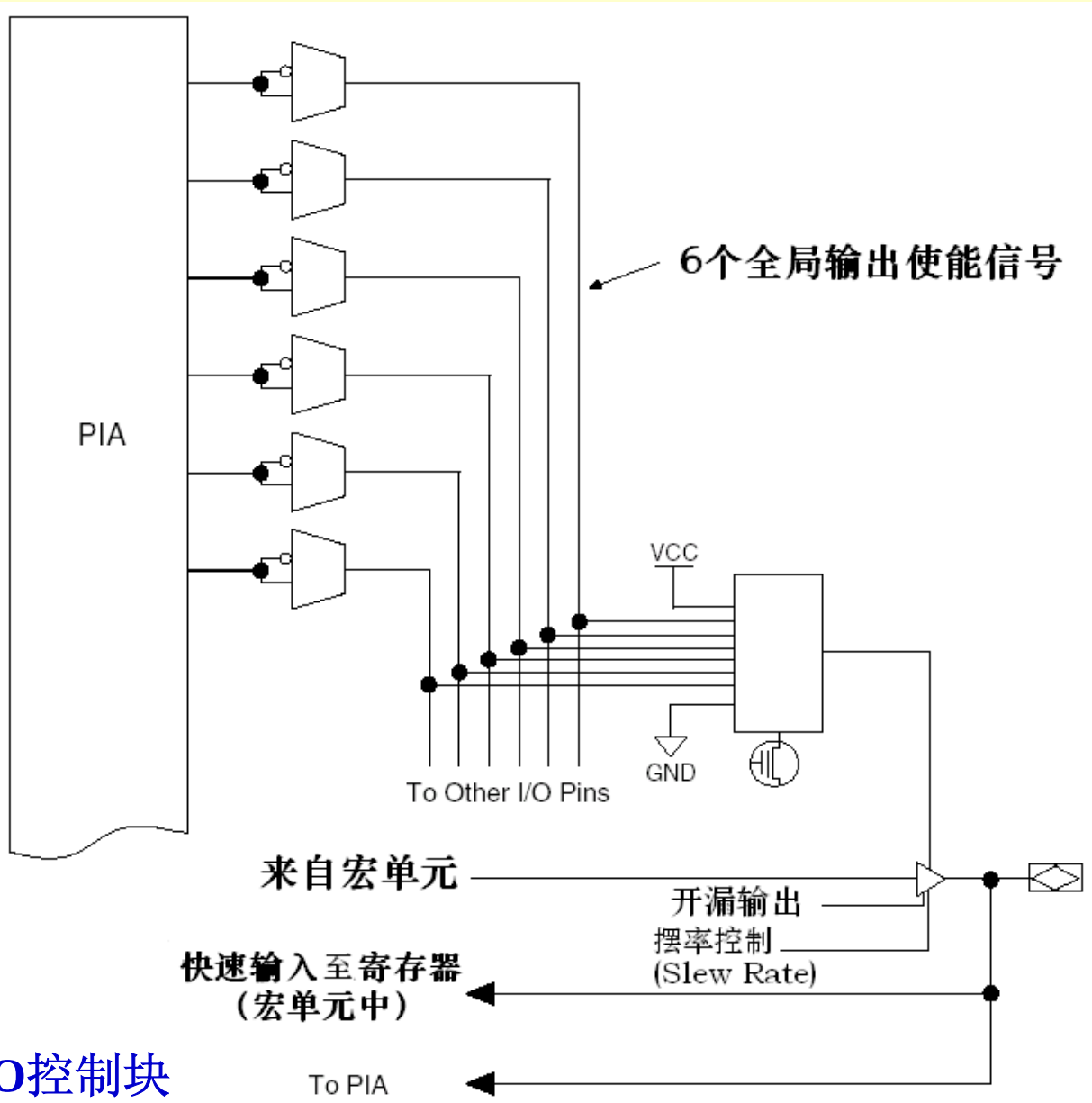


图3-31 EPM7128S器件的I/O控制块

3.4 FPGA的结构与工作原理

3.4.1 查找表逻辑结构

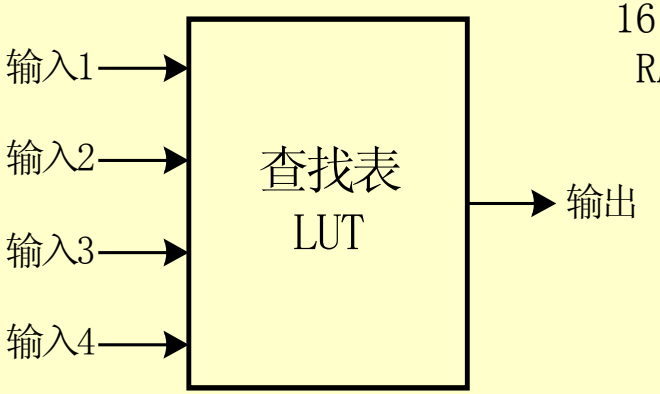


图3-32 FPGA查找表单元

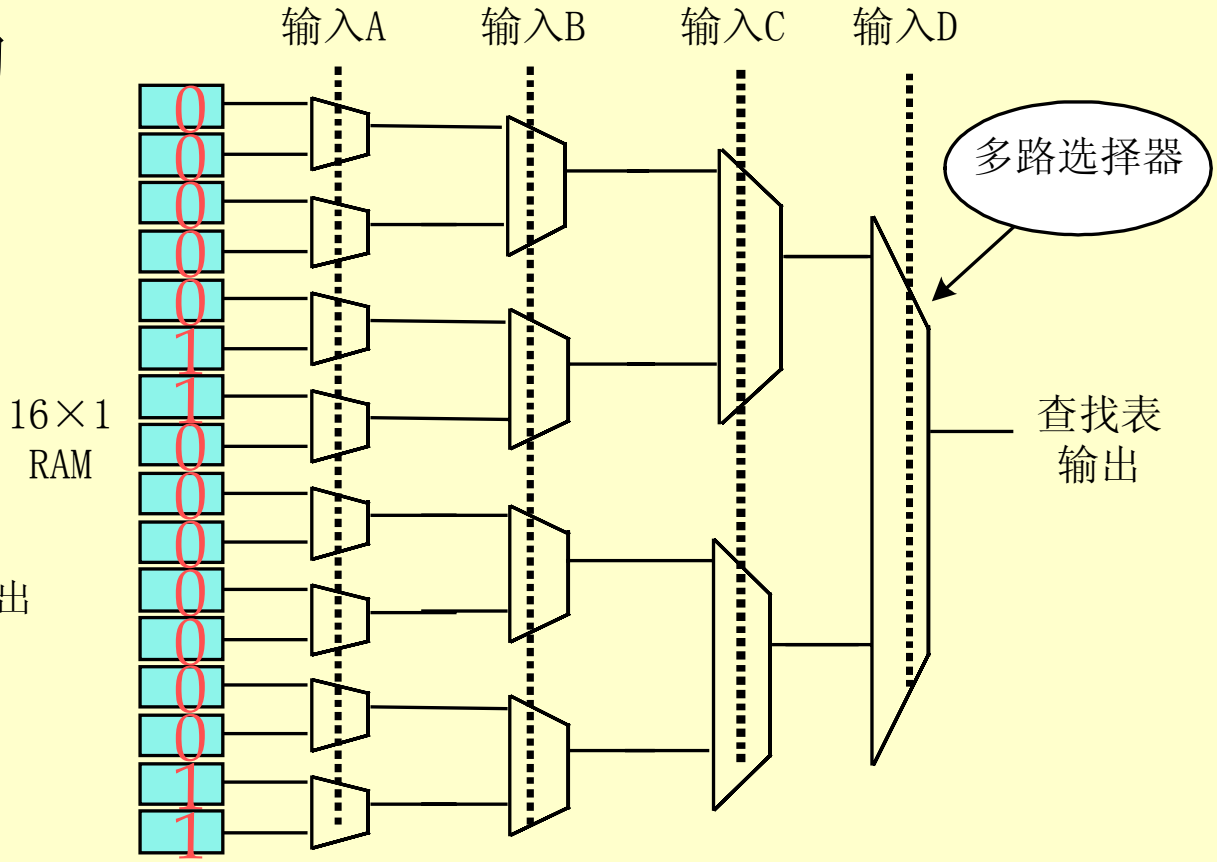


图3-33 FPGA查找表单元内部结构

KX 康芯科技

3.4.2 Cyclone/CycloneII系列器件的结构与原理

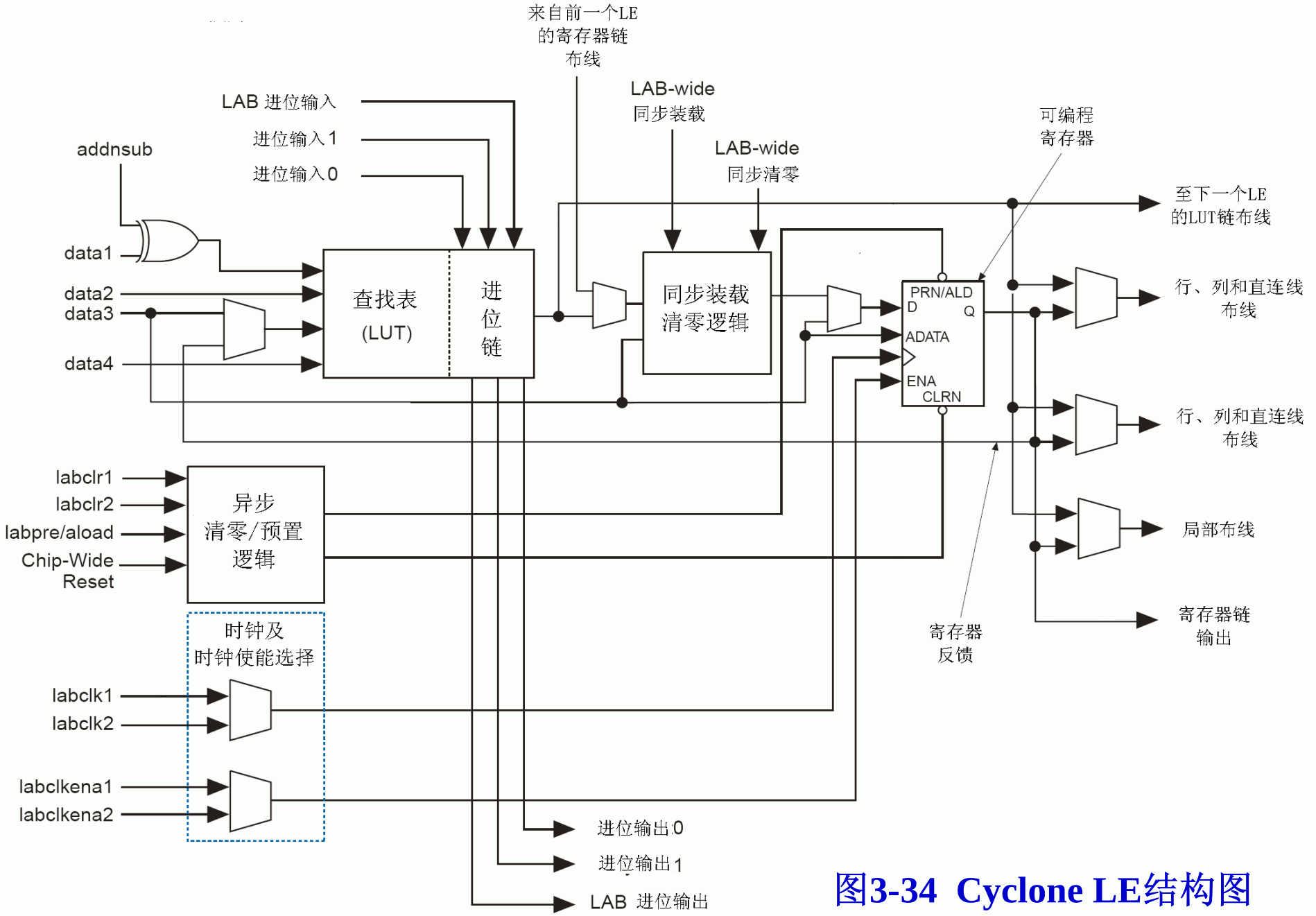


图3-34 Cyclone LE结构图

3.4 FPGA的结构与工作原理

3.4.2 Cyclone/CycloneII系列器件的结构与原理

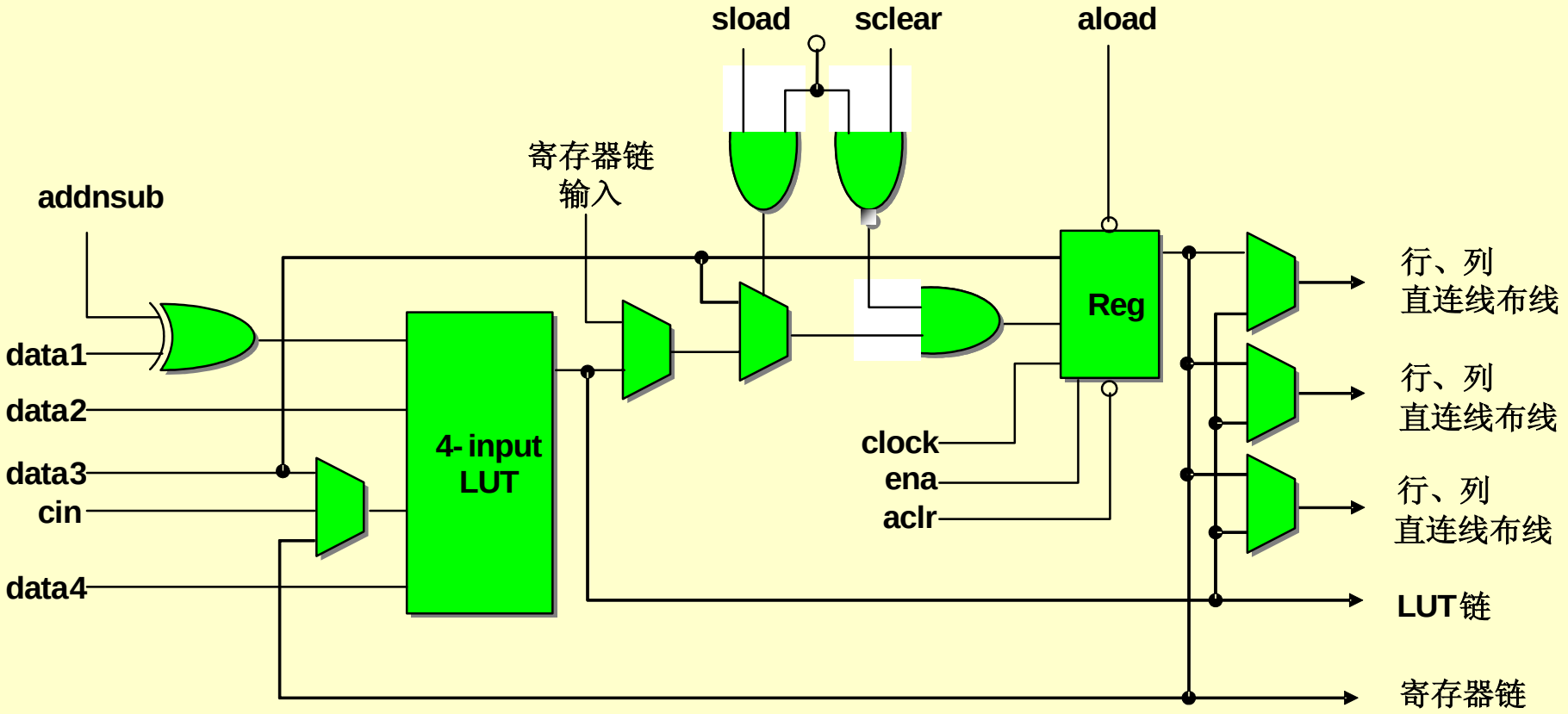


图3-35 Cyclone LE普通模式

3.4.2 Cyclone/CycloneII系列器件的结构与原理

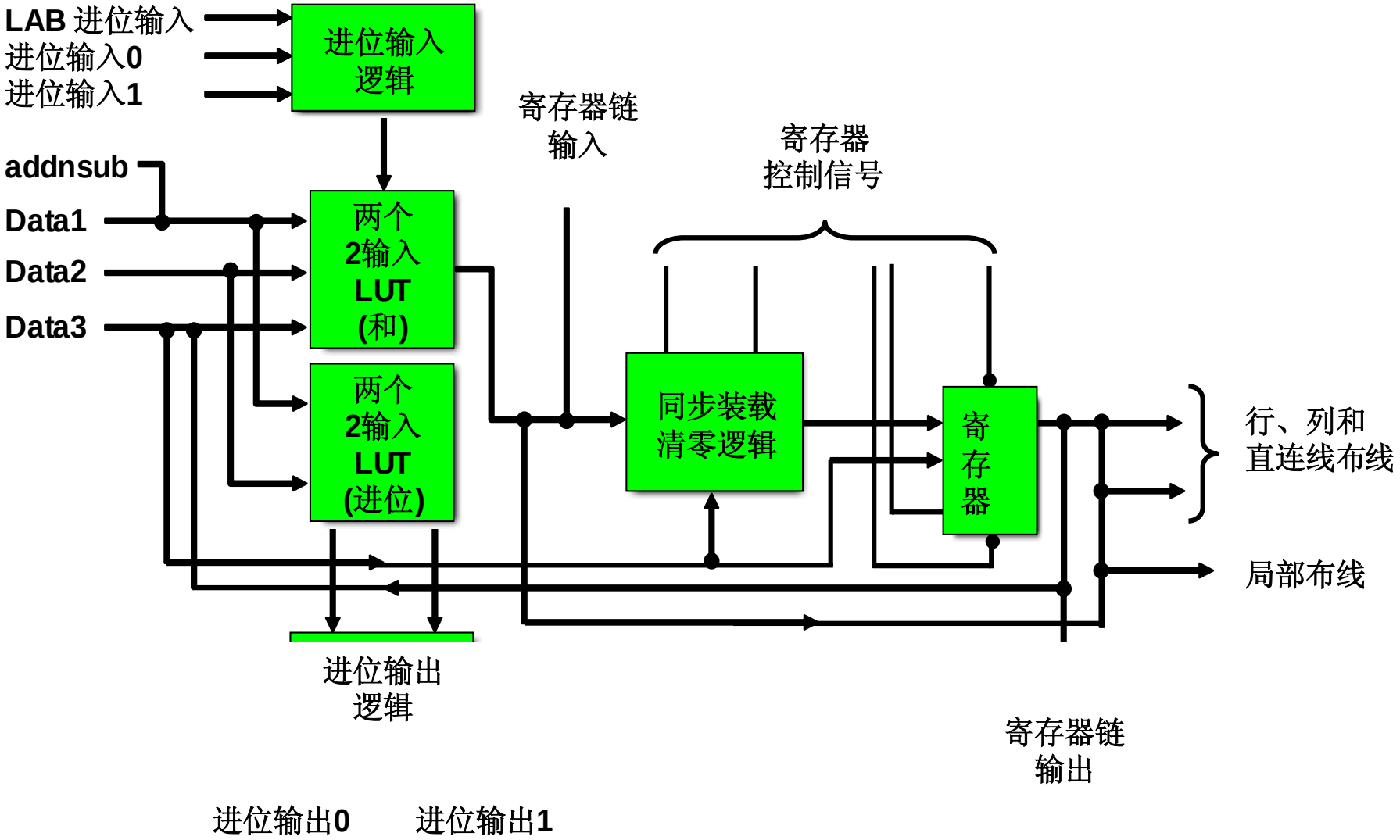


图3-36 Cyclone LE动态算术模式

3.4.2 Cyclone/CycloneII系列器件的结构与原理

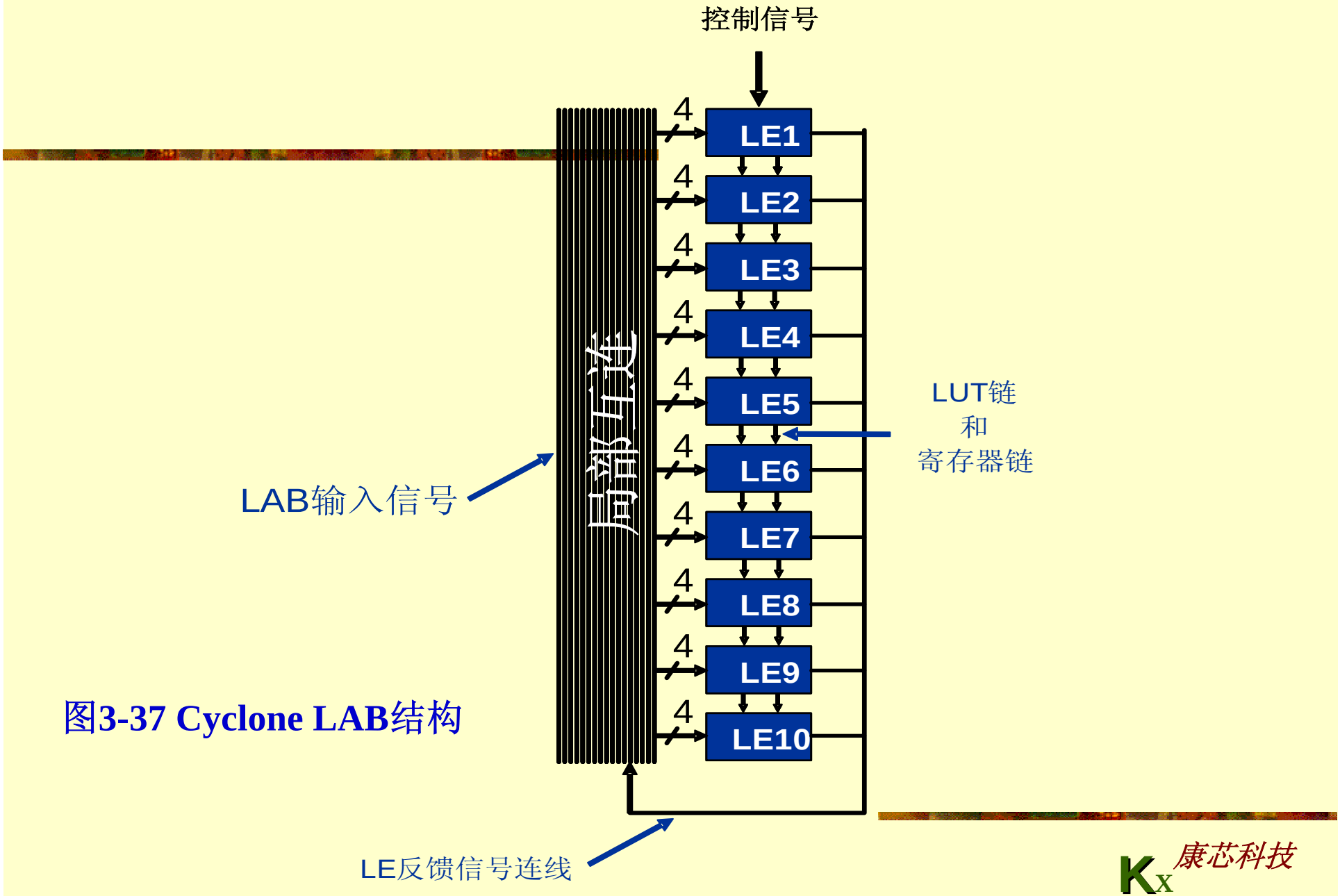


图3-37 Cyclone LAB结构

3.4.2 Cyclone/CycloneII系列器件的结构与原理

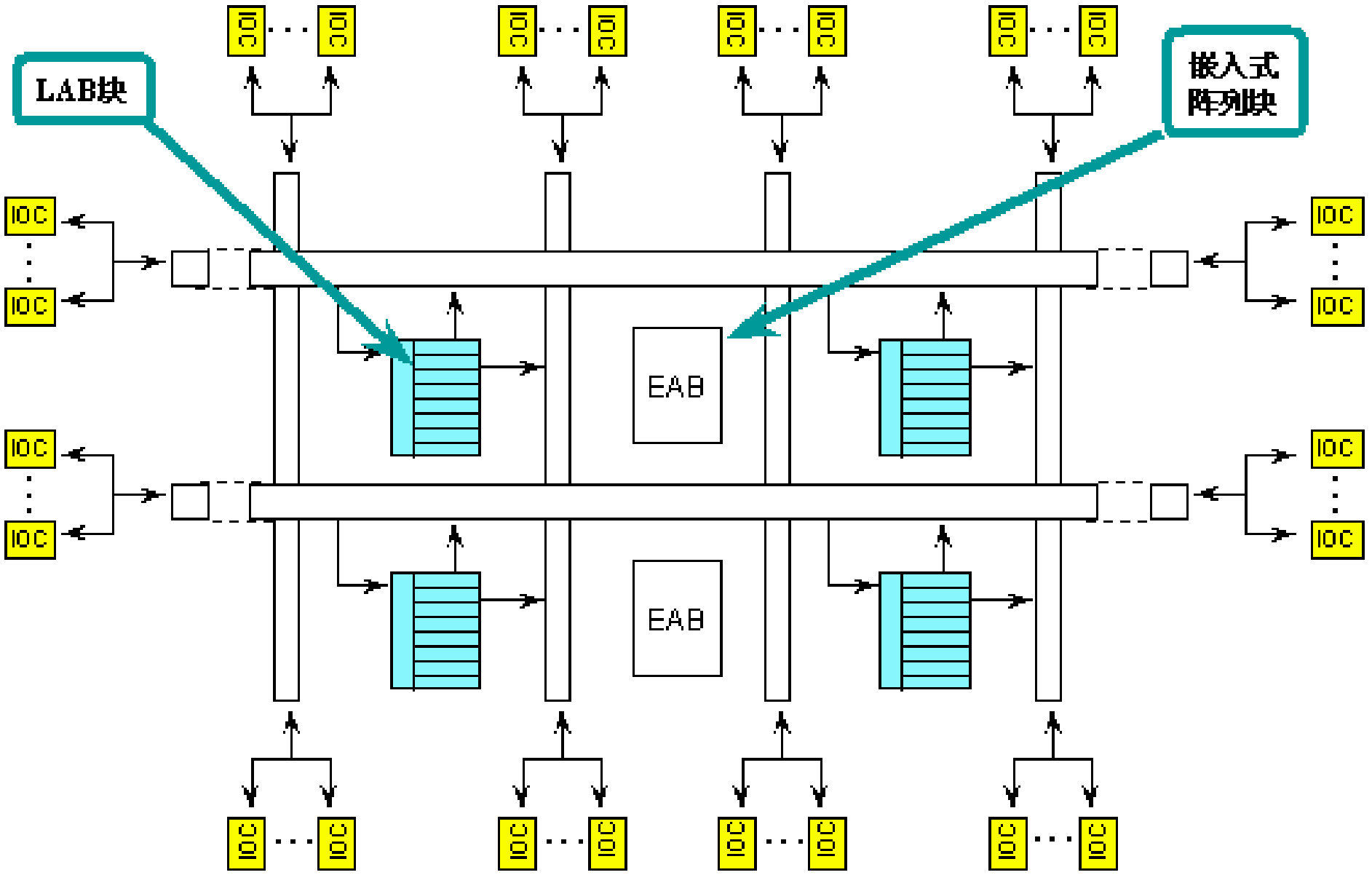


图3-38 LAB阵列

KX 康心科技

3.4.2 Cyclone/CycloneII系列器件的结构与原理

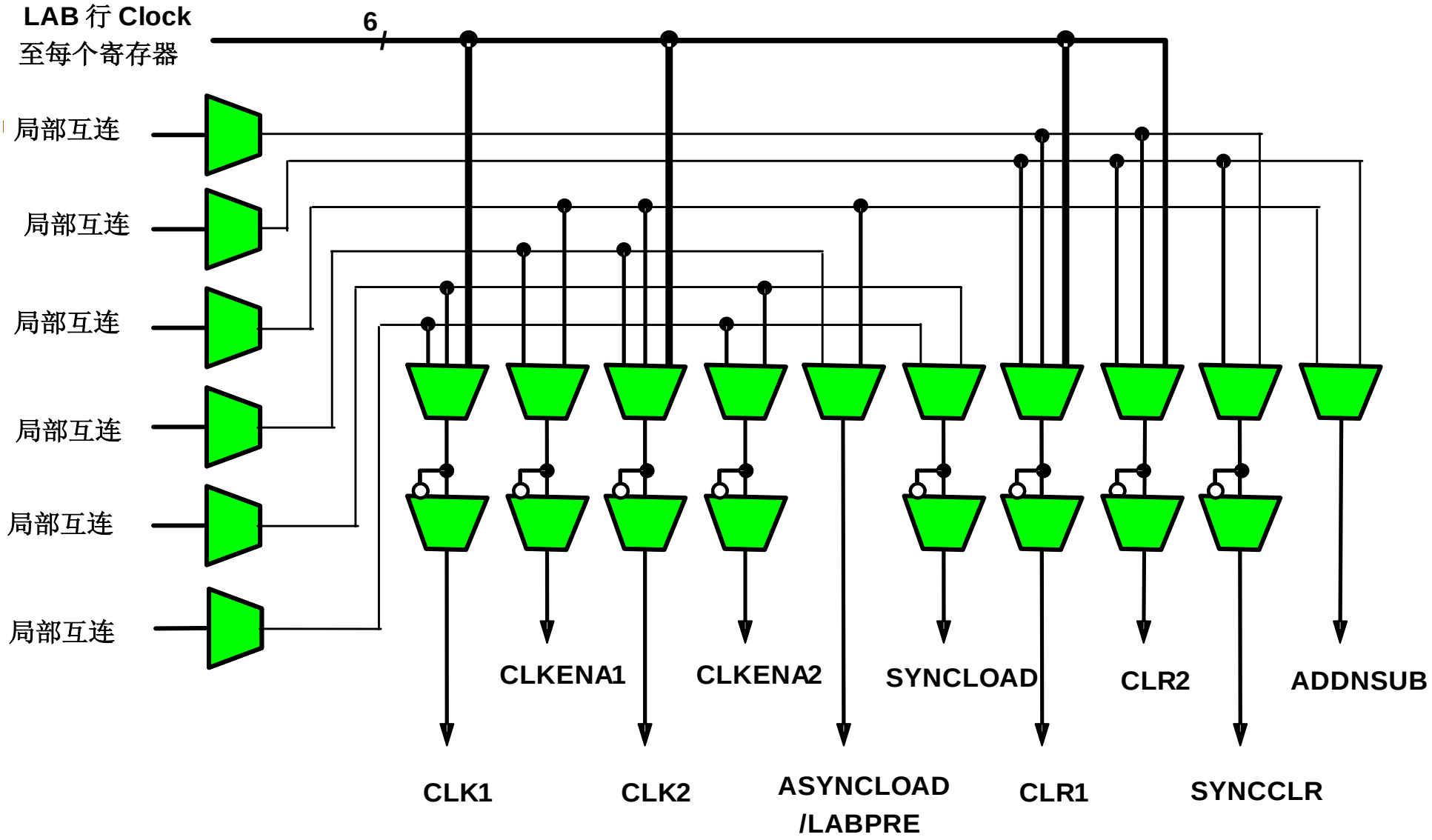


图3-39 LAB控制信号生成

3.4.2 Cyclone/CycloneII系列器件的结构与原理

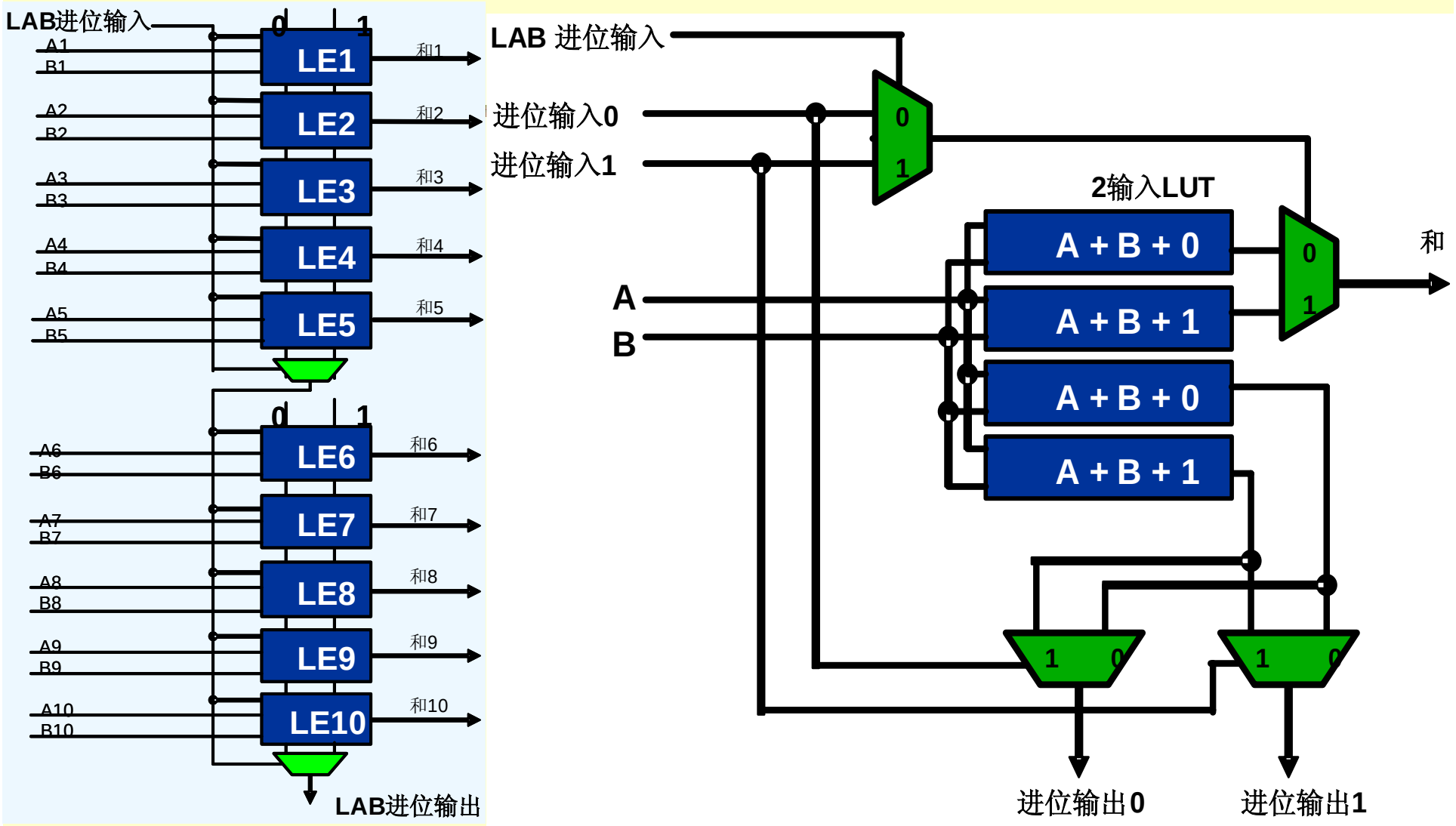


图2-40 快速进位选择链

3.4 FPGA的结构与工作原理

3.4.2 Cyclone/CycloneII系列器件的结构与原理

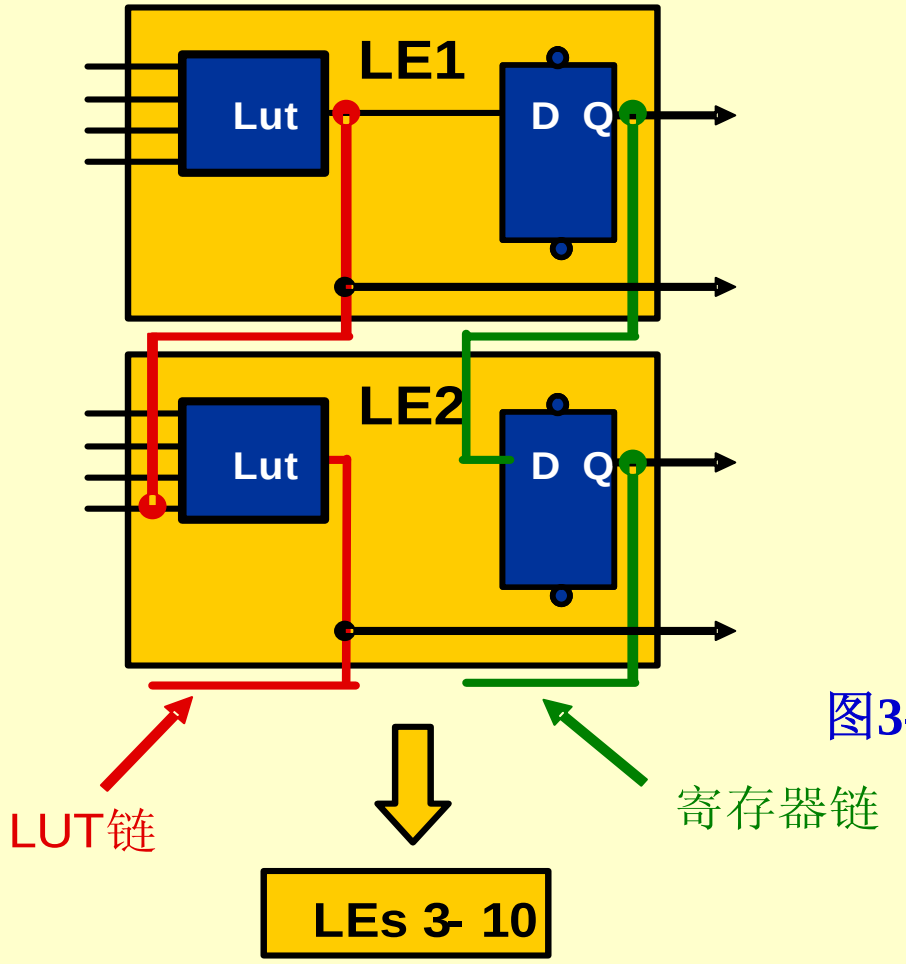


图3-41 LUT链和寄存器链的使用

3.4 FPGA的结构与工作原理

3.4.2 Cyclone/CycloneII系列器件的结构与原理

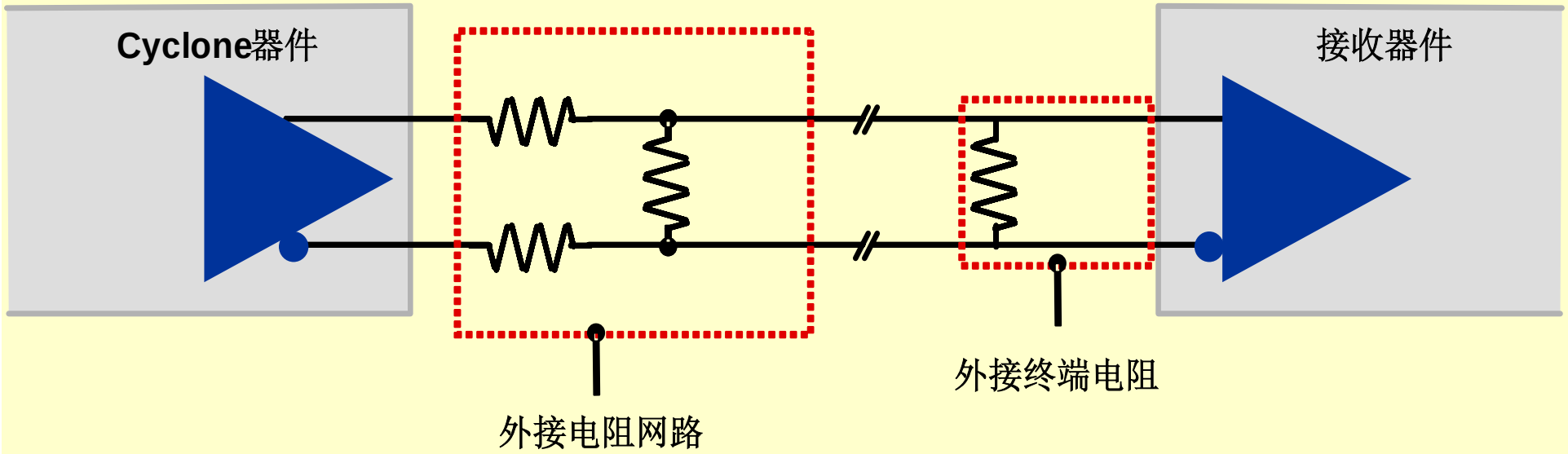


图3-42 LVDS连接

3.5 硬件测试技术

3.5.1 内部逻辑测试

3.5.2 JTAG边界扫描测试

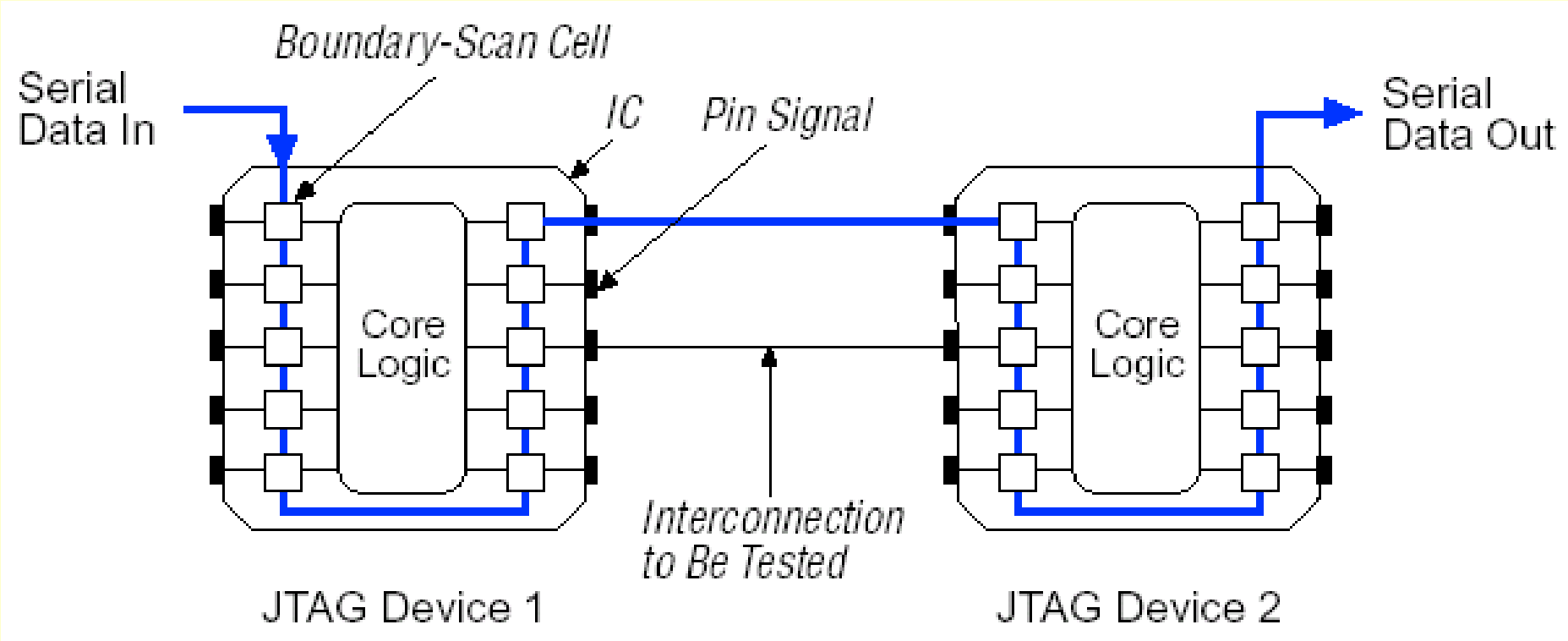


图3-43 边界扫描电路结构

康芯科技

3.5 硬件测试技术

3.5.2 JTAG边界扫描测试

表3-1 边界扫描IO引脚功能

引 脚	描 述	功 能
TDI	测试数据输入(Test Data Input)	测试指令和编程数据的串行输入引脚。数据在TCK的上升沿移入。
TDO	测试数据输出(Test Data Output)	测试指令和编程数据的串行输出引脚，数据在TCK的下降沿移出。如果数据没有被移出时，该引脚处于高阻态。
TMS	测试模式选择(Test Mode Select)	控制信号输入引脚，负责TAP控制器的转换。TMS必须在TCK的上升沿到来之前稳定。
TCK	测试时钟输入(Test Clock Input)	时钟输入到BST电路，一些操作发生在上升沿，而另一些发生在下降沿。
TRST	测试复位输入(Test Reset Input)	低电平有效，异步复位边界扫描电路(在IEEE规范中，该引脚可选)。

3.5 硬件测试技术

3.5.2 JTAG边界扫描测试

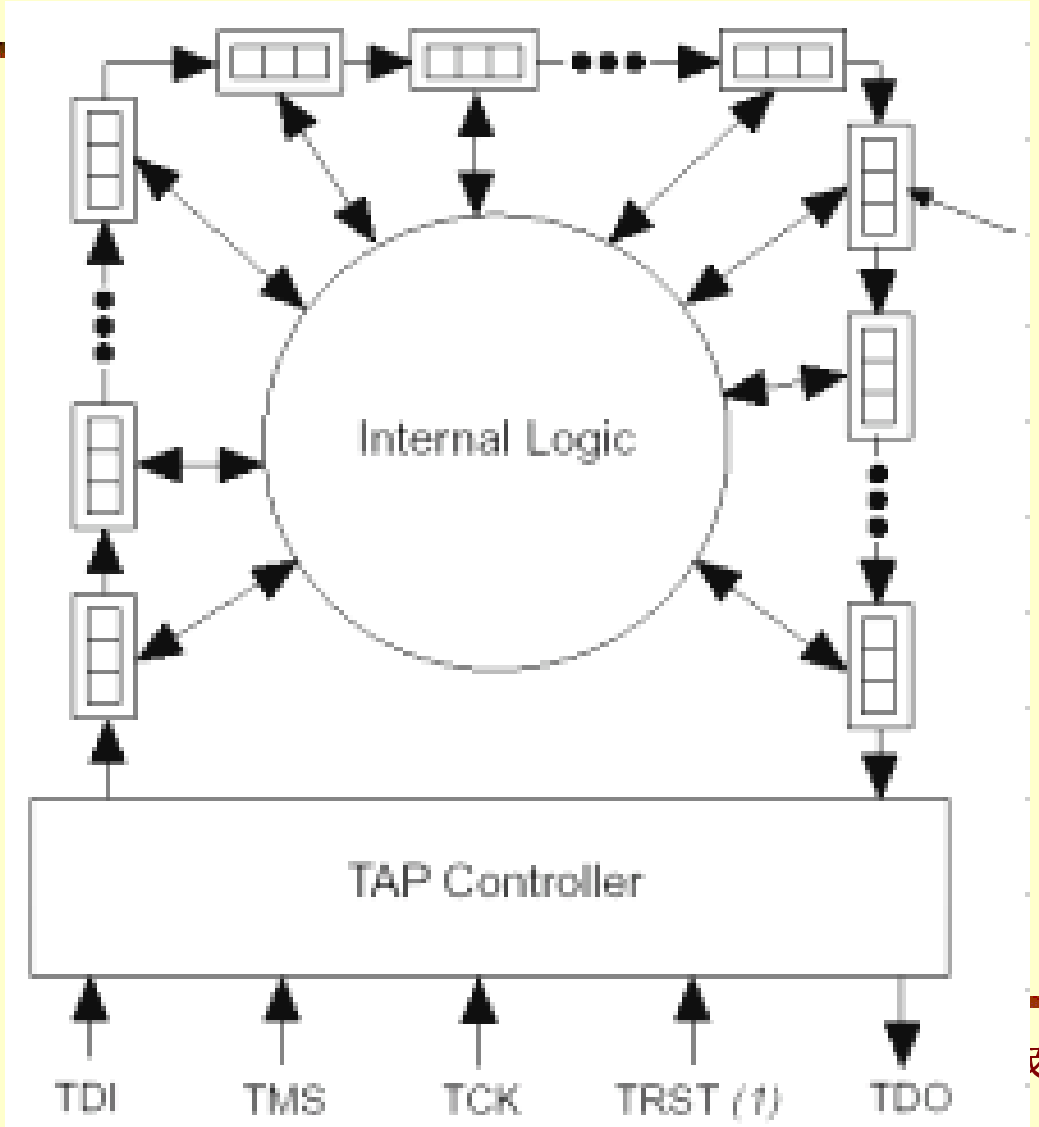


图3-44 边界扫描数据移位方式

3.5.2 JTAG边界扫描测试

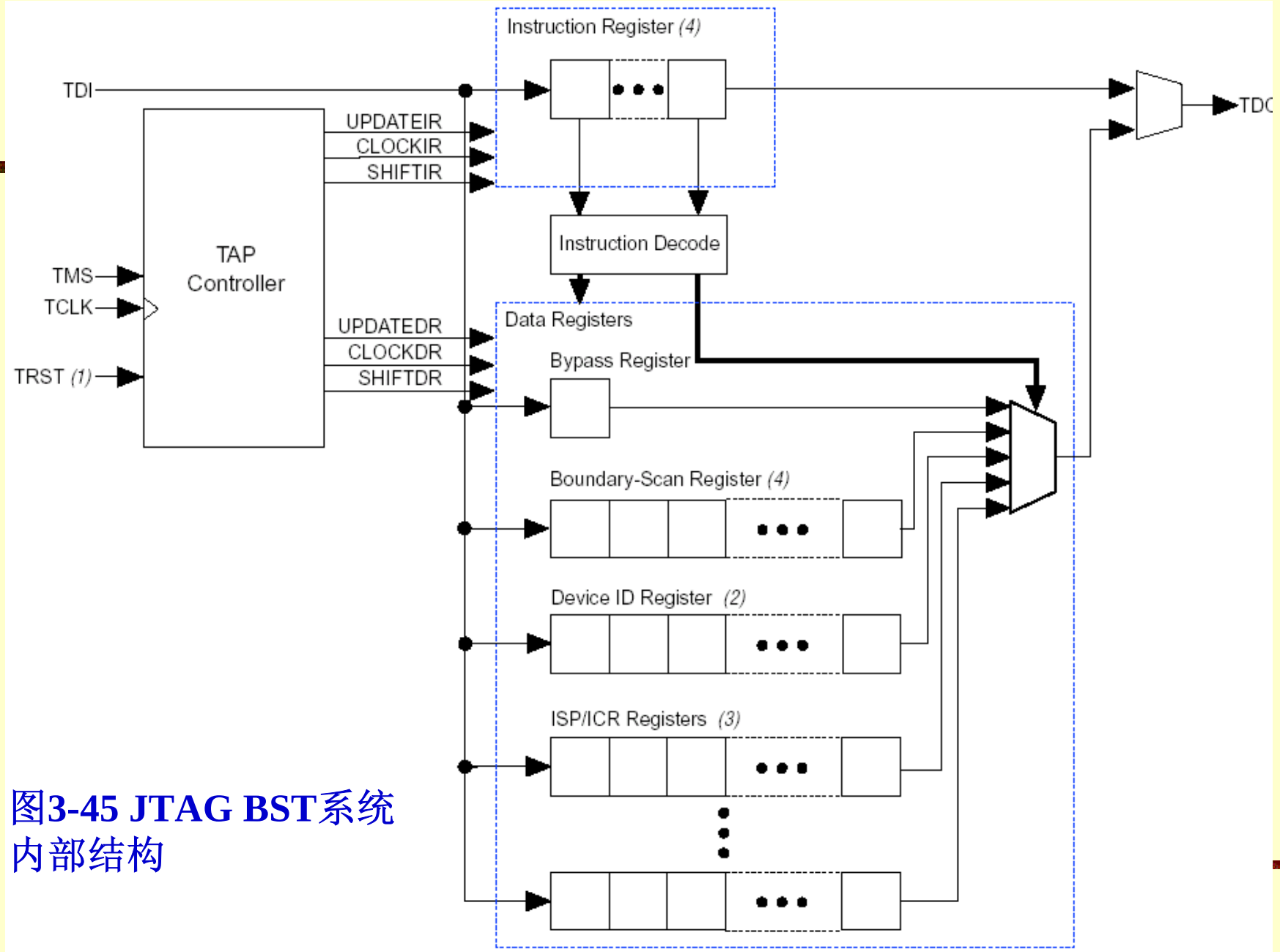


图3-45 JTAG BST系统内部结构

3.5.2 JTAG边界扫描测试

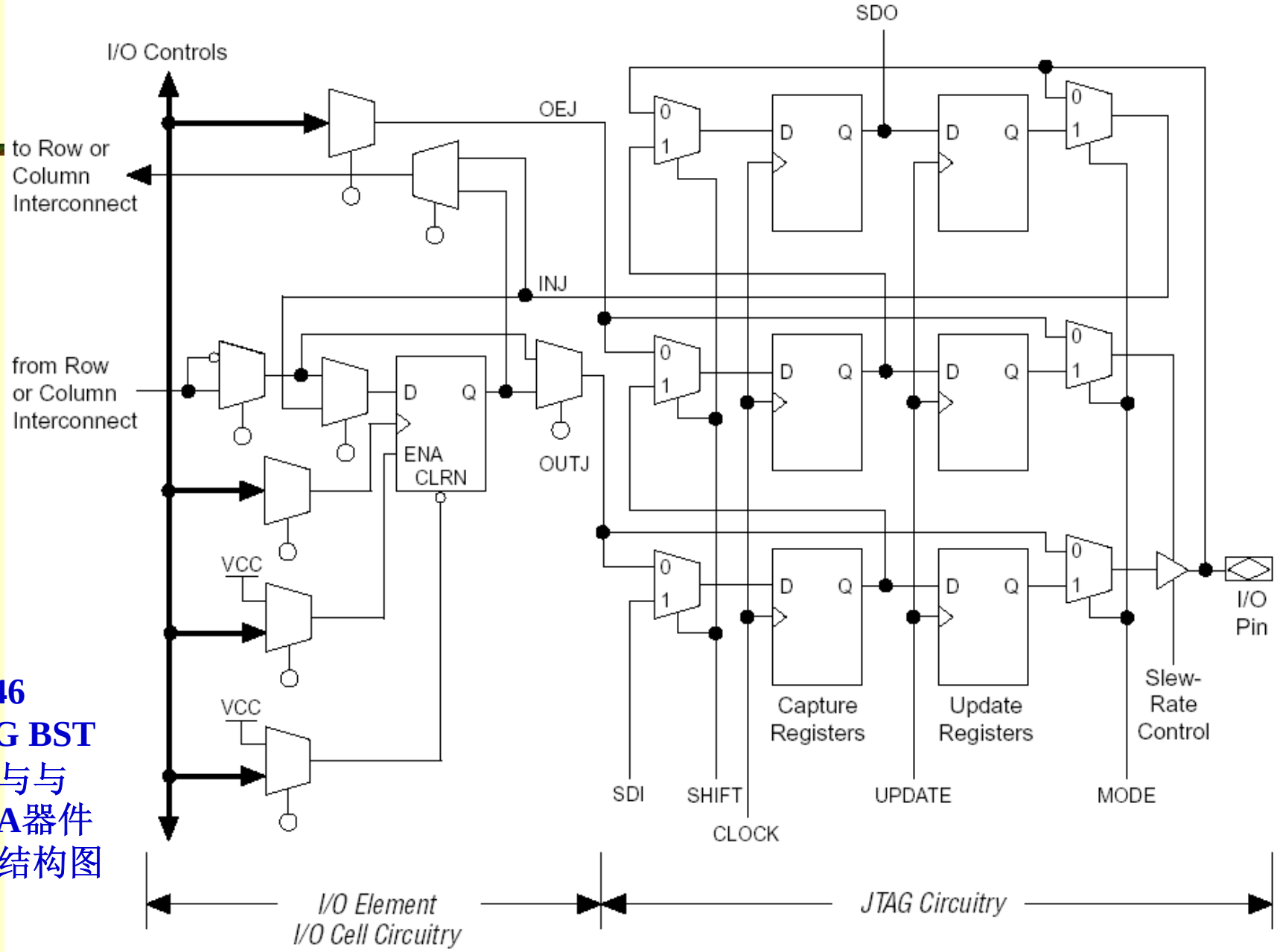


图3-46
JTAG BST
系统与与
FPGA器件
关联结构图

3.5 硬件测试技术

3.5.2 JTAG边界扫描测试

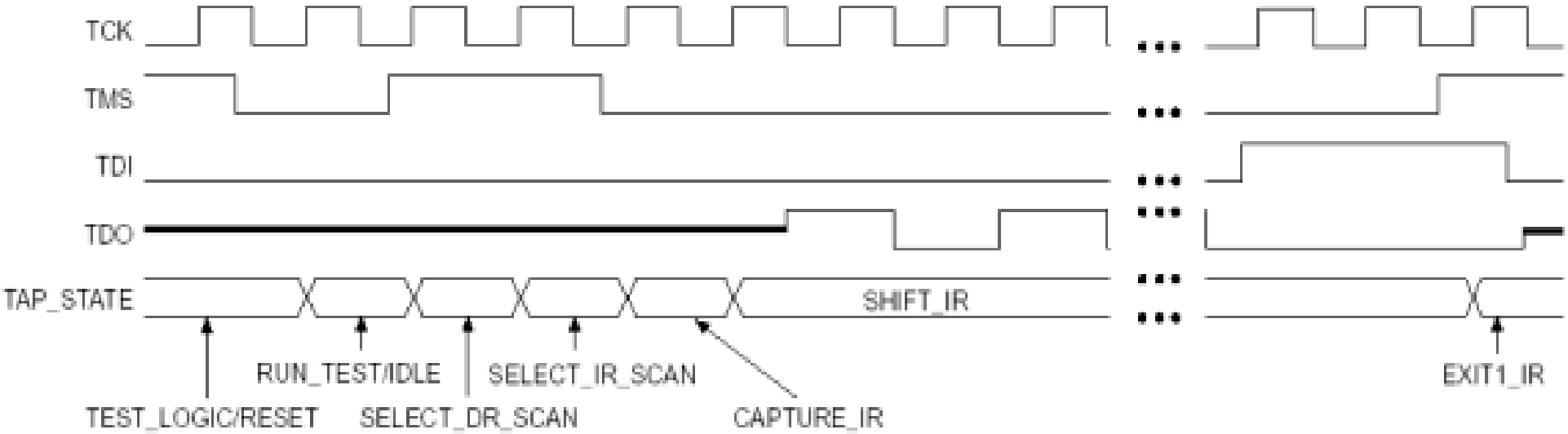


图3-47 JTAG BST选择命令模式时序

3.5.3 嵌入式逻辑分析仪

3.6 FPGA/CPLD产品概述

3.6.1 Lattice公司CPLD器件系列

1. ispLSI器件系列

- ispLSI1000E系列
- ispLSI2000E/2000VL/200VE系列
- ispLSI5000V系列
- ispLSI 8000/8000V系列

2. ispMACH4000系列

- IspMACH 4000Z、ispMACH 4000V 、 ispMACH 4000Z

3. Lattice EC & ECP系列

3.6 FPGA/CPLD产品概述

3.6.2 Xilinx公司的FPGA和CPLD器件系列

1. Virtex-4系列FPGA　Virtex-4 LX　Virtex-4 SX　Virtex-4 FX

2. Spartan Ⅱ & Spartan-3 & Spartan 3E器件系列

3. XC9500 & XC9500XL系列CPLD

4. Xilinx FPGA配置器件SPROM

5. Xilinx的IP核

3.6 FPGA/CPLD产品概述

3.6.3 Altera公司FPGA和CPLD器件系列

1. Stratix II 系列FPGA
2. Stratix系列FPGA
3. ACEX系列FPGA
4. FLEX系列FPGA
5. MAX系列CPLD
6. Cyclone系列FPGA低成本FPGA
7. Cyclone II系列FPGA
8. MAX II系列器件
9. Altera宏功能块及IP核

3.6 FPGA/CPLD产品概述

3.6.4 Actel公司的FPGA器件

3.6.5 Altera公司的FPGA配置方式与配置器件

表3-2 Altera FPGA常用配置器件

器 件	功能描述	封装形式
EPC2	1695680×1位，3.3/5V供电	20脚PLCC、32 脚 TQFP
EPC1	1046496×1位，3.3/5V供电	8脚PDIP、20脚PLCC
EPC1441	440 800×1位，3.3/5V供电	8脚PDIP、20脚PLCC

3.7 编程与配置

编程工艺

- 基于电可擦除存储单元的**EEPROM**或**Flash**技术
- 基于**SRAM**查找表的编程单元
- 基于反熔丝编程单元

表3-3 图3-48接口各引脚信号名称

引脚	1	2	3	4	5	6	7	8	9	10
PS模式	DCK	GND	CONF_DONE	VCC	nCONFIG	-	nSTATUS	-	DATA0	GND
JATG模式	TCK	GND	TDO	VCC	TMS	-	-	-	TDI	GND

3.7 编程与配置

3.7.1 JTAG方式的在系统编程

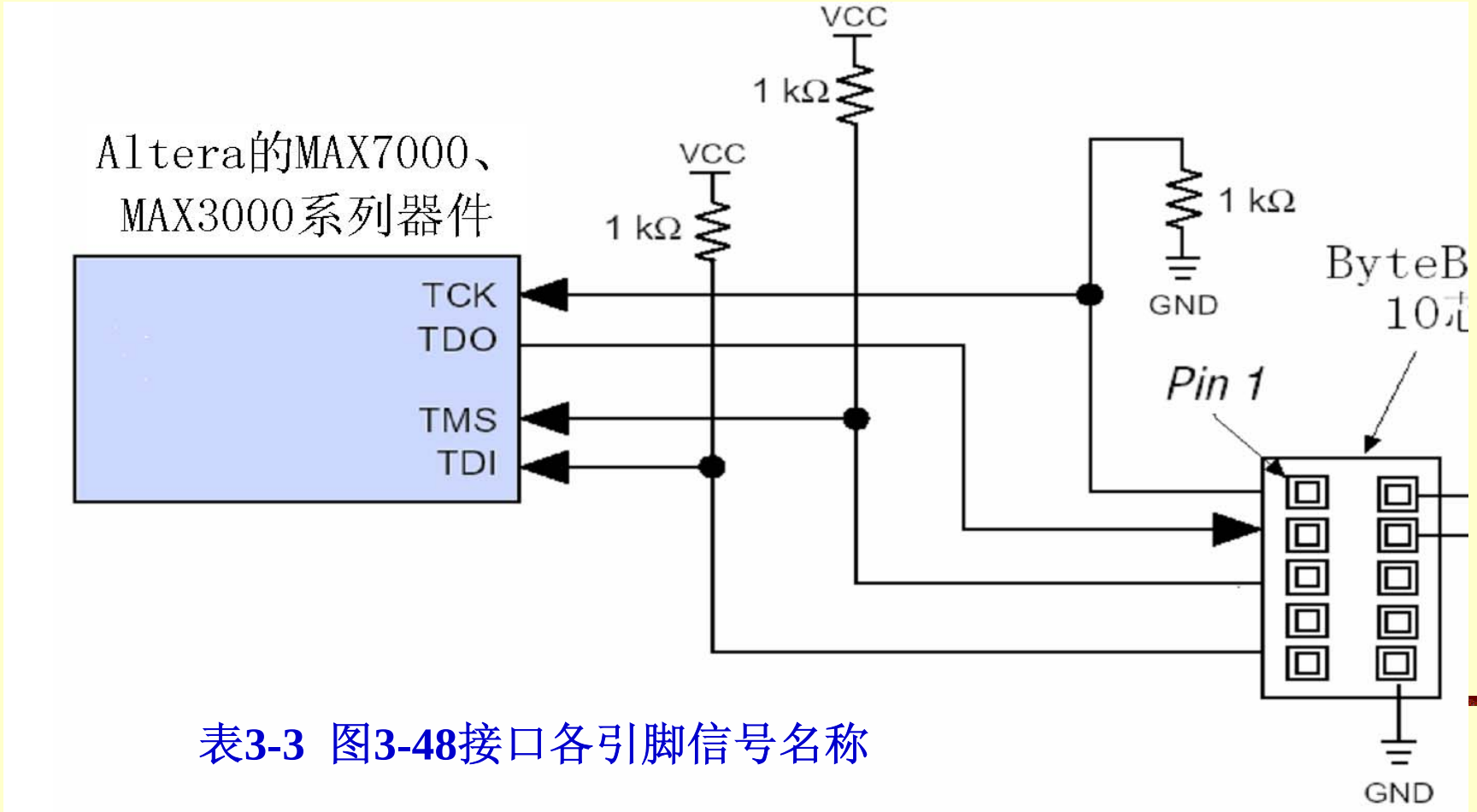


表3-3 图3-48接口各引脚信号名称

3.7 编程与配置

3.7.1 JTAG方式的在系统编程

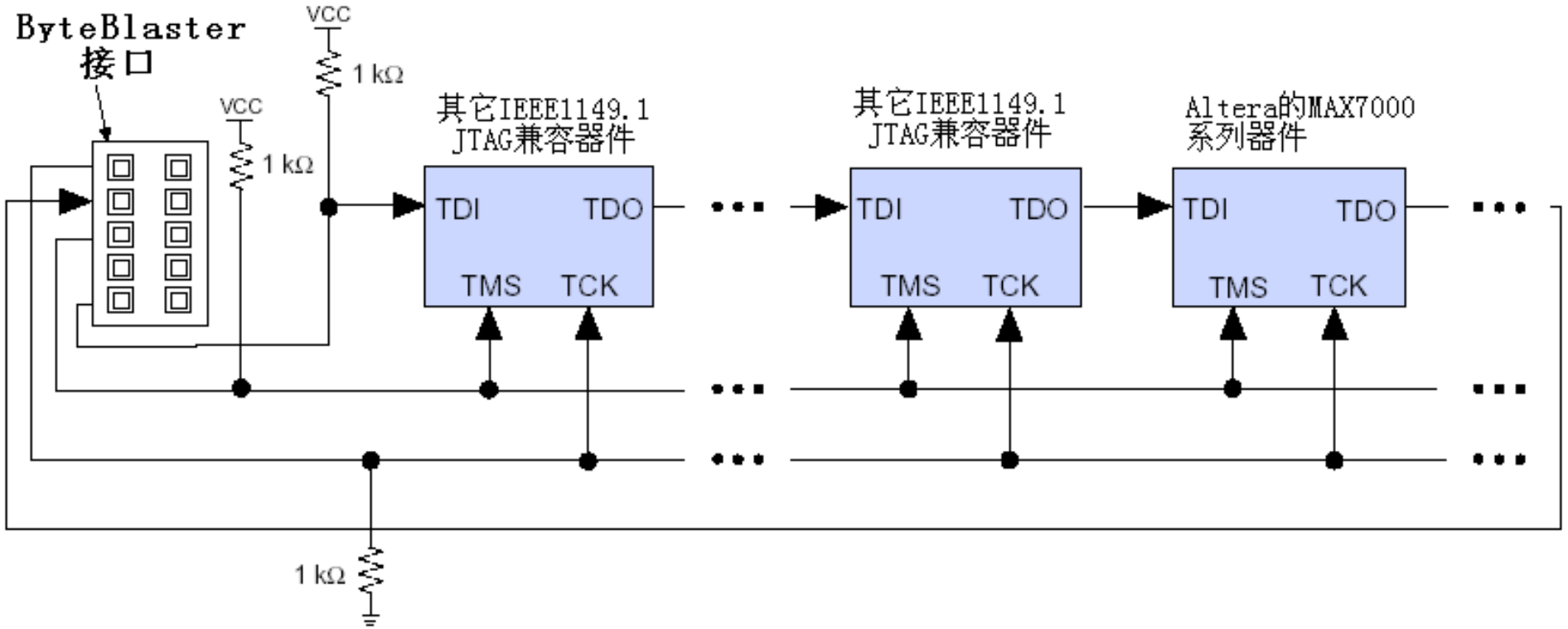


图3-49 多CPLD芯片ISP编程连接方式

3.7 编程与配置

3.7.2 使用PC并行口配置FPGA

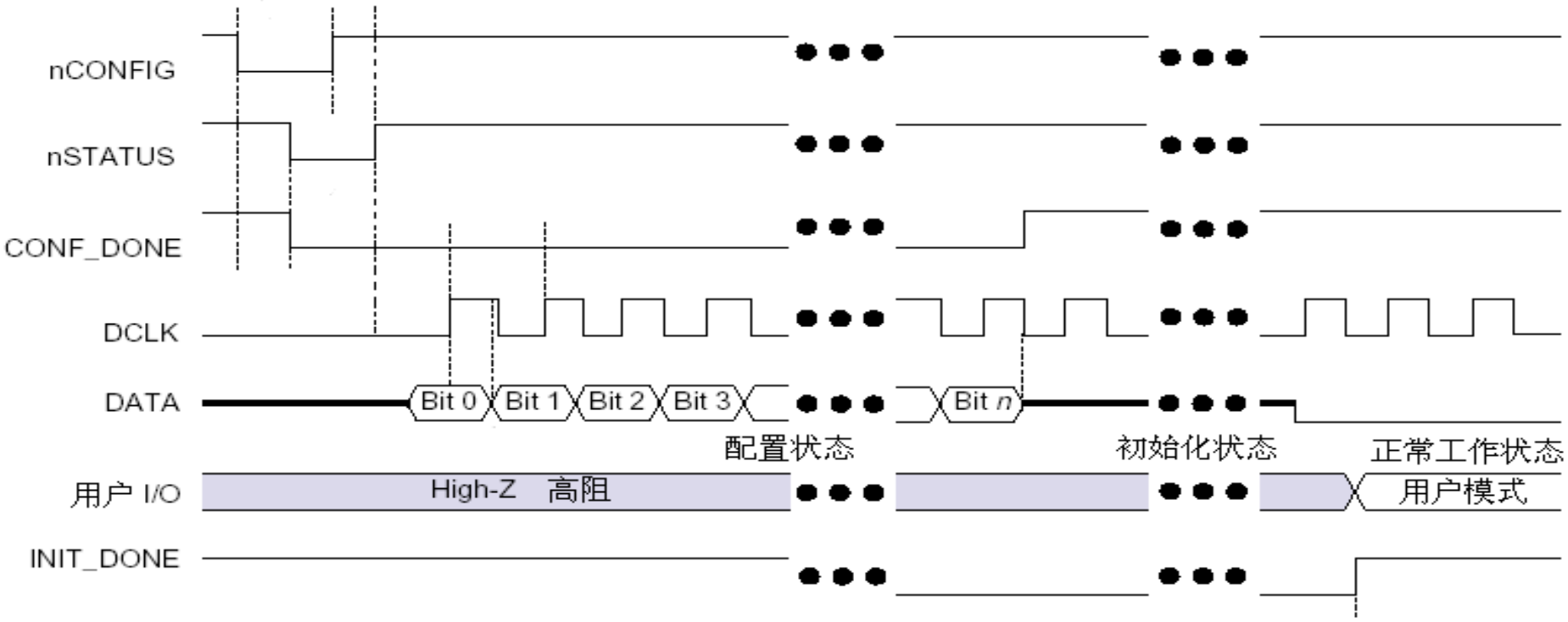


图3-50 PS模式的FPGA配置时序

3.7 编程与配置

3.7.3 FPGA专用配置器件

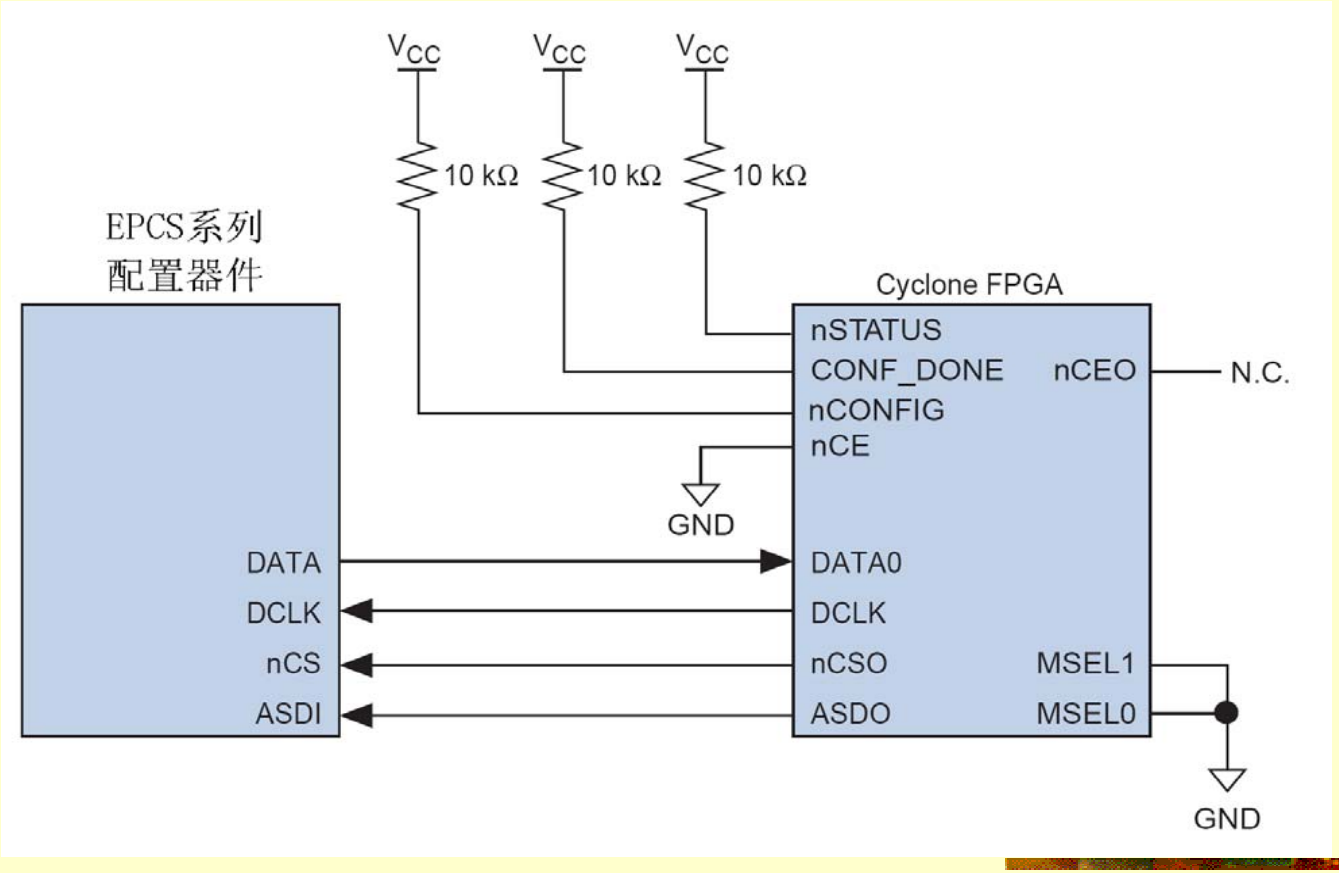


图3-51 EPCS器件配置FPGA的电路原理图

KX 康芯科技

3.7 编程与配置

3.7.4 使用单片机配置FPGA

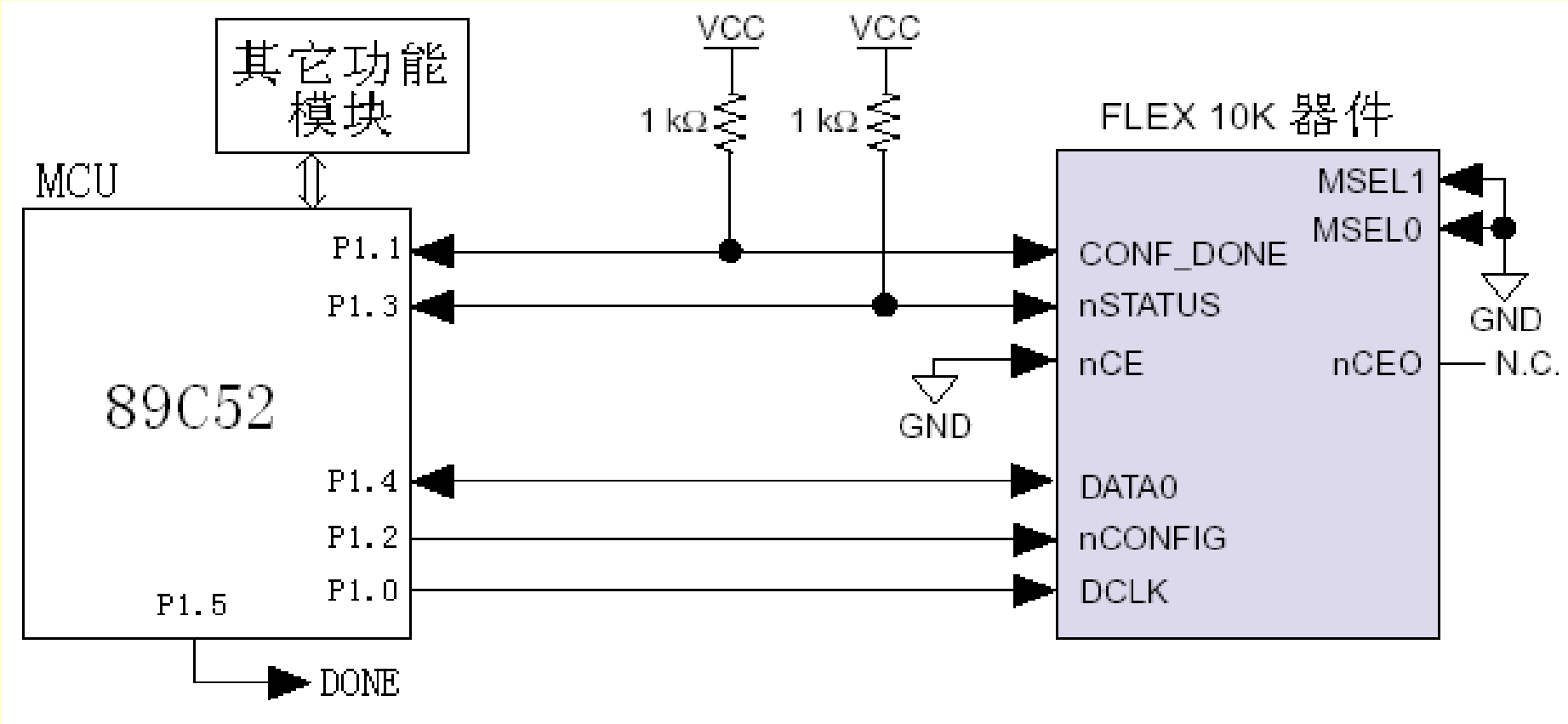


图3-52 用89C52进行配置

3.7 编程与配置

3.7.5 使用CPLD配置FPGA

使用单片机配置的缺点：

1、速度慢，不适用于大规模FPGA和高可靠应用；

2、容量小，单片机引脚少，不适合接大的ROM以存储较大的配置文件；

3、体积大，成本和功耗都不利于相关的设计。

习 题

习题3-1 **OLMC有何功能？说明GAL是怎样实现可编程组合电路与时序电路的。**

习题3-2 什么是基于乘积项的可编程逻辑结构？

习题3-3 什么是基于查找表的可编程逻辑结构？

习题3-4 **FPGA系列器件中的EAB有何作用？**

习题3-5 与传统的测试技术相比，边界扫描技术有何优点？

习题3-6 解释编程与配置这两个概念。

习题3-7 请参阅相关资料，并回答问题：如本章给出的归类方式，将基于乘积项的可编程逻辑结构的**PLD**器件归类为**CPLD**；将基于查找表的可编程逻辑结构的**PLD**器件归类为**FPGA**，那么，**APEX**系列属于什么类型**PLD**器件？**MAX II**系列又属于什么类型的**PLD**器件？为什么？

EDA 技术实用教程

第 4 章

VHDL设计初步

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

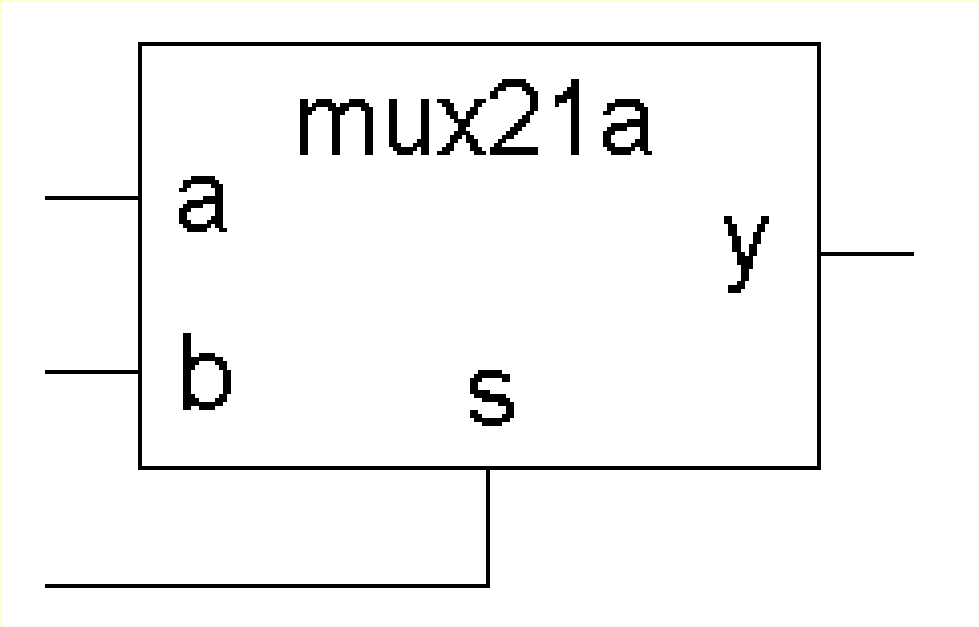


图4-1 mux21a实体

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

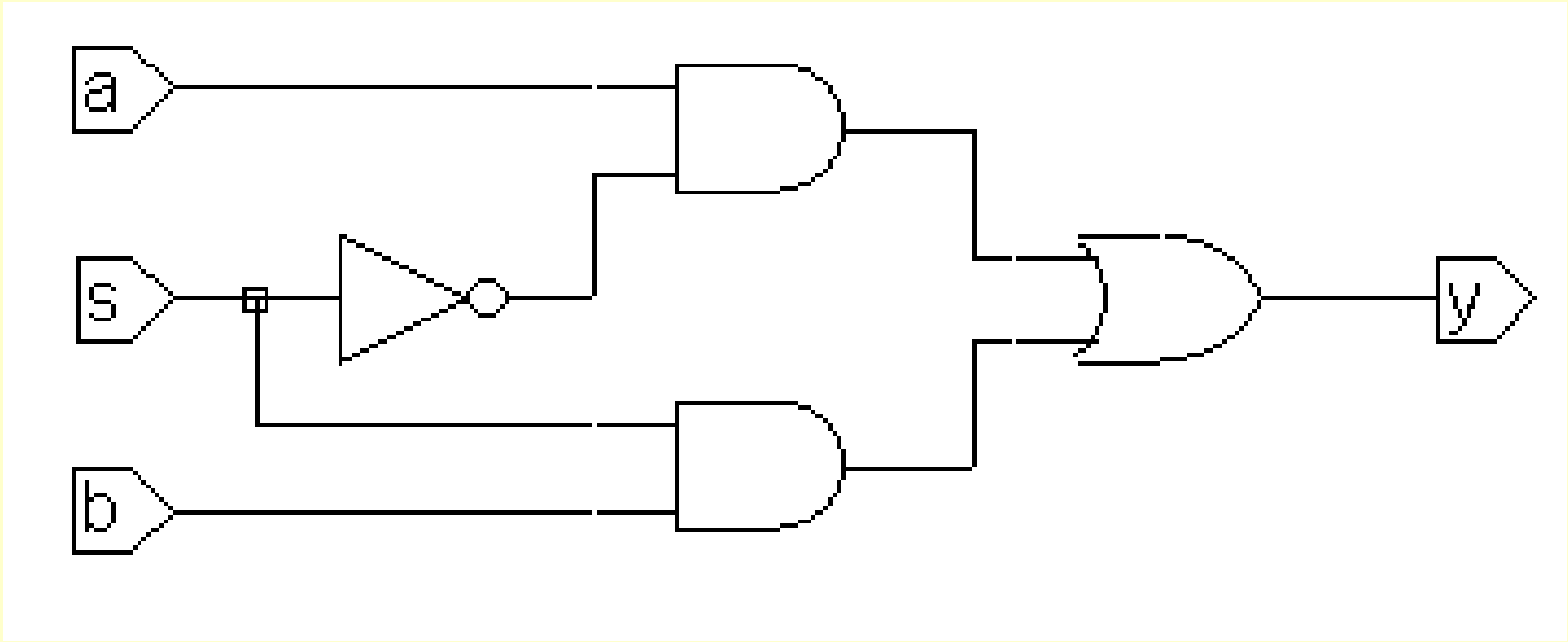


图4-2 mux21a结构体

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

【例4-1】

```
ENTITY mux21a IS
  PORT ( a, b : IN  BIT;
               s : IN  BIT;
               y : OUT BIT  );
END ENTITY mux21a;
ARCHITECTURE one OF mux21a IS
   BEGIN
     y <= a  WHEN  s = '0'  ELSE  b  ;
END ARCHITECTURE one ;
```

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

【例4-2】

```
ENTITY mux21a IS
 PORT ( a, b : IN  BIT;
             s : IN  BIT;
             y : OUT BIT  );
END ENTITY mux21a;
ARCHITECTURE one OF mux21a IS
      SIGNAL d,e :  BIT;
  BEGIN
d <= a AND (NOT S) ;
e <= b AND s ;
y <= d OR e  ;
 END ARCHITECTURE one ;
```

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

【例4-3】

```
 ENTITY mux21a IS
  PORT ( a, b, s: IN  BIT;
                y : OUT BIT  );
END ENTITY mux21a;
ARCHITECTURE one OF mux21a IS
 BEGIN
   PROCESS (a,b,s)
BEGIN
     IF s = '0'  THEN
        y <= a ; ELSE
y <= b ;
END IF;
   END PROCESS;
END ARCHITECTURE one ;
```

4.1 多路选择器的VHDL描述

4.1.1 2选1多路选择器的VHDL描述

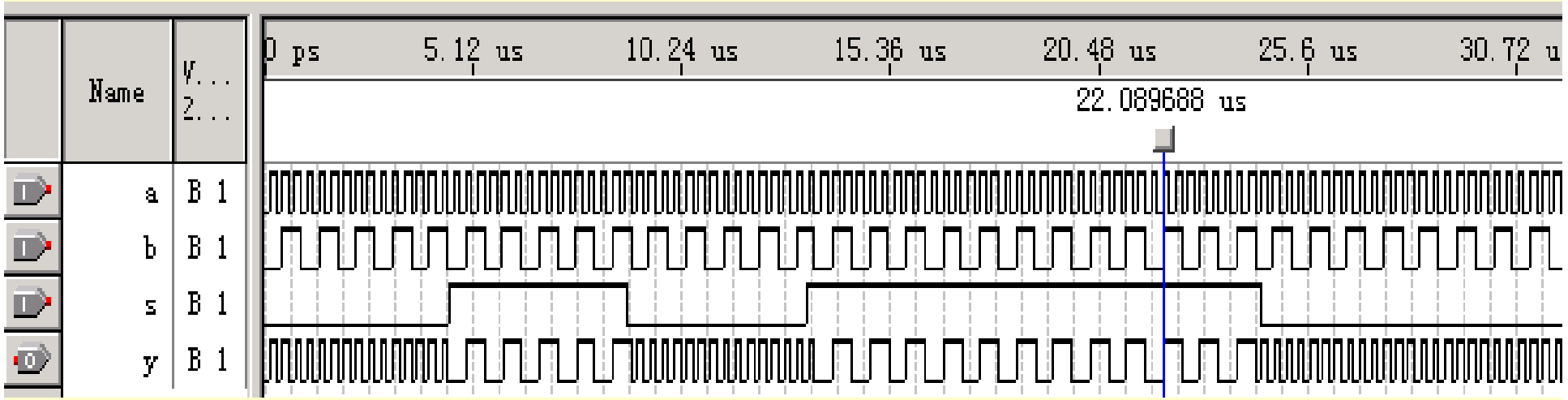


图4-3 mux21a功能时序波形

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

1. 实体表达

【例4-4】

```
ENTITY  e_name  IS
PORT ( p_name :  port_m   data_type;
                 ...
               p_namei : port_mi   data_type );
END ENTITY e_name;
```

2. 实体名

3. 端口语句和端口信号名

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

4. 端口模式

- **IN** 输入端口，定义的通道为单向只读模式
- **OUT** 输出端口，定义的通道为单向输出模式
- **INOUT** 定义的通道确定为输入输出双向端口
- **BUFFER** 缓冲端口，其功能与INOUT类似

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

5. 数据类型

6. 结构体表达

【例4-5】

```
ARCHITECTURE arch_name OF e_name IS
   [说明语句]
BEGIN
   (功能描述语句)
END ARCHITECTURE arch_name ;
```

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

7. 赋值符号和数据比较符号

赋值符 “<=”

表式中的等号“=”没有赋值的含义，只是一种数据比较符号。

```
IF a THEN ...      -- 注意，a的数据类型必须是boolean
  IF (s1='0')AND(s2='1')OR(c<b+1) THEN ..
```

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

8. 逻辑操作符

AND、OR、NOT

9. 条件语句

IF_THEN_ELSE

IF语句必须以语句 **“END IF；”**结束

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

10. WHEN_ELSE条件信号赋值语句

赋值目标 <= 表达式 WHEN 赋值条件 ELSE

表达式 WHEN 赋值条件 ELSE

...

表达式；

```
z  <= a WHEN p1 = '1' ELSE
      b WHEN p2 = '1' ELSE
      c  ;
```

4.1 多路选择器的VHDL描述

4.1.2 相关语句结构和语法说明

11. 进程语句和顺序语句

在一个结构体中可以包含任意个进程语句结构，所有的进程语句都是并行语句，而由任一进程**PROCESS**引导的语句（包含在其中的语句）结构属于顺序语句。

12. 文件取名和存盘

4.2 寄存器描述及其VHDL语言现象

4.2.1 D触发器的VHDL描述

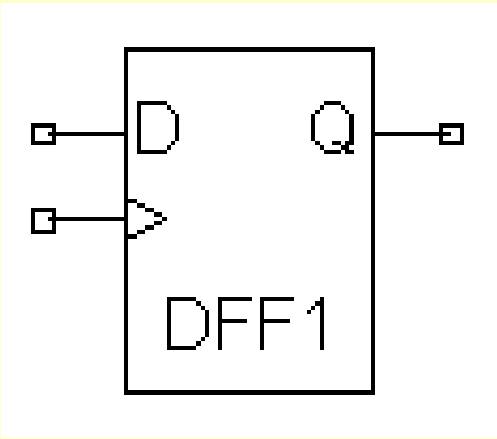


图4-4 D触发器

【例4-6】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DFF1 IS
  PORT (CLK : IN STD_LOGIC ;
            D : IN STD_LOGIC ;
            Q : OUT STD_LOGIC );
 END ;
 ARCHITECTURE bhv OF DFF1 IS
  SIGNAL Q1 : STD_LOGIC ; --类似于在芯片内部定义一个数据的暂存节点
  BEGIN
   PROCESS (CLK,Q1)
    BEGIN
     IF  CLK'EVENT AND CLK = '1'
         THEN  Q1 <= D ;
     END IF;
    END PROCESS ;
Q <= Q1 ;          --将内部的暂存数据向端口输出（双横线--是注释符号）
  END bhv;
```

4.2 寄存器描述及其VHDL语言现象

4.2.2 VHDL描述的语言现象说明

1. 标准逻辑位数据类型STD_LOGIC

BIT数据类型定义：

```
TYPE BIT IS('0','1');        --只有两种取值
```

STD_LOGIC数据类型定义：

```
TYPE STD_LOGIC IS ('U','X','0','1','Z','W','L','H','-');
```

4.2 寄存器描述及其VHDL语言现象

4.2.2 VHDL描述的语言现象说明

2. 设计库和标准程序包

```
LIBRARY  WORK ;
LIBRARY  STD ;
  USE STD.STANDARD.ALL ;
```

使用库和程序包的一般定义表式是：

```
LIBRARY  <设计库名>;
USE  < 设计库名>.<程序包名>.ALL ;
```

4.2 寄存器描述及其VHDL语言现象

4.2.2 VHDL描述的语言现象说明

3. 信号定义和数据对象

“SIGNAL Q1：STD_LOGIC；”

4. 上升沿检测表式和信号属性函数EVENT

```
“CLK'EVENT AND CLK='1'”
```

<信号名>’EVENT

5. 不完整条件语句与时序电路

【例4-7】

```
ENTITY COMP_BAD IS
  PORT( a1, b1  : IN BIT;
    q1  : OUT BIT    );
END ;
ARCHITECTURE one OF COMP_BAD IS
  BEGIN
   PROCESS (a1,b1)
BEGIN
IF  a1 > b1   THEN  q1 <= '1' ;
ELSIF a1 < b1 THEN  q1 <= '0' ;--未提及当a1=b1时，q1作何操作
END IF;
   END PROCESS ;
END ;
```

4.2 寄存器描述及其VHDL语言现象

4.2.2 VHDL描述的语言现象说明

5. 不完整条件语句与时序电路

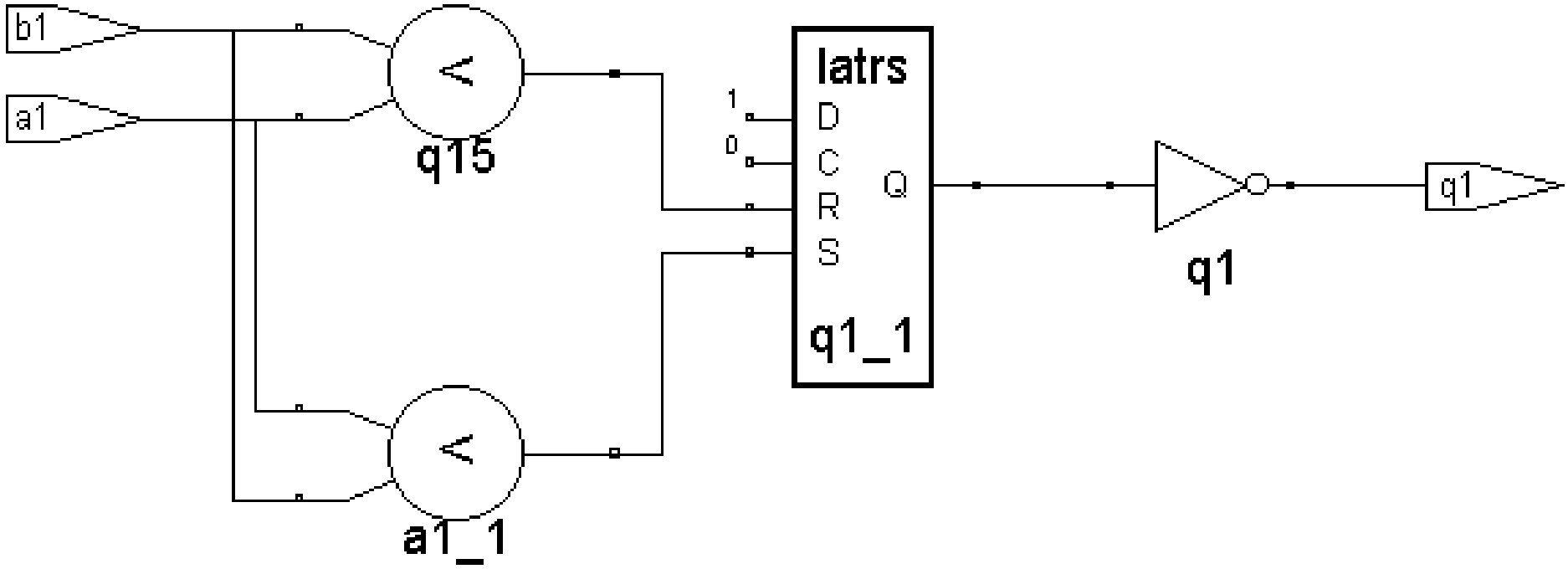


图4-5 例4-7的电路图（Synplify综合）

4.2 寄存器描述及其VHDL语言现象

4.2.2 VHDL描述的语言现象说明

5. 不完整条件语句与时序电路

【例4-8】

```
...
      IF  a1 > b1  THEN  q1 <= '1' ;
      ELSE   q1 <= '0' ;  END IF;
...
```

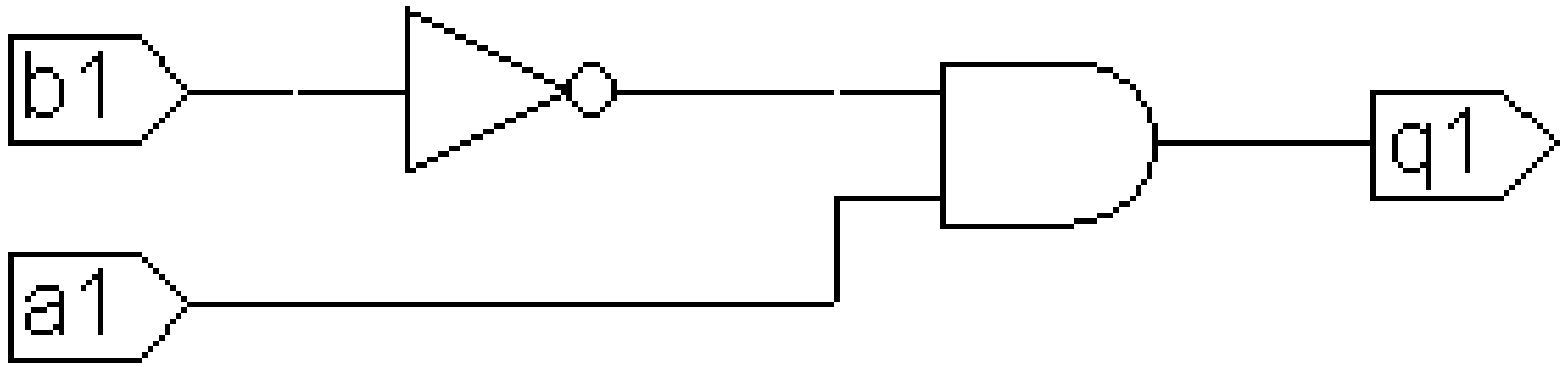


图4-6 例4-8的电路图（Synplify综合）

4.2 寄存器描述及其VHDL语言现象

4.2.3 实现时序电路的VHDL不同表述

【例4-9】

```
...
PROCESS (CLK)
  BEGIN
IF  CLK'EVENT AND (CLK='1') AND (CLK'LAST_VALUE='0')
        THEN  Q <= D ;     --确保CLK的变化是一次上升沿的跳变
     END IF;
END PROCESS ;
```

4.2 寄存器描述及其VHDL语言现象

4.2.3 实现时序电路的VHDL不同表述

【例4-10】

```
...
PROCESS (CLK)
  BEGIN
IF  CLK='1' AND CLK'LAST_VALUE='0'    --同例3-9
        THEN  Q <= D ;
     END IF;
END PROCESS ;
```

【例4-11】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DFF3 IS
  PORT (CLK, D : IN STD_LOGIC ;
              Q : OUT STD_LOGIC );
 END ;
 ARCHITECTURE bhv OF DFF3 IS
   SIGNAL Q1 : STD_LOGIC;
 BEGIN
   PROCESS (CLK)
    BEGIN
    IF  rising_edge(CLK) -- 必须打开STD_LOGIC_1164程序包
    THEN  Q1 <= D ;
    END IF;
     END PROCESS ;
    Q <= Q1 ;    --在此，赋值语句可以放在进程外，作为并行赋值语句
END;
```

4.2 寄存器描述及其VHDL语言现象

4.2.3 实现时序电路的VHDL不同表述

【例4-12】

```
...
PROCESS
  BEGIN
    wait until CLK = '1'   ;      --利用wait语句
     Q <= D ;
END PROCESS;
```

4.2.3 实现时序电路的VHDL不同表述

【例4-13】

```
...
PROCESS (CLK)
   BEGIN
    IF  CLK = '1'
    THEN  Q <= D ; --利用进程的启动特性产生对CLK的边沿检测
    END IF;
END PROCESS ;
```

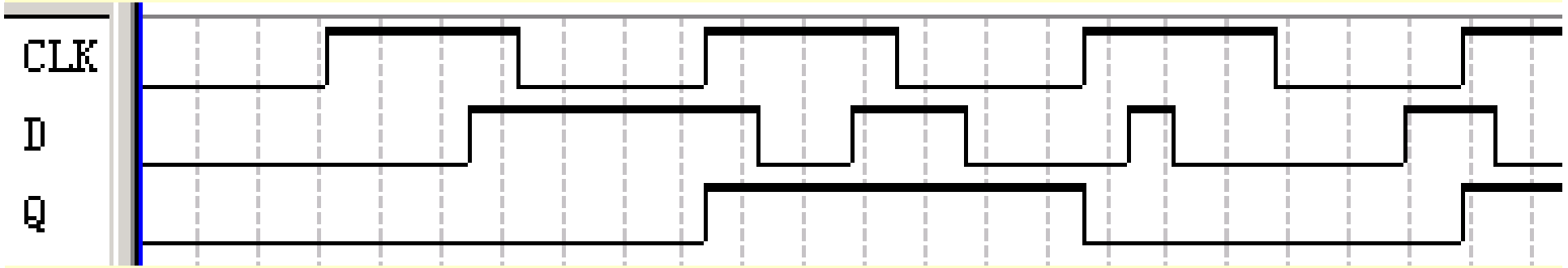


图4-7 例4-13的时序波形

4.2.3 实现时序电路的VHDL不同表述

【例4-14】

```
...
PROCESS (CLK, D)  BEGIN
    IF  CLK = '1'                --电平触发型寄存器
    THEN  Q <= D ;
    END IF;
END PROCESS ;
```

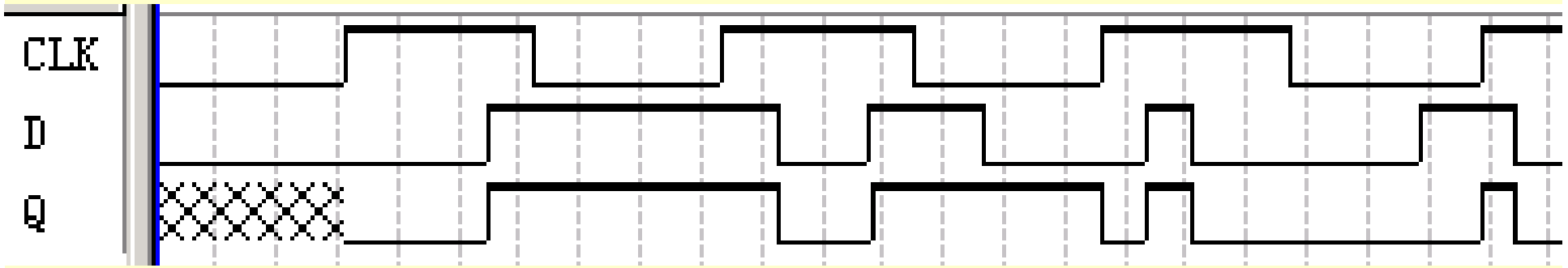


图4-8 例4-14的时序波形

4.2.4 异步时序电路设计

【例4-15】

```
  ...
   ARCHITECTURE bhv OF MULTI_DFF IS
      SIGNAL Q1,Q2 : STD_LOGIC;
   BEGIN
PRO1: PROCESS (CLK)
       BEGIN
        IF  CLK'EVENT AND CLK='1'
          THEN  Q1 <= NOT (Q2 OR A);
        END IF;
       END PROCESS ;
PRO2: PROCESS (Q1)
       BEGIN
        IF  Q1'EVENT AND Q1='1'
          THEN  Q2 <= D;
        END IF;
       END PROCESS ;
QQ <= Q2 ;
  …
```

4.2 寄存器描述及其VHDL语言现象

4.2.4 异步时序电路设计

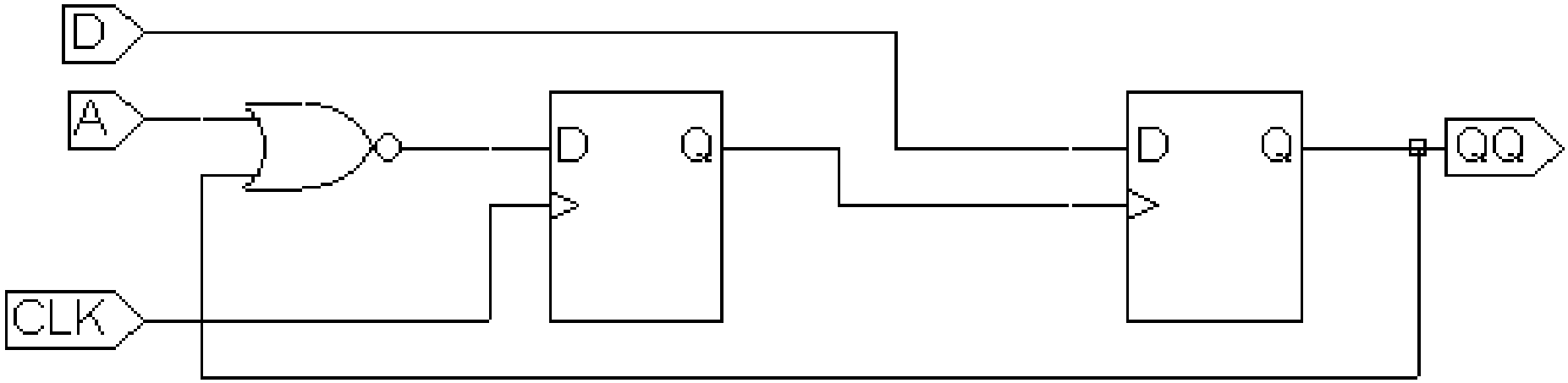
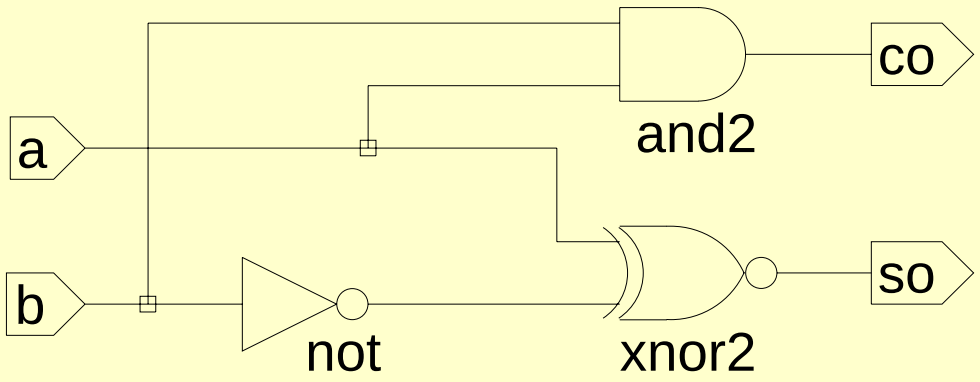


图4-9 例4-15综合后的电路（Synplify综合）

4.3 1位二进制全加器的VHDL描述

4.3.1 半加器描述



a	b	so	co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

图4-10 半加器h_adder电路图及其真值表

4.3 1位二进制全加器的VHDL描述

4.3.1 半加器描述

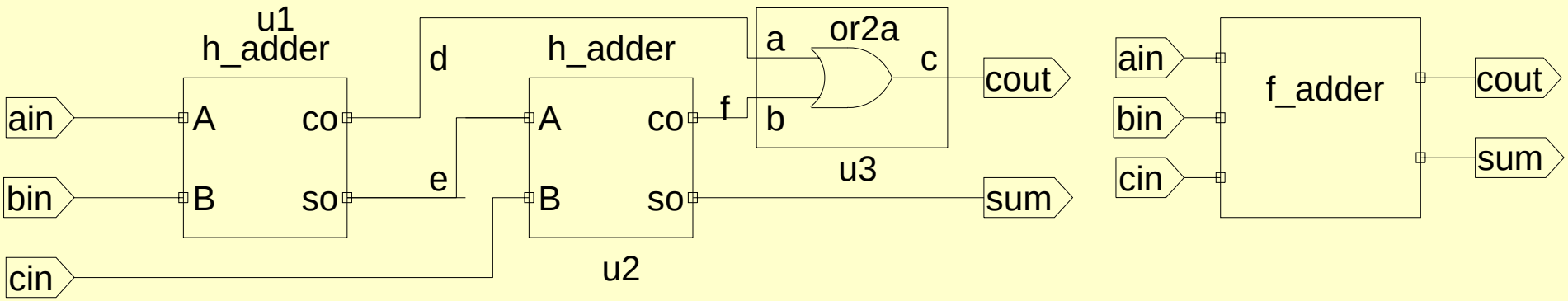


图4-11 全加器f_adder电路图及其实体模块

4.3 1位二进制全加器的VHDL描述

4.3.1 半加器描述

【例4-16】

```
LIBRARY  IEEE;     --半加器描述(1)：布尔方程描述方法
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY h_adder IS
  PORT (a, b : IN STD_LOGIC;
        co, so : OUT STD_LOGIC);
END ENTITY  h_adder;
ARCHITECTURE fh1 OF h_adder  is
BEGIN
  so <= NOT(a XOR (NOT b)) ;   co <= a AND b ;
END ARCHITECTURE fh1;
```

【例4-17】

```
LIBRARY  IEEE;    --半加器描述(2)：真值表描述方法
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY h_adder IS
PORT (a, b : IN STD_LOGIC;
      co, so : OUT STD_LOGIC);
END ENTITY h_adder;
ARCHITECTURE fh1 OF h_adder  is
 SIGNAL abc : STD_LOGIC_VECTOR(1 DOWNTO 0) ; --定义标准逻辑位矢量
数据类型
BEGIN
  abc <= a & b ;    --a相并b，即a与b并置操作
 PROCESS(abc)
  BEGIN
   CASE abc IS        --类似于真值表的CASE语句
    WHEN "00" => so<='0'; co<='0' ;
    WHEN "01" => so<='1'; co<='0' ;
    WHEN "10" => so<='1'; co<='0' ;
    WHEN "11" => so<='0'; co<='1' ;
    WHEN OTHERS => NULL ;
   END CASE;
 END PROCESS;
END ARCHITECTURE fh1 ;
```

4.3 1位二进制全加器的VHDL描述

4.3.1 半加器描述

【例4-18】

```
LIBRARY  IEEE ;    --或门逻辑描述
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY or2a IS
  PORT (a, b :IN STD_LOGIC;
             c : OUT STD_LOGIC );
END ENTITY or2a;
ARCHITECTURE one OF or2a IS
  BEGIN
  c <= a OR b ;
END ARCHITECTURE one ;
```

【例4-19】

```
LIBRARY  IEEE;    --1位二进制全加器顶层设计描述
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY f_adder IS
  PORT (ain，bin，cin  : IN STD_LOGIC;
              cout，sum   : OUT STD_LOGIC );
END ENTITY f_adder;
ARCHITECTURE fd1 OF f_adder IS
  COMPONENT h_adder                          --调用半加器声明语句
    PORT (  a，b :  IN STD_LOGIC;
      co，so :  OUT STD_LOGIC);
  END COMPONENT ；
  COMPONENT or2a
     PORT (a，b : IN STD_LOGIC;
                c : OUT STD_LOGIC);
  END COMPONENT;
SIGNAL d，e，f  :  STD_LOGIC; --定义3个信号作为内部的连接线。
  BEGIN
  u1 : h_adder PORT MAP(a=>ain，b=>bin，co=>d，so=>e);--例化语句
  u2 : h_adder PORT MAP(a=>e，   b=>cin，  co=>f，so=>sum);
  u3 :   or2a   PORT MAP(a=>d，   b=>f，    c=>cout);
END ARCHITECTURE fd1;
```

4.3 1位二进制全加器的VHDL描述

4.3.2 CASE语句

1. CASE语句

```
CASE <表达式> IS
When <选择值或标识符> => <顺序语句>; ... ; <顺序语句> ;
When <选择值或标识符> => <顺序语句>; ... ; <顺序语句> ;
...
WHEN OTHERS => <顺序语句>;
END CASE ;
```

4.3 1位二进制全加器的VHDL描述

4.3.2 CASE语句

2. 标准逻辑矢量数据类型

STD_LOGIC_VECTOR　　STD_LOGIC

在使用STD_LOGIC_VECTOR中，必须注明其数组宽度，即位宽，如：

```
B : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) ;
```

或 `SIGNAL A : STD_LOGIC_VECTOR(1 TO 4)`

```
B <= "01100010" ;                -- B(7)为 '0'
B(4 DOWNTO 1) <= "1101" ;    -- B(4)为 '1'
  B(7 DOWNTO 4) <= A ;           -- B(6)等于 A(2)
```

4.3 1位二进制全加器的VHDL描述

4.3.2 CASE语句

3. 并置操作符 &

```
SIGNAL a : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL d : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
...
a  <= '1' & '0' & d(1) & '1'  ;   -- 元素与元素并置，并置后的数组长度为4
...
```

IF a & d = "101011" THEN ... –- 在IF条件句中可以使用并置符

4.3 1位二进制全加器的VHDL描述

4.3.3 全加器描述和例化语句

```
COMPONENT 元件名 IS
PORT  (端口名表) ;
END COMPONENT 文件名 ;
```

```
COMPONENT h_adder
        PORT (  c, d :  IN STD_LOGIC;
                e, f :  OUT STD_LOGIC);
```

例化名 : 元件名 PORT MAP([端口名 =>] 连接端口名,...);

4.4 计数器设计

【例4-20】

```
ENTITY CNT4 IS
  PORT ( CLK : IN BIT ;
           Q  : BUFFER INTEGER RANGE 15 DOWNTO 0  ) ;
  END ;
ARCHITECTURE bhv OF CNT4 IS
  BEGIN
  PROCESS (CLK)
    BEGIN
      IF  CLK'EVENT AND CLK = '1'  THEN
        Q <= Q + 1 ;
      END IF;
  END PROCESS ;
END bhv;
```

4.4 计数器设计

4.4.1 4位二进制加法计数器设计

表面上，**BUFFER**具有双向端口**INOUT**的功能，但实际上其输入功能是不完整的，它只能将自己输出的信号再反馈回来，并不含有IN的功能。

表式**Q <= Q + 1**的右项与左项并非处于相同的时刻内，对于时序电路，除了传输延时外，前者的结果出现于当前时钟周期；后者，即左项要获得当前的**Q + 1**，需等待下一个时钟周期。

4.4 计数器设计

4.4.2 整数类型

Q : BUFFER INTEGER RANGE 15 DOWNTO 0;

整数常量的书写方式示例

```
1                    十进制整数
0                    十进制整数
35                   十进制整数
10E3                 十进制整数，等于十进制整数1000
16#D9#               十六进制整数，等于十六进制整数D9H
8#720#               八进制整数，等于八进制整数720O
2#11010010#    二进制整数，等于二进制整数11010010B
```

Q : BUFFER NATURAL RANGE 15 DOWNTO 0;

4.4.3 计数器设计的其他表述方法

【例4-21】

```
 LIBRARY IEEE ;
 USE IEEE.STD_LOGIC_1164.ALL ;
 USE IEEE.STD_LOGIC_UNSIGNED.ALL ;
 ENTITY CNT4 IS
 PORT ( CLK : IN STD_LOGIC ;
           Q  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)  ) ;
 END ;
 ARCHITECTURE bhv OF CNT4 IS
SIGNAL Q1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
 BEGIN
   PROCESS (CLK)
   BEGIN
       IF  CLK'EVENT AND CLK = '1'  THEN
   Q1 <= Q1 + 1 ;
       END IF;
   END PROCESS ;
       Q <= Q1 ;
END bhv;
```

4.4 计数器设计

4.4.3 计数器设计的其他表述方法

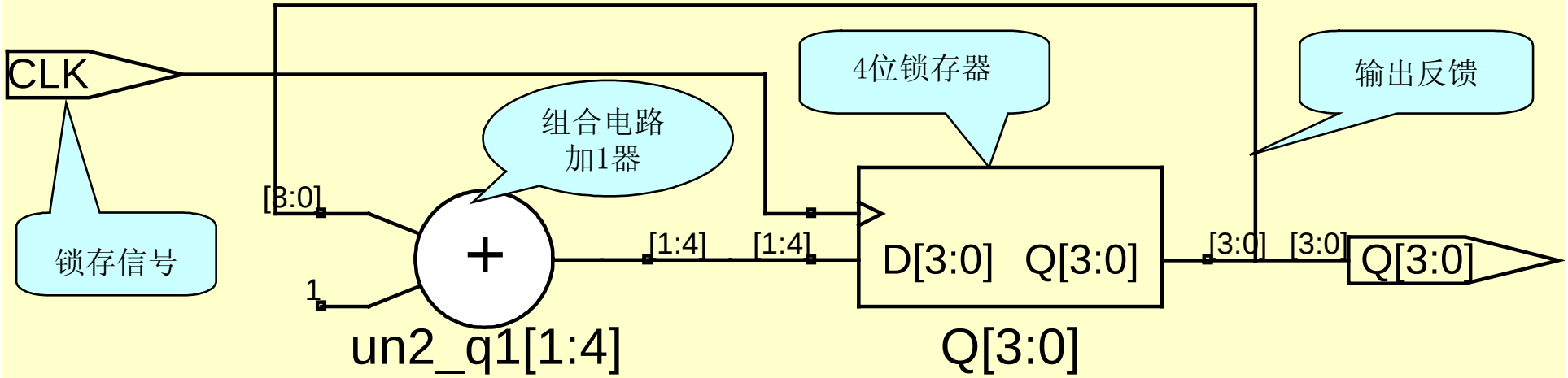


图4-12 4位加法计数器RTL电路（Synplify综合）

4.4 计数器设计

4.4.3 计数器设计的其他表述方法

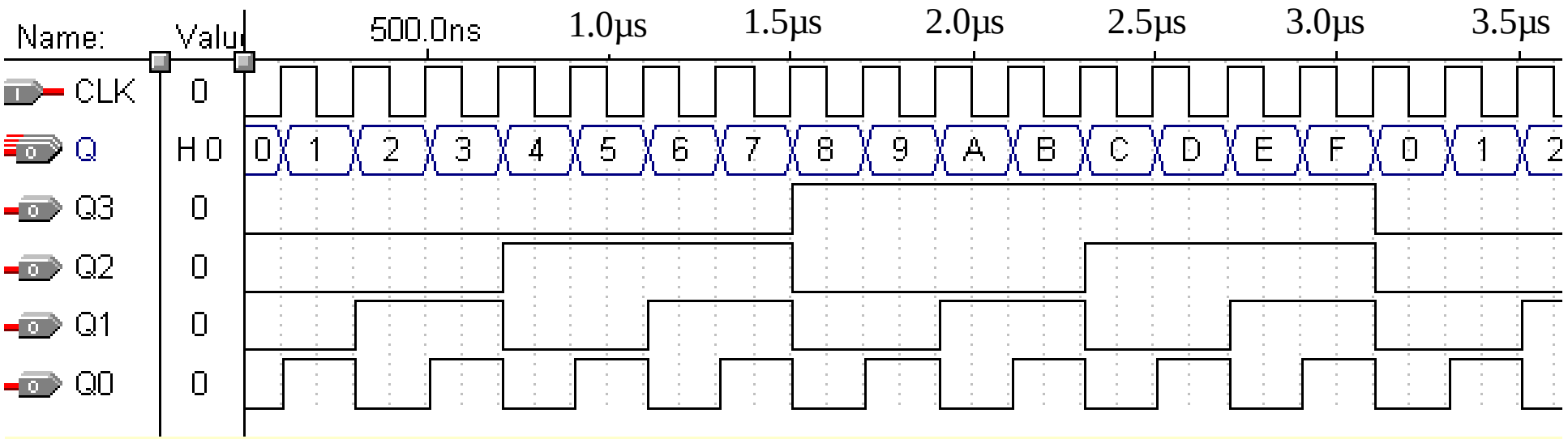


图4-13 4位加法计数器工作时序

4.5 一般加法计数器设计

【例4-22】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT10 IS
    PORT (CLK,RST,EN : IN STD_LOGIC;
                  CQ : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
COUT : OUT STD_LOGIC  );
END CNT10;
ARCHITECTURE behav OF CNT10 IS
BEGIN
   PROCESS(CLK, RST, EN)
     VARIABLE  CQI : STD_LOGIC_VECTOR(3 DOWNTO 0);
   BEGIN
     IF RST = '1' THEN   CQI := (OTHERS =>'0') ; --计数
                                                  器异步复位
       ELSIF CLK'EVENT AND CLK='1' THEN     --检测时钟上升沿
```

接下页

4.5 一般加法计数器设计

```
IF EN = '1' THEN          -检测是否允许计数（同步使能）
          IF CQI < 9 THEN   CQI := CQI + 1; --允许计数,
                                              检测是否小于9
            ELSE    CQI := (OTHERS =>'0');      --大于9,
                                                  计数值清零
          END IF;
        END IF;
      END IF;
       IF CQI = 9 THEN COUT <= '1'; --计数大于9，输出进位信号
         ELSE    COUT <= '0';
       END IF;
         CQ <= CQI;          --将计数值向端口输出
   END PROCESS;
END behav;
```

4.5 一般加法计数器设计

4.5.1 相关语法说明

1. 变量

```
VARIABLE  CQI : STD_LOGIC_VECTOR(3 DOWNTO 0)
```

2. 省略赋值操作符(OTHERS=>X)

```
SIGNAL    d1  : STD_LOGIC_VECTOR(4 DOWNTO 0);
VARIABLE  a1 : STD_LOGIC_VECTOR(15 DOWNTO 0);
 ...
  d1 <= (OTHERS=>'0');   a1 := (OTHERS=>'0') ;
d1 <= (1=>e(3),3=>e(5), OTHERS=>e(1) );
f <= e(1) & e(5) & e(1) & e(3) & e(1) ;
```

4.5 一般加法计数器设计

4.5.2 程序分析

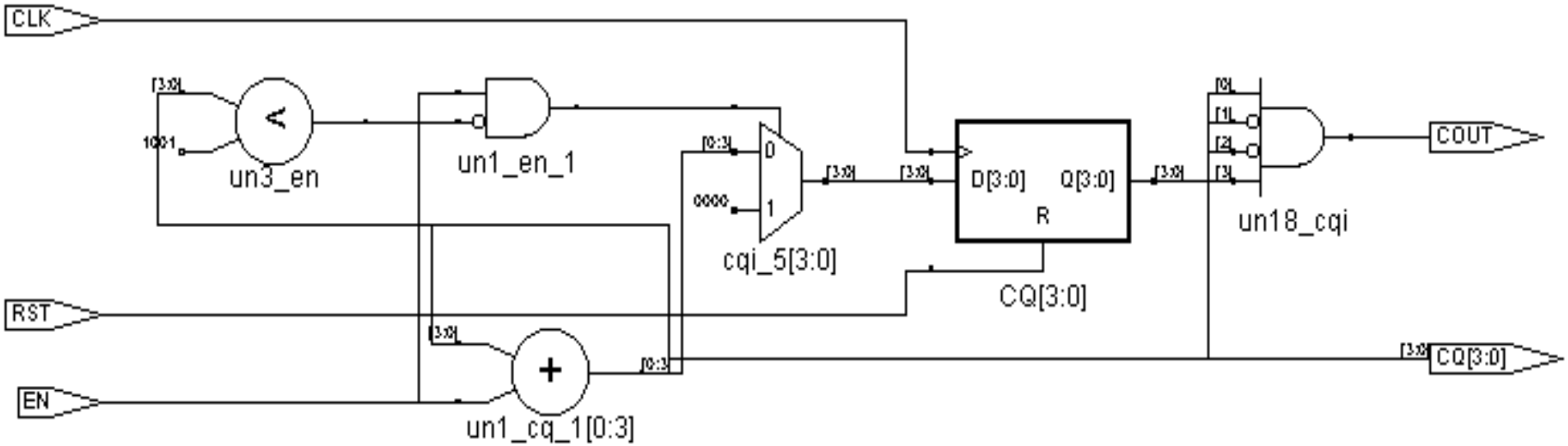


图4-14 例4-22的RTL电路（Synplify综合）

4.5 一般加法计数器设计

4.5.2 程序分析

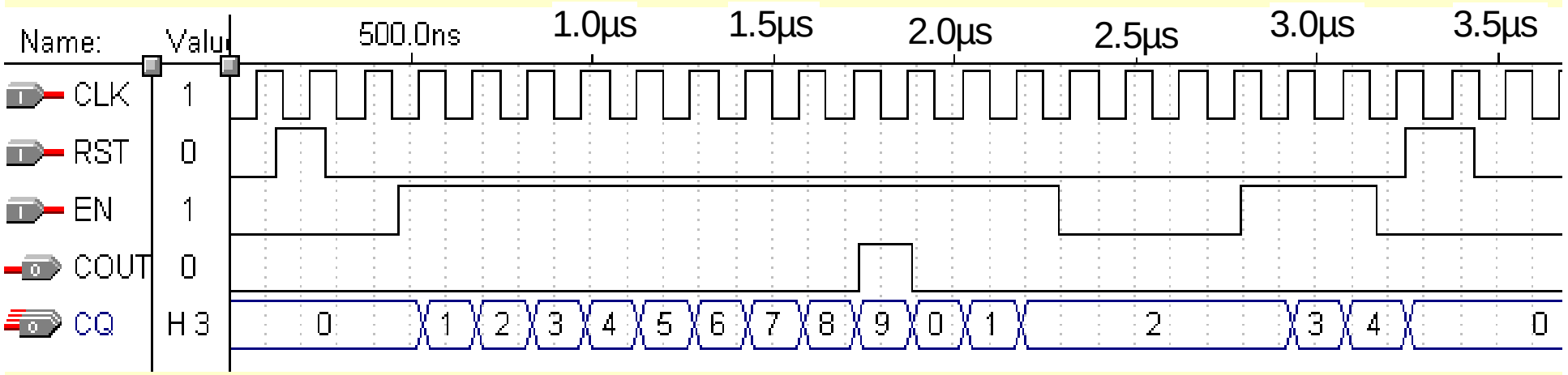


图4-15 例4-22的工作时序

4.5.3 含并行置位的移位寄存器设计

【例4-23】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SHFRT IS                          -- 8位右移寄存器
    PORT (  CLK，LOAD : IN STD_LOGIC;
                  DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                   QB : OUT STD_LOGIC  );
END SHFRT;
ARCHITECTURE behav OF SHFRT IS
    BEGIN
    PROCESS (CLK, LOAD)
      VARIABLE REG8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
    BEGIN
         IF CLK'EVENT AND CLK = '1' THEN
            IF LOAD = '1' THEN  REG8 := DIN;  --由（LOAD='1'）装
载新数据
              ELSE  REG8(6 DOWNTO 0) := REG8(7 DOWNTO 1);
END IF;
         END IF;
            QB <= REG8(0); -- 输出最低位
    END PROCESS;
END behav;
```

4.5 一般加法计数器设计

4.5.3 含并行置位的移位寄存器设计

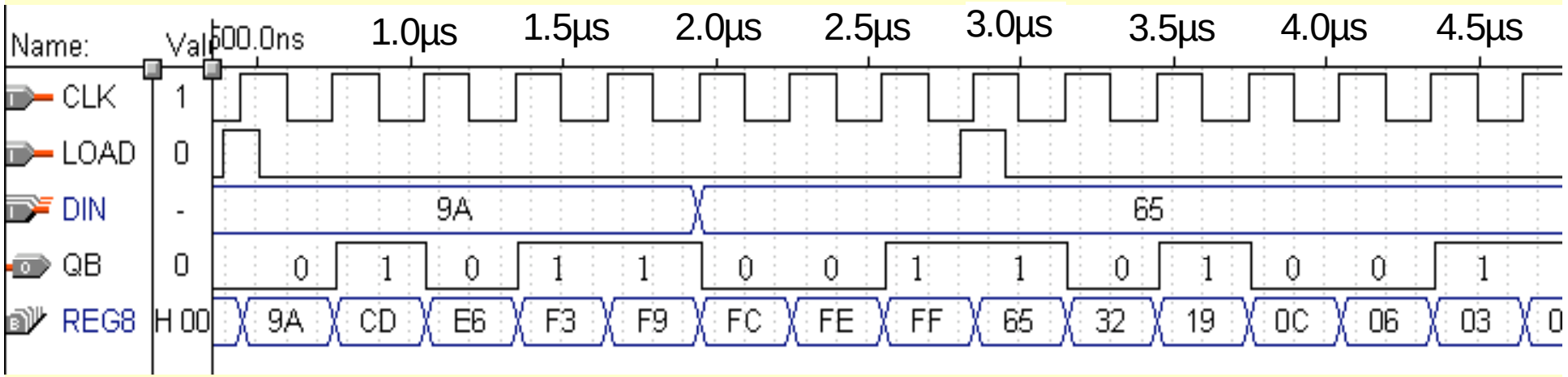


图4-16 例4-23的工作时序

习 题

4-1. 画出与下例实体描述对应的原理图符号元件：

```
ENTITY buf3s IS          -- 实体1：  三态缓冲器
    PORT (input : IN STD_LOGIC ;        -- 输入端
          enable : IN STD_LOGIC ;       -- 使能端
          output : OUT STD_LOGIC ) ;    -- 输出端
END buf3x ;
ENTITY mux21 IS          --实体2：   2选1多路选择器
    PORT (in0,  in1, sel :  IN STD_LOGIC;
           output : OUT STD_LOGIC);
```

习 题

4-2. 图4-17所示的是4选1多路选择器，试分别用IF_THEN语句和CASE语句的表达方式写出此电路的VHDL程序。

选择控制的信号s1和s0的数据类型为STD_LOGIC_VECTOR;

当s1='0'，s0='0'；s1='0'，s0='1'；s1='1'，s0='0'和s1='1'，s0='1'分别执行y<=a、y<=b、y<=c、y<=d。

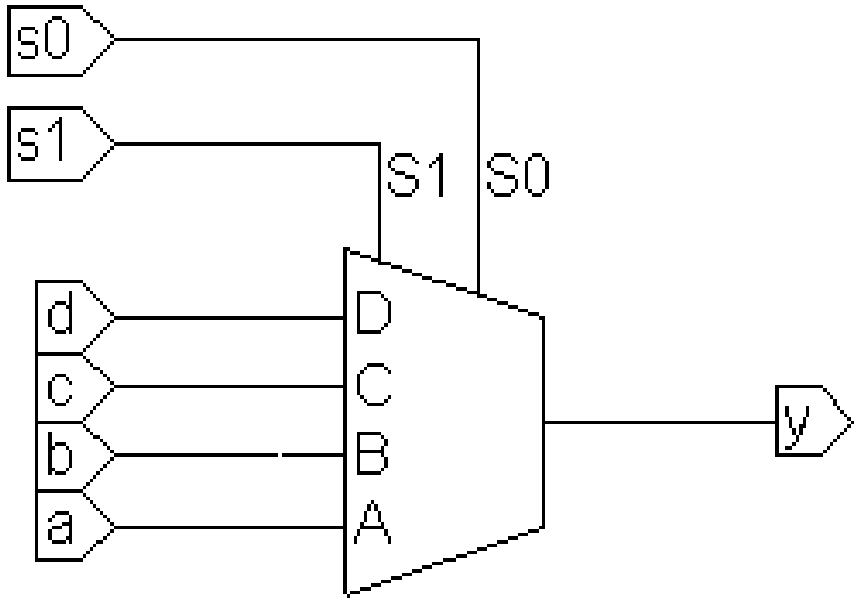


图4-17 4选1多路选择器

习 题

4-3. 图4-18所示的是双2选1多路选择器构成的电路MUXK，对于其中MUX21A，当s='0'和'1'时，分别有y<='a'和y<='b'。试在一个结构体中用两个进程来表达此电路，每个进程中用CASE语句描述一个2选1多路选择器MUX21A。

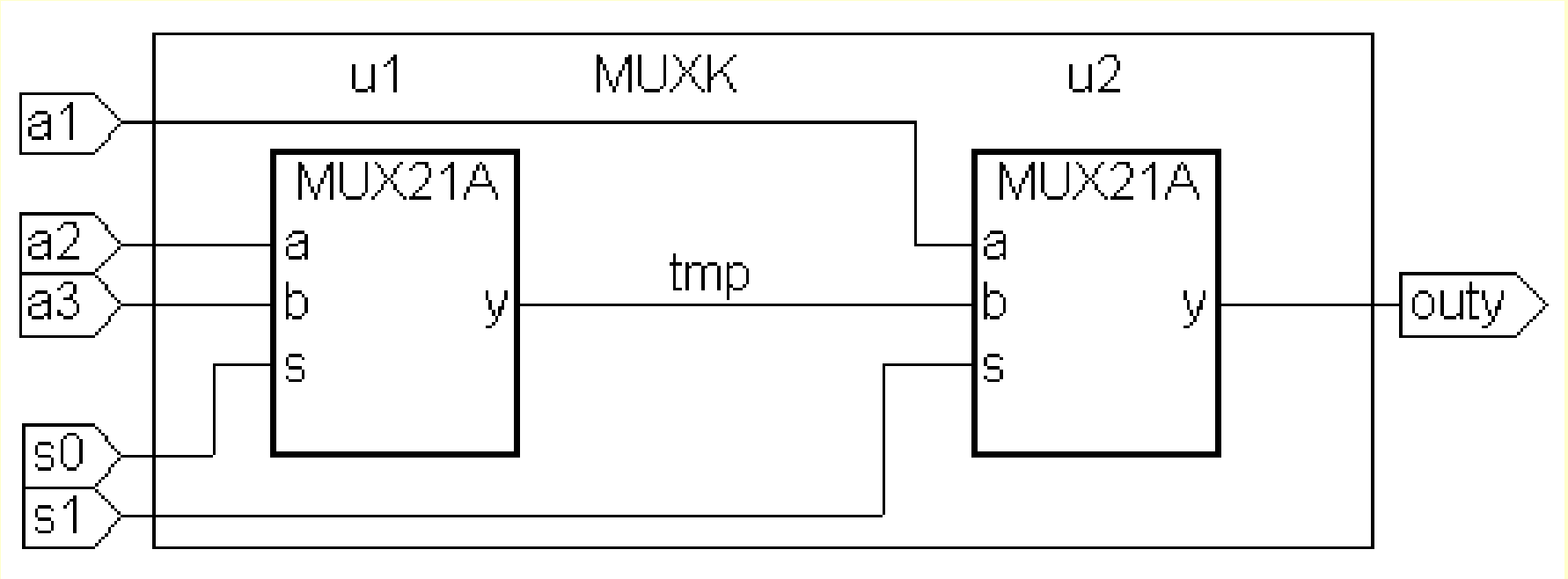


图4-18 双2选1多路选择器

习 题

4-4. 图4-19是一个含有上升沿触发的D触发器的时序电路，试写出此电路的VHDL设计文件。

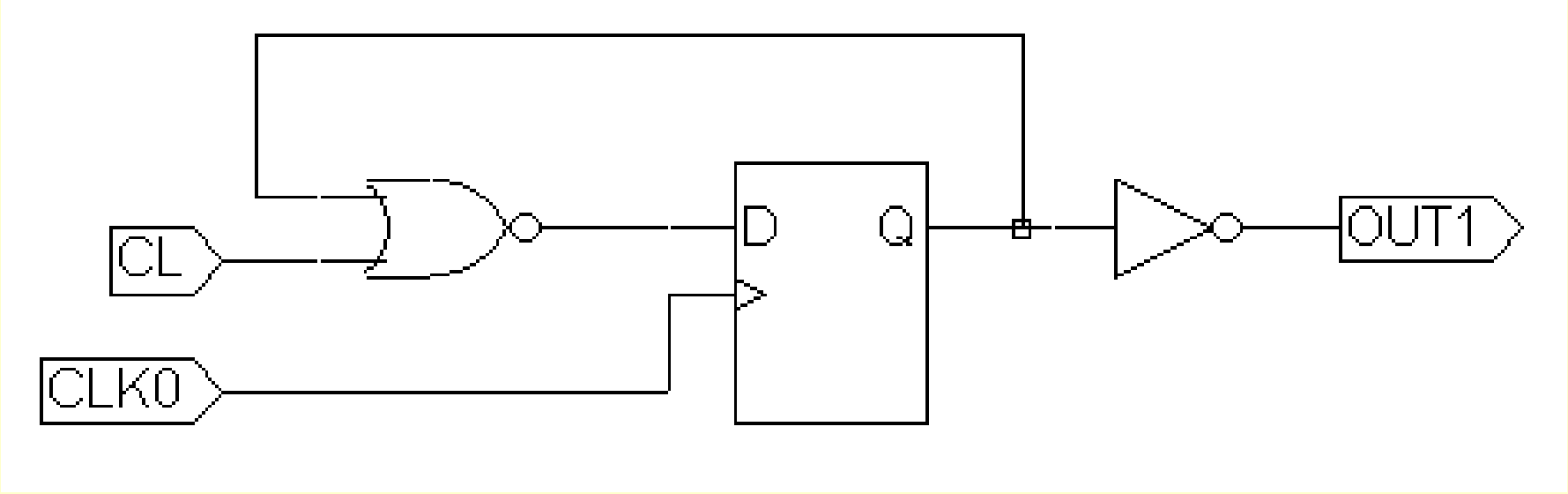


图4-19 时序电路图

习 题

4-5. 给出1位全减器的VHDL描述。要求：

(1) 首先设计1位半减器，然后用例化语句将它们连接起来，图4-20中h_suber是半减器，diff是输出差，s_out是借位输出，sub_in是借位输入。

(2) 以1位全减器为基本硬件，构成串行借位的8位减法器，要求用例化语句来完成此项设计(减法运算是 x – y – sun_in = diffr)。

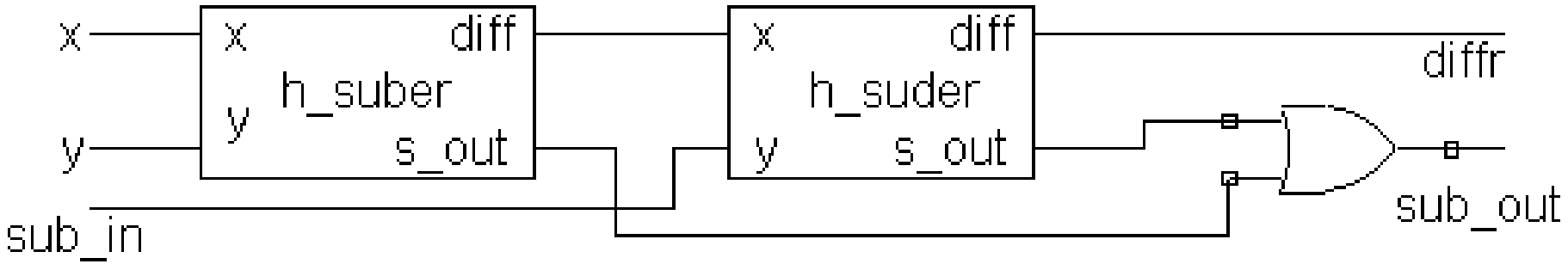


图4-19 时序电路图

习 题

4-6. 根据图4-21，写出顶层文件MX3256.VHD的VHDL设计文件。

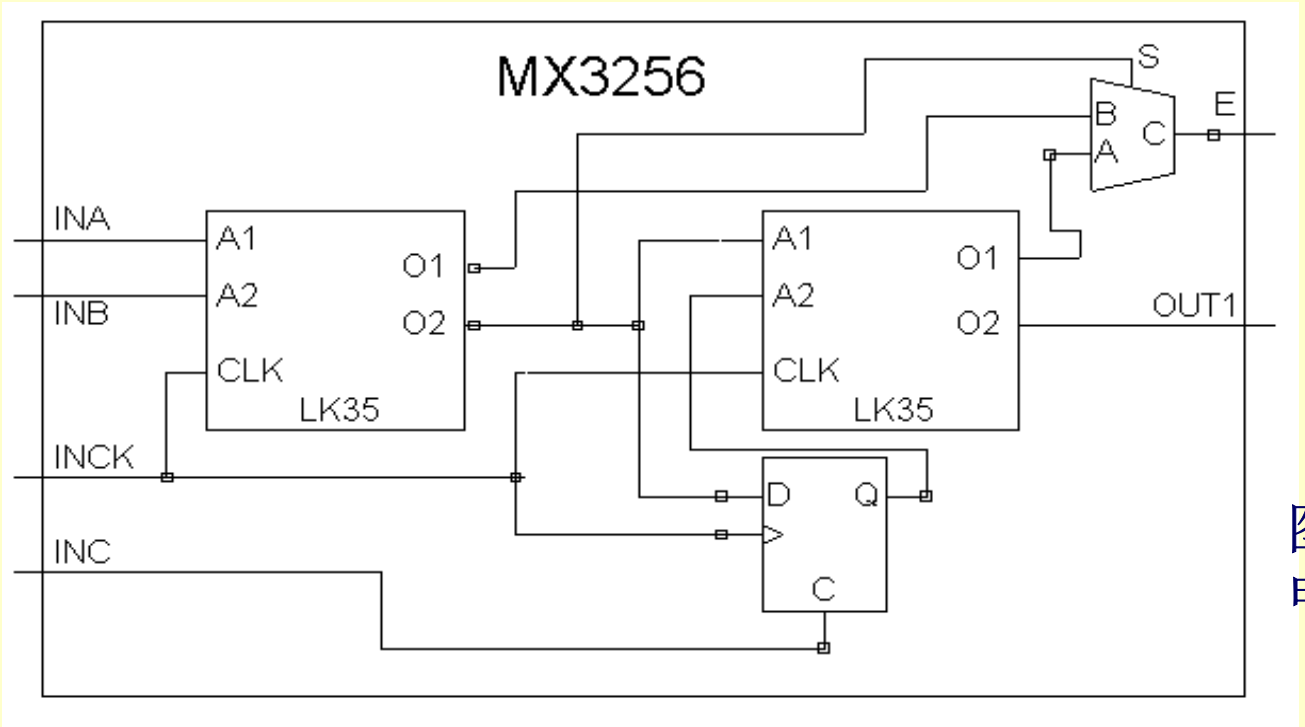


图4-21 题4-6 电路图

4-7. 设计含有异步清零和计数使能的16位二进制加减可控计数器。

EDA 技术实用教程

第 5 章

QuartusII 应用向导

5.1 基本设计流程

5.1.1 建立工作库文件夹和编辑设计文件

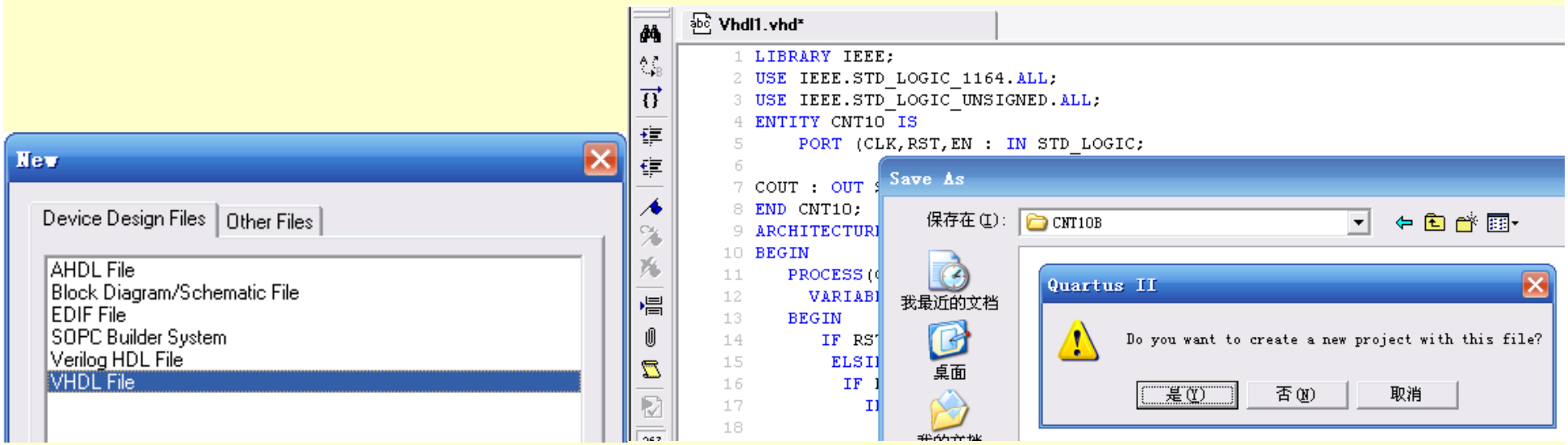


图5-1 选择编辑文件的语言类型，键入源程序并存盘

5.1 基本设计流程

5.1.2 创建工程

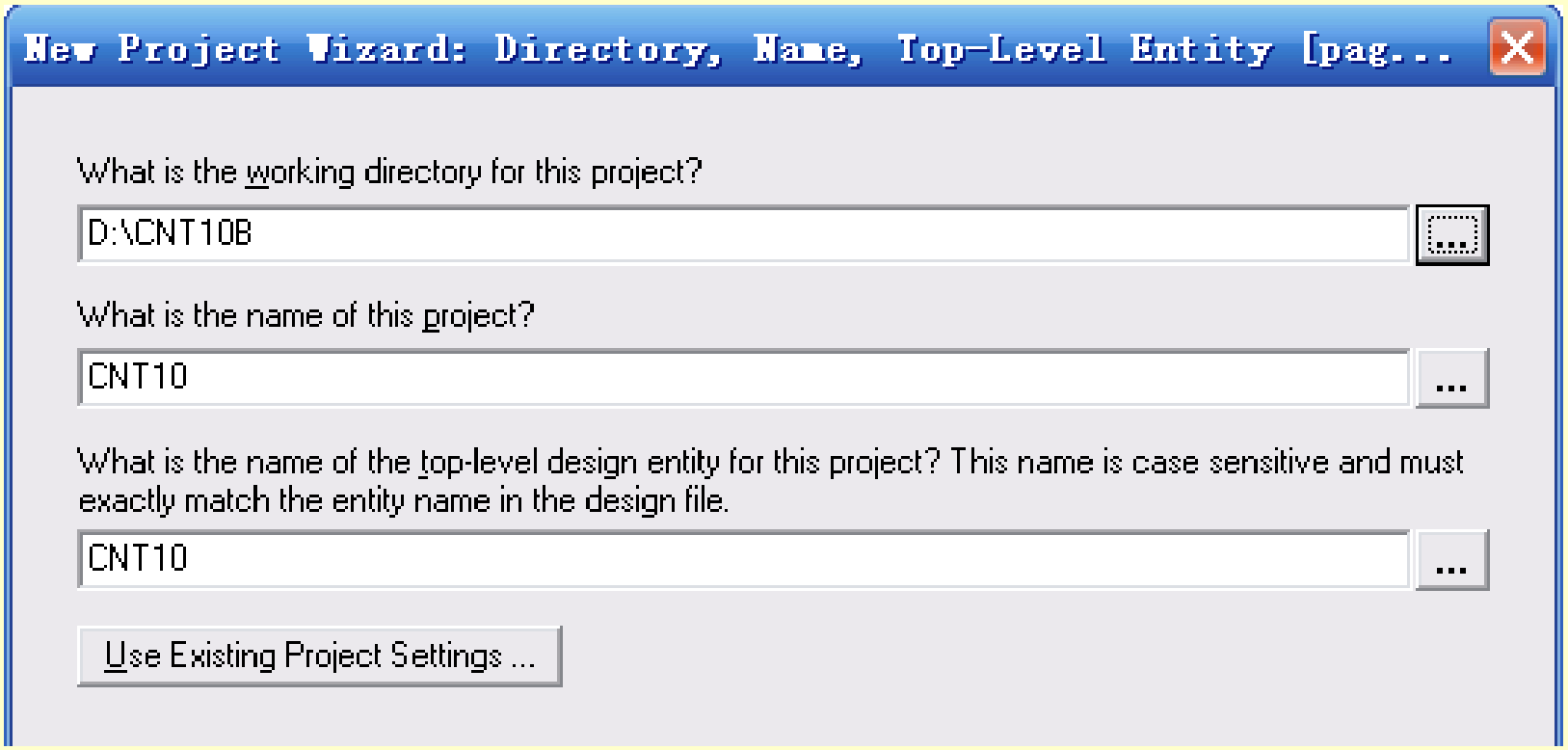


图5-2 利用“New Preject Wizard”创建工程cnt10

5.1 基本设计流程

5.1.2 创建工程

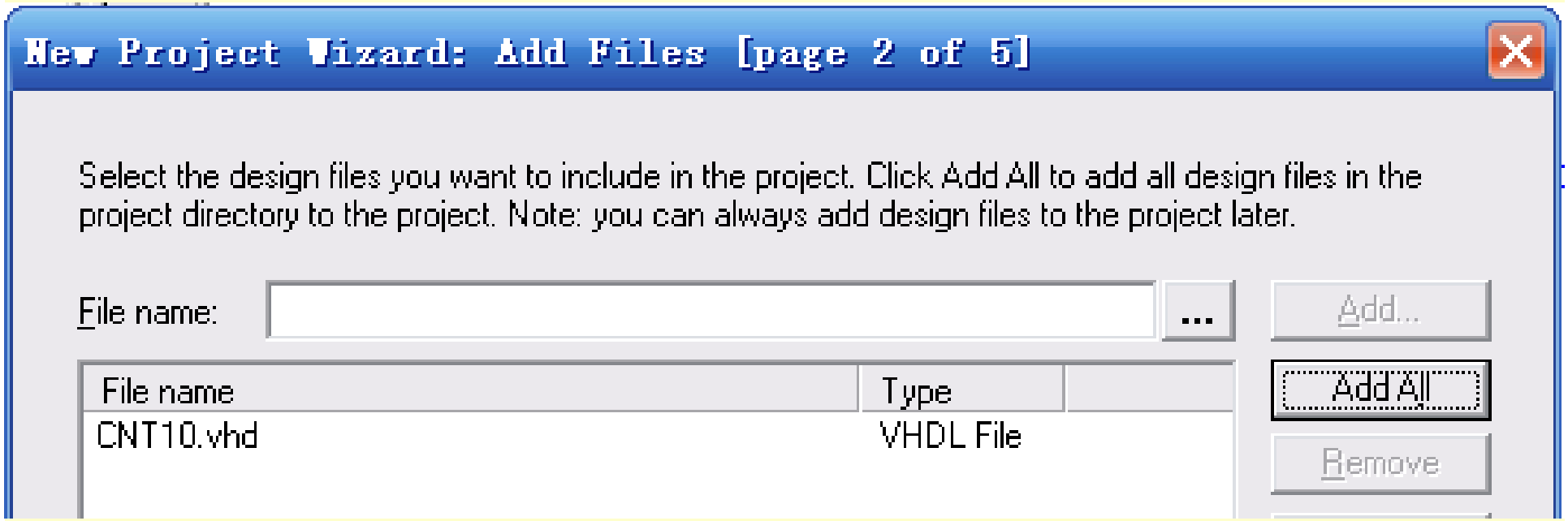


图5-3 将所有相关的文件都加入进此工程

5.1 基本设计流程

5.1.2 创建工程

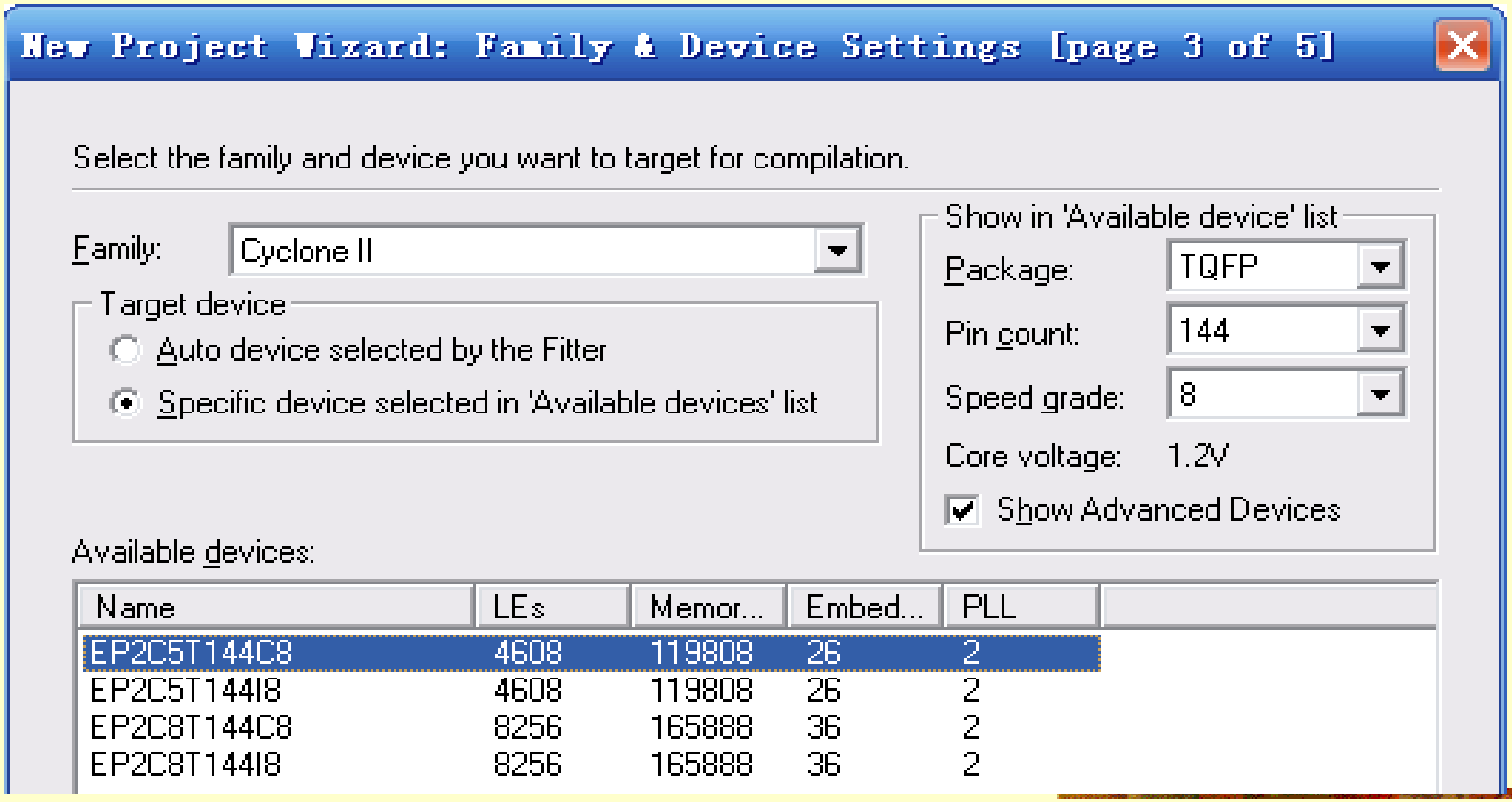


图5-4 选择目标器件EP2C5T144C8

5.1 基本设计流程

5.1.2 创建工程

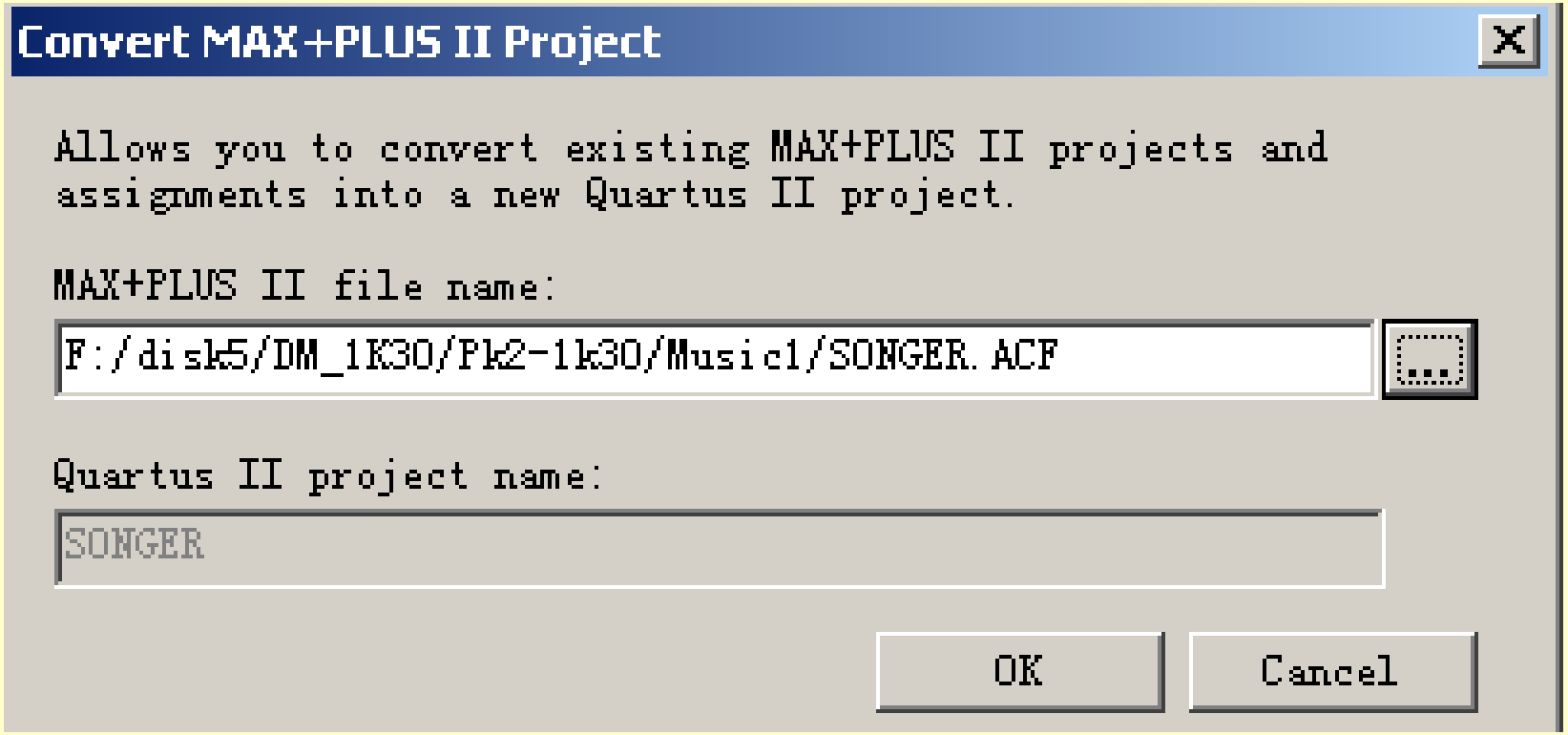


图5-5 将Max+plusII工程转换为QuartusII工程

5.1 基本设计流程

5.1.3 编译前设置

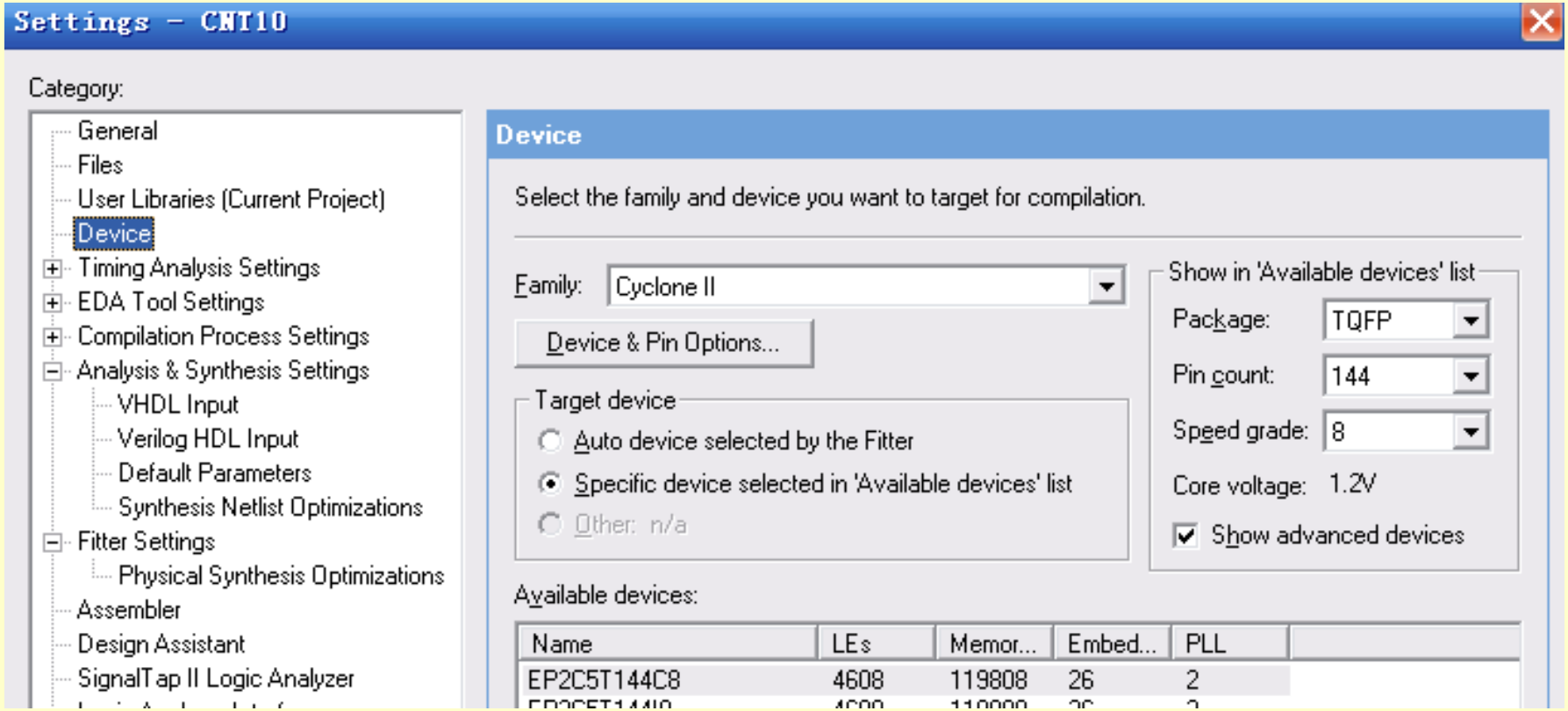


图5-6 选择目标器件EP2C5T144C8

5.1 基本设计流程

5.1.3 编译前设置

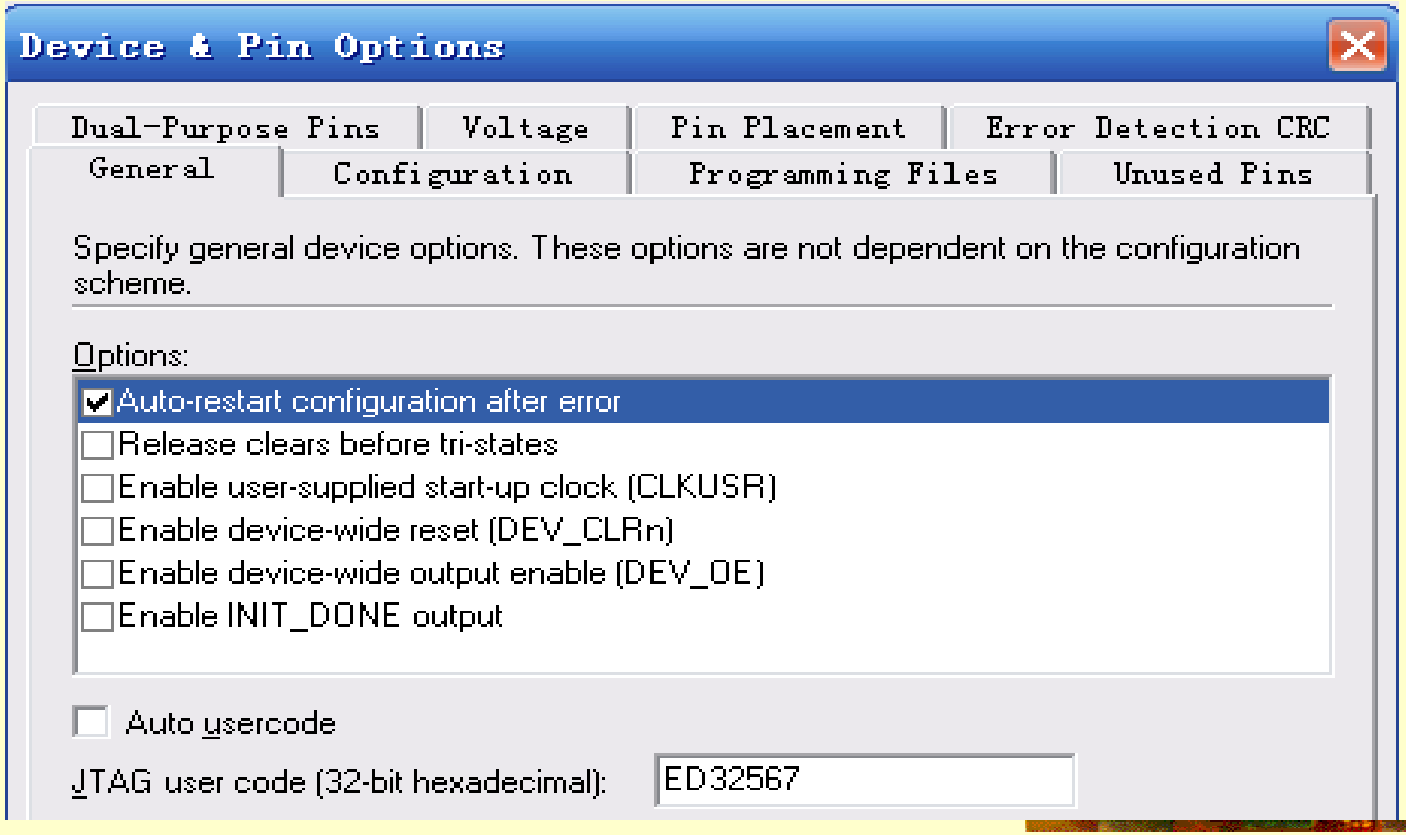


图5-7选择配置器件的工作方式

5.1 基本设计流程

5.1.3 编译前设置

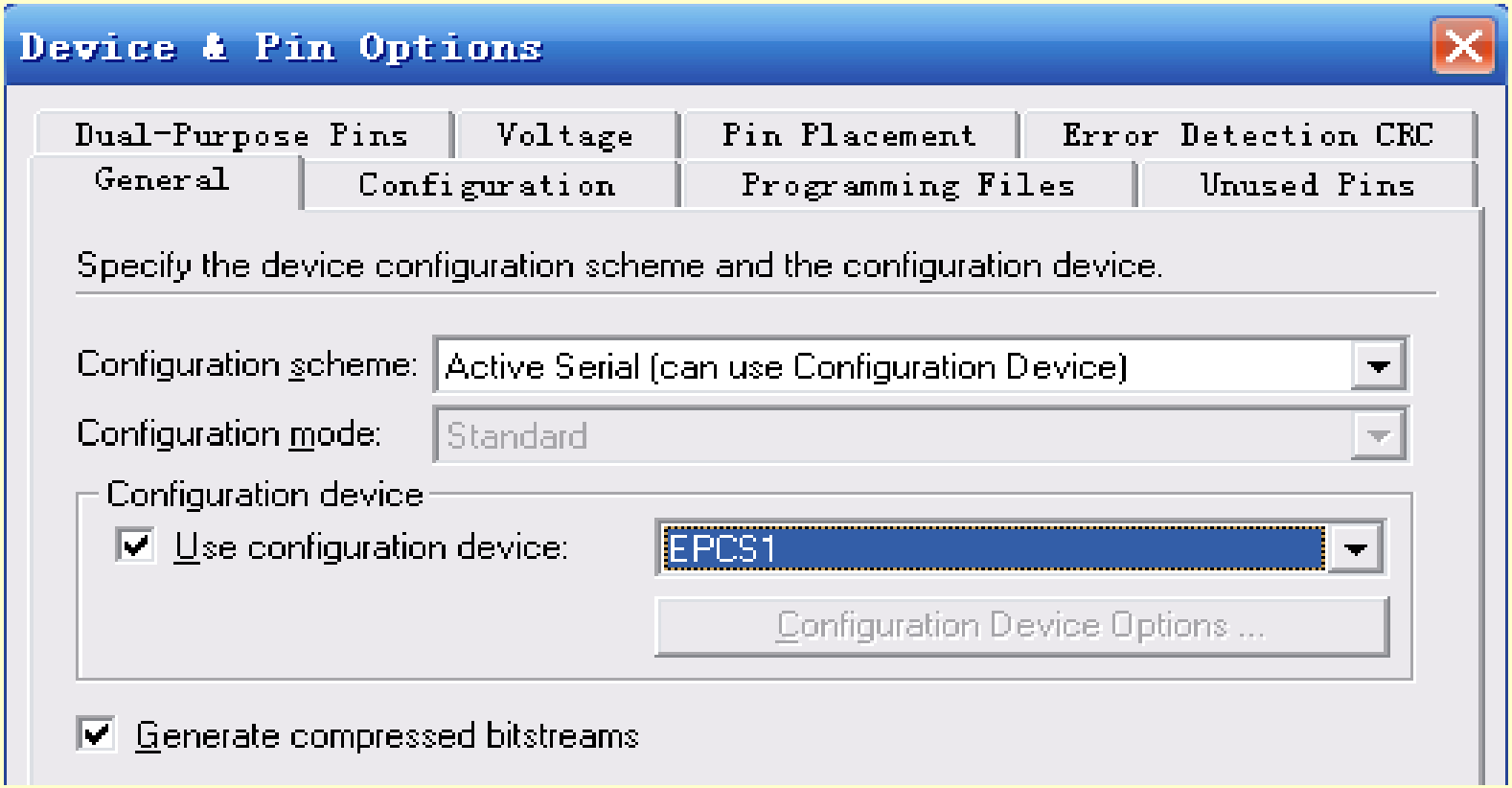


图5-8 选择配置器件和编程方式

5.1.4 全程编译

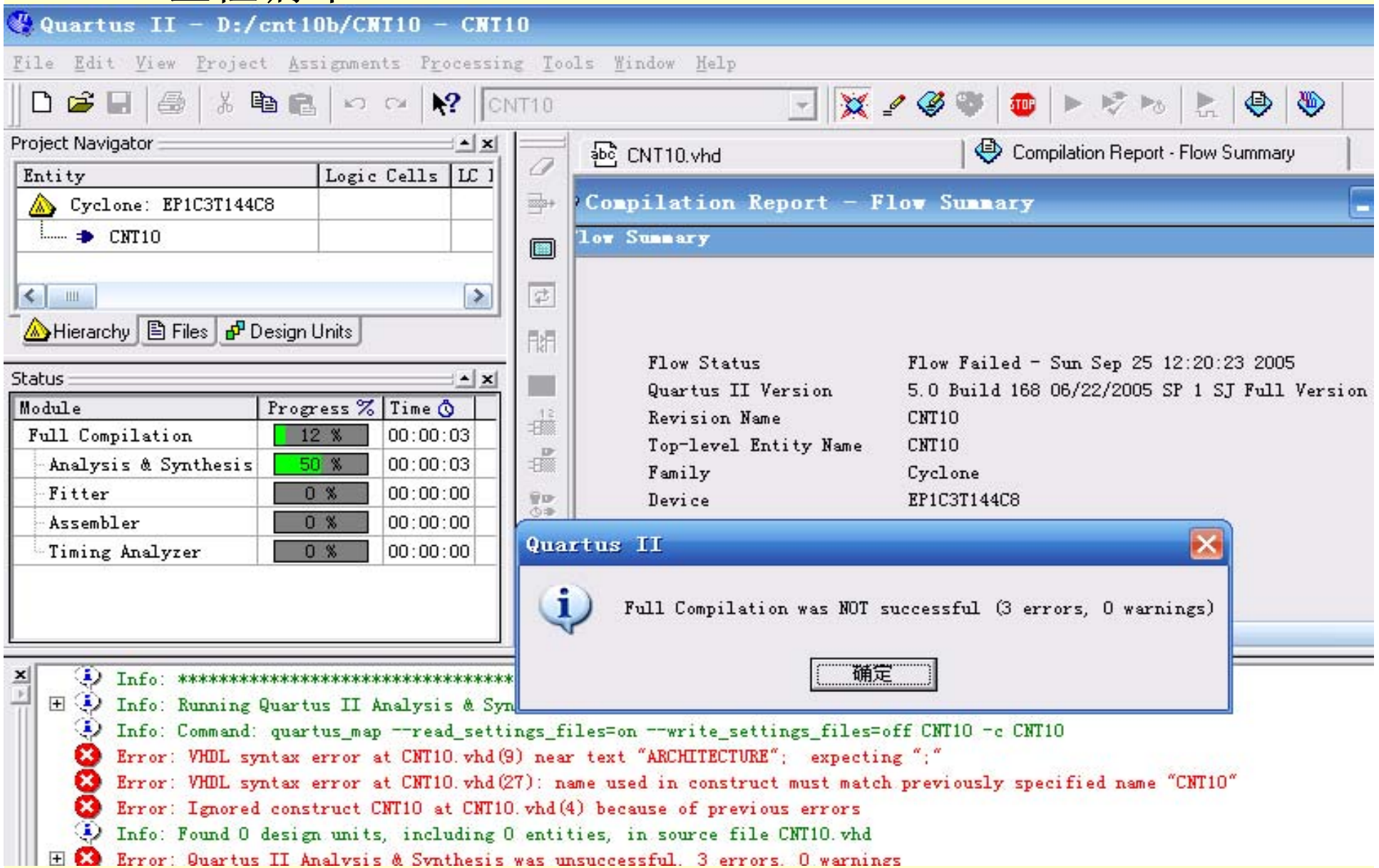


图5-9 全程编译后出现报错信息

KX 康芯科技

5.1 基本设计流程

5.1.5 时序仿真

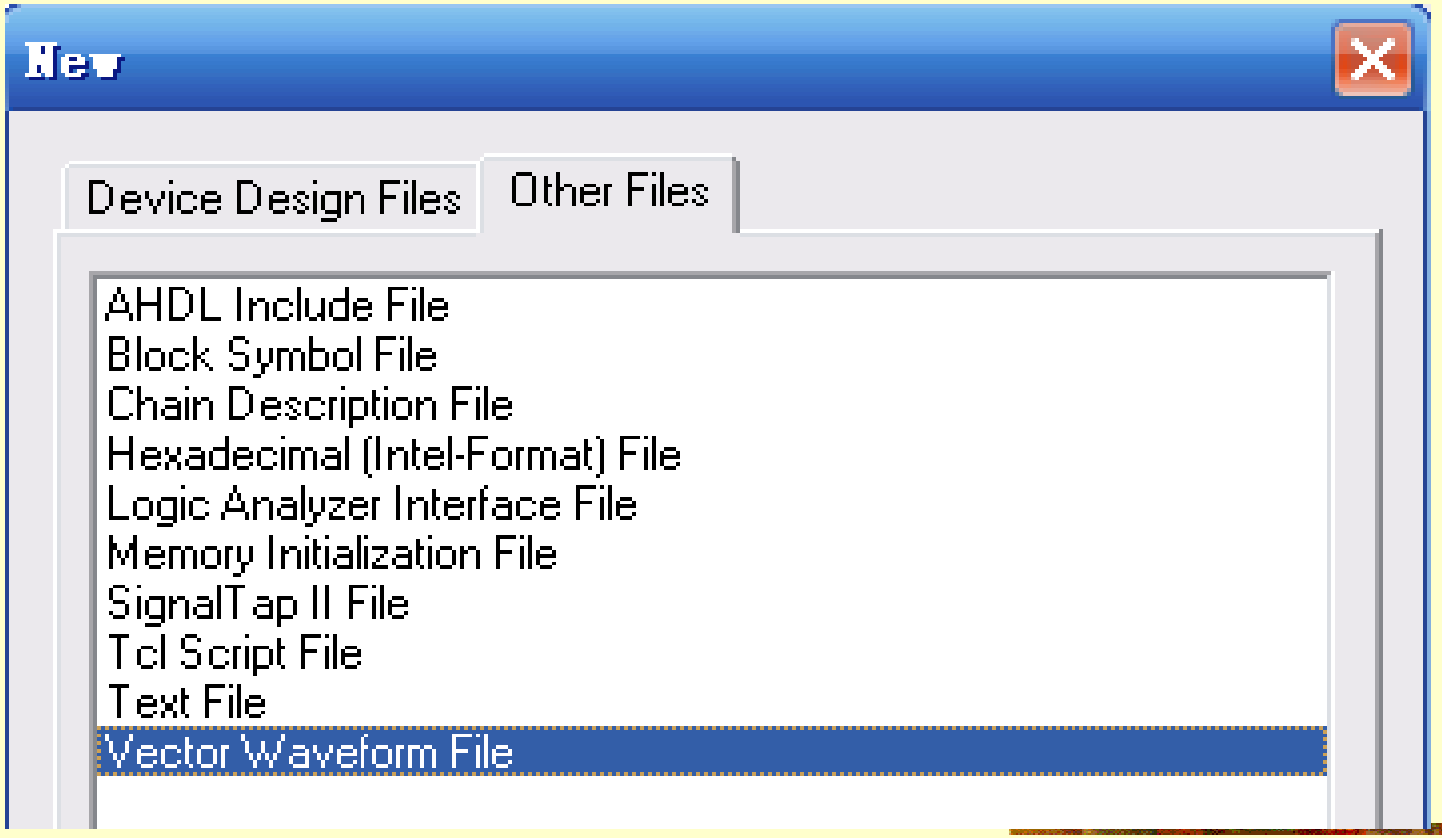


图5-10 选择编辑矢量波形文件

5.1 基本设计流程

5.1.5 时序仿真

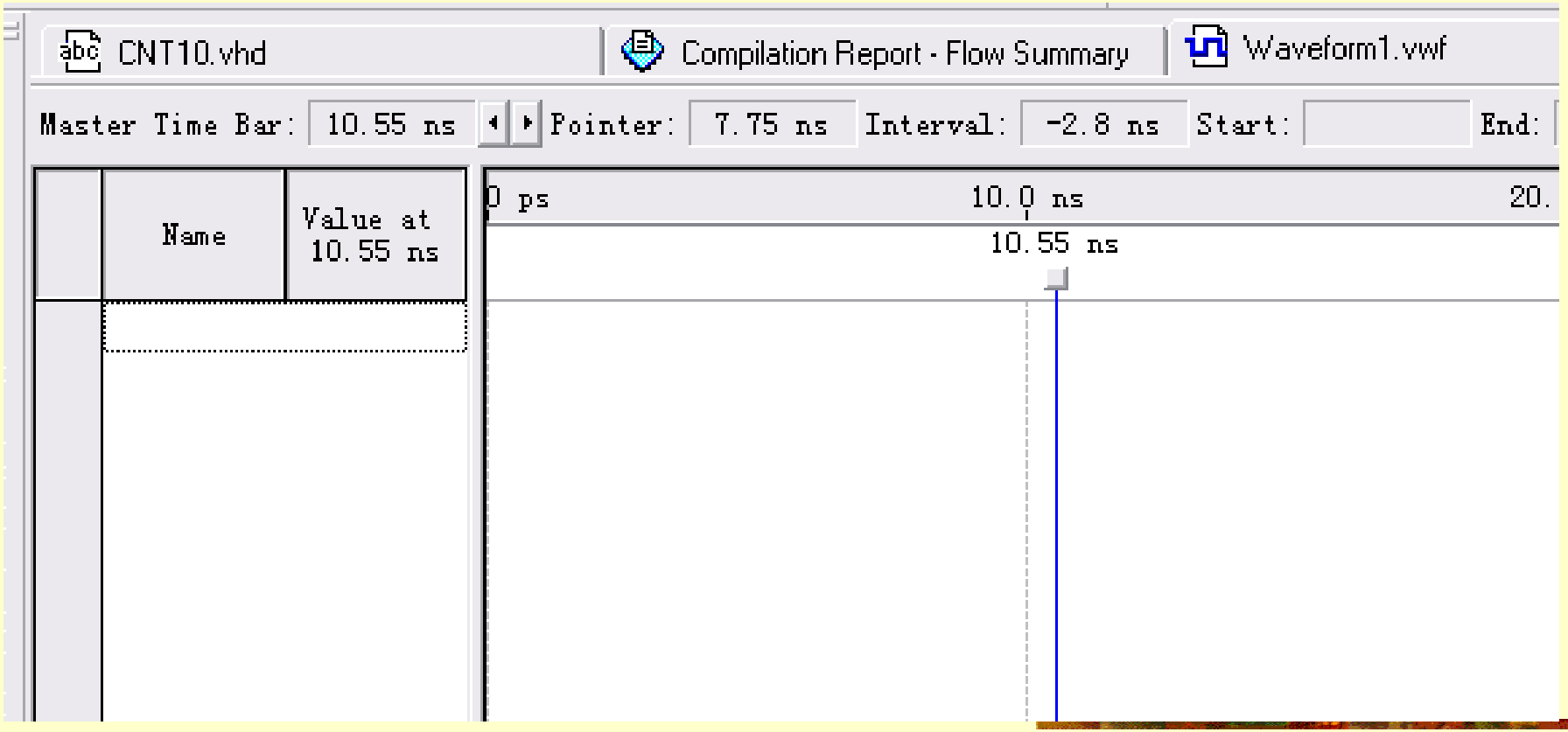


图5-11 波形编辑器

5.1 基本设计流程

5.1.5 时序仿真

图5-12 设置仿真时间长度

5.1 基本设计流程

5.1.5 时序仿真

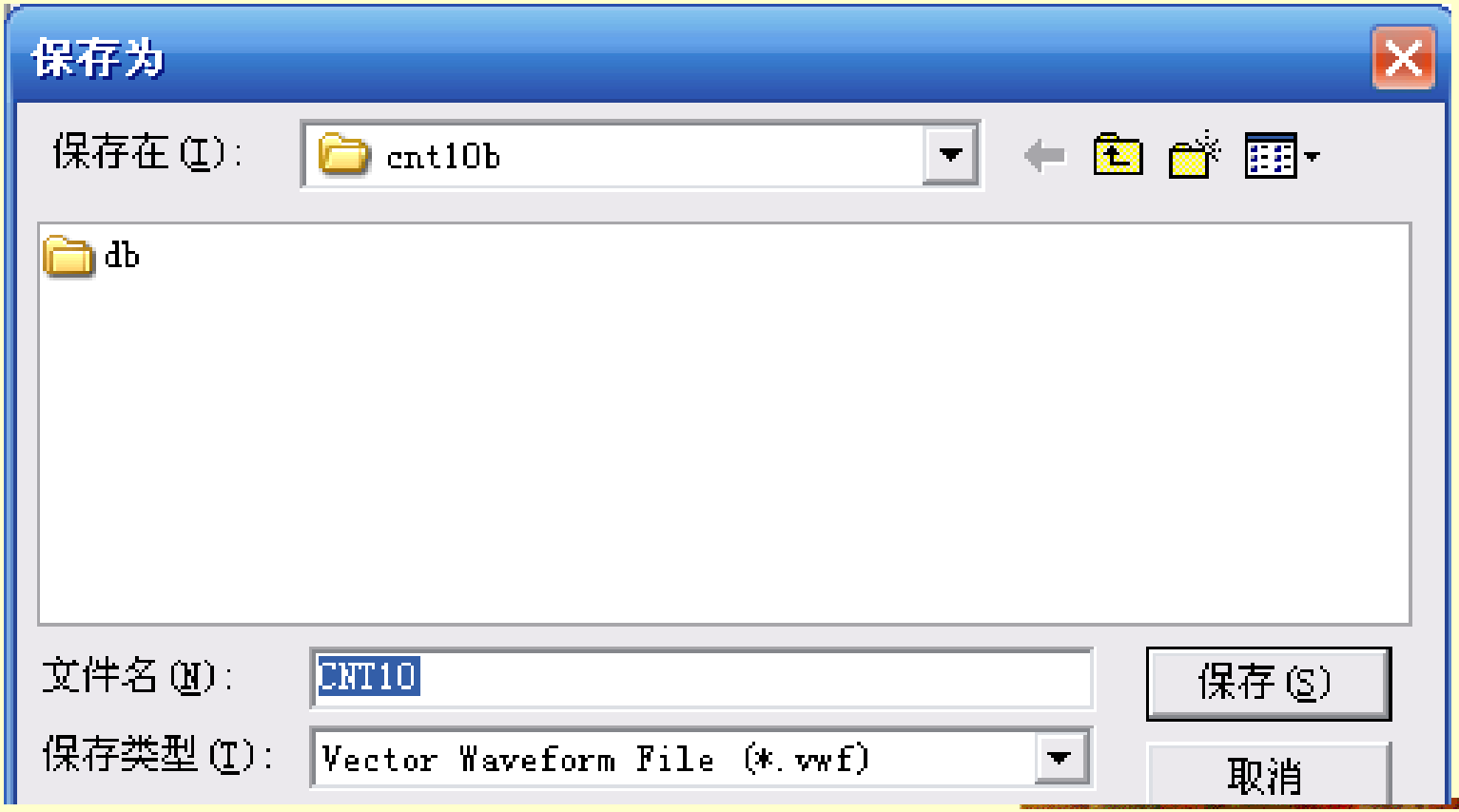


图5-13 vwf激励波形文件存盘

5.1.5 时序仿真

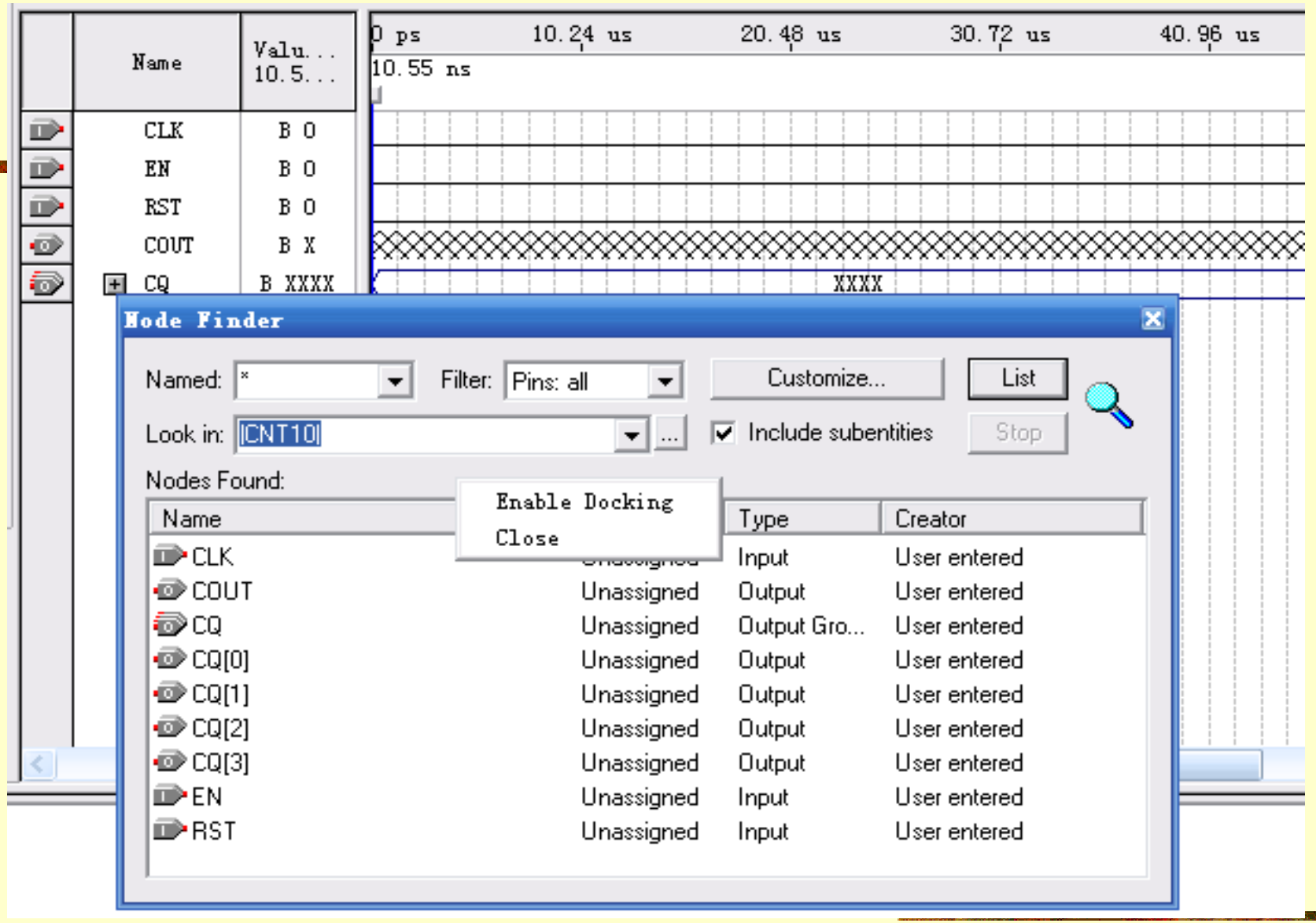


图5-14 向波形编辑器拖入信号节点

5.1 基本设计流程

5.1.5 时序仿真

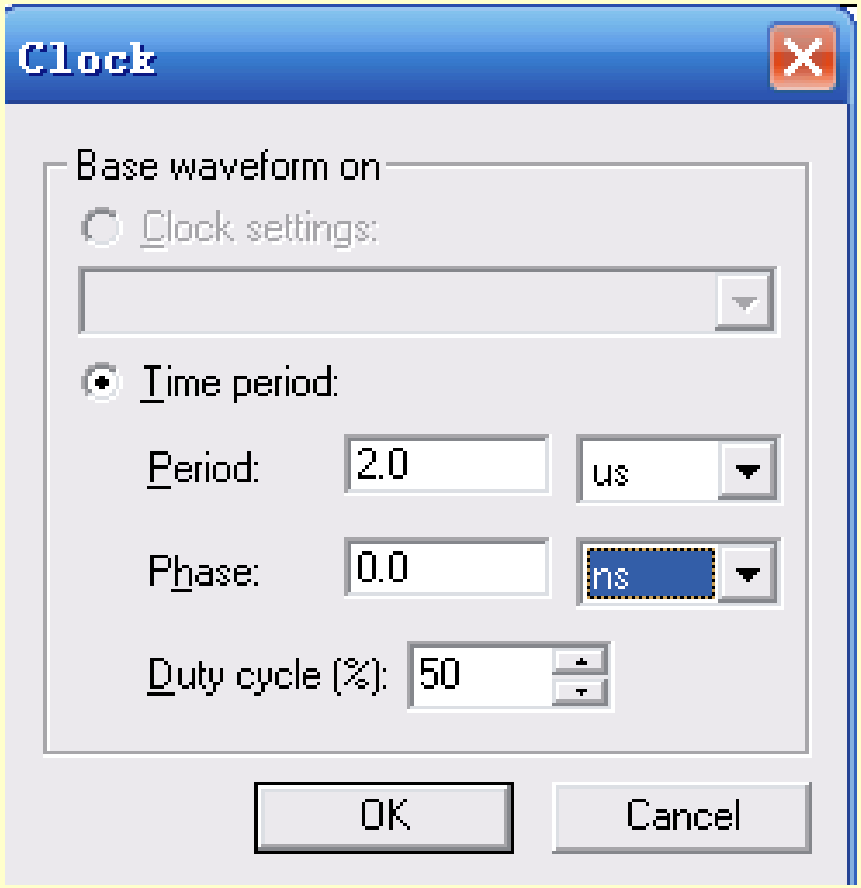


图5-15 设置时钟CLK的周期

5.1 基本设计流程

5.1.5 时序仿真

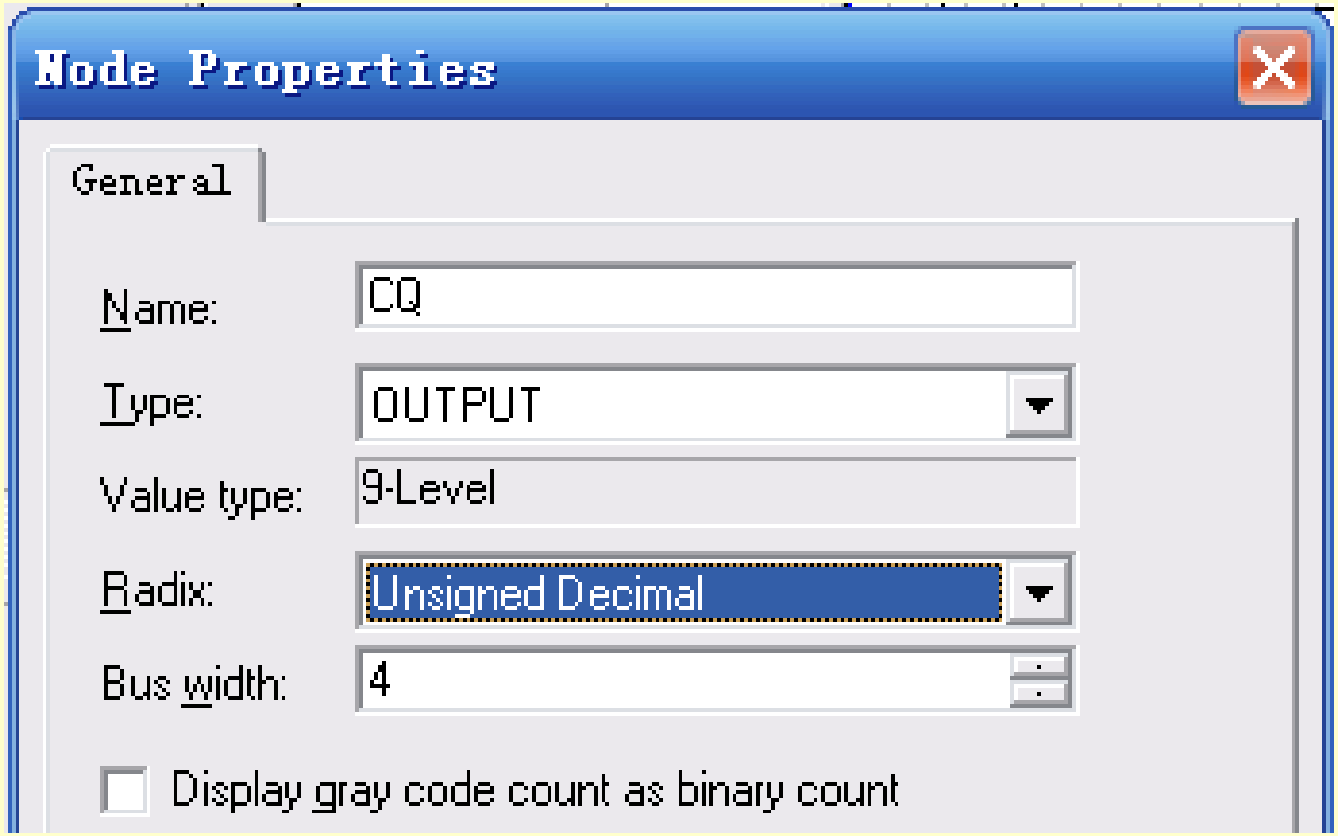


图5-16 选择总线数据格式

5.1 基本设计流程

5.1.5 时序仿真

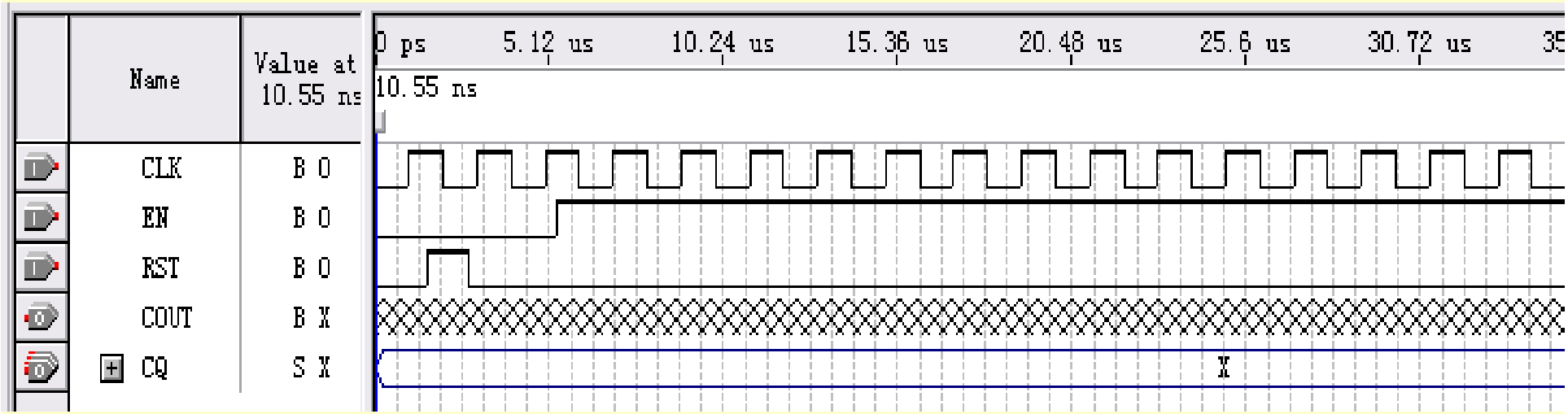


图5-17设置好的激励波形图

5.1 基本设计流程

5.1.5 时序仿真

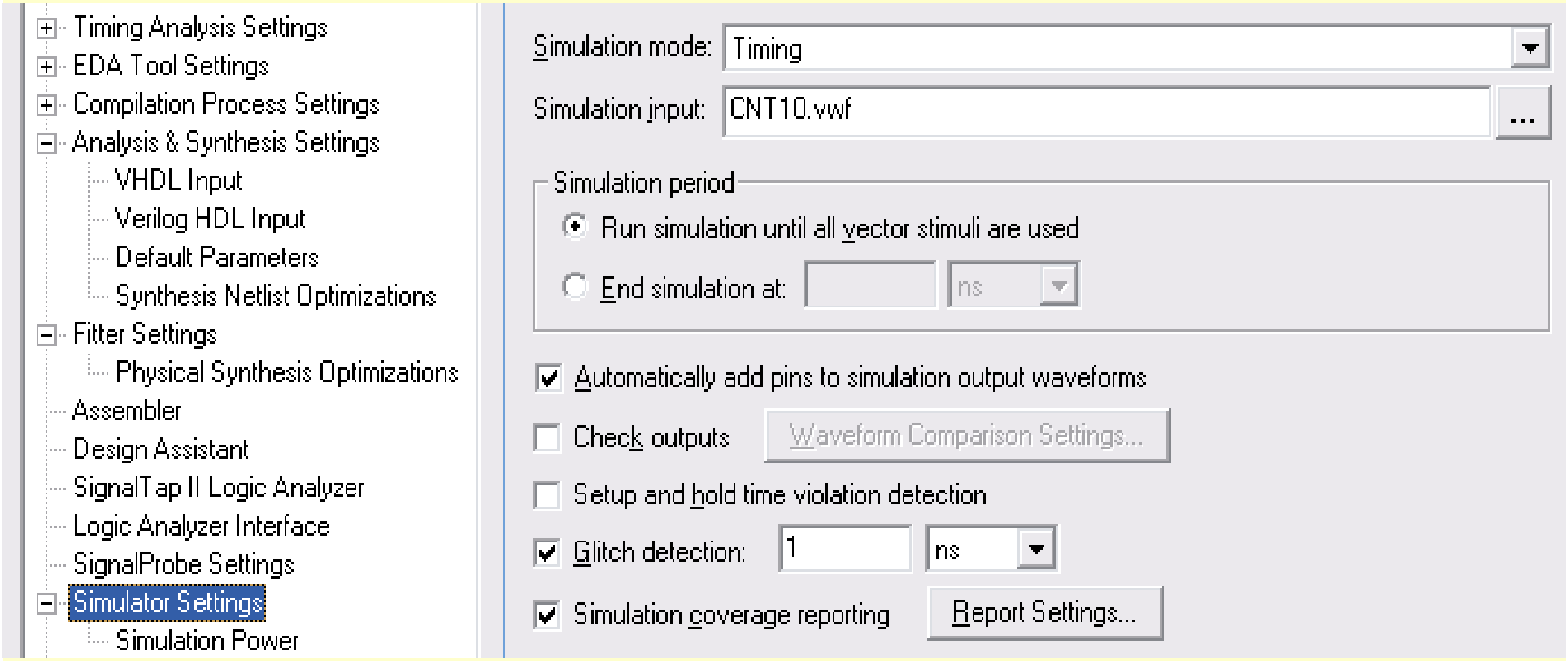


图5-18 选择仿真控制

5.1 基本设计流程

5.1.5 时序仿真

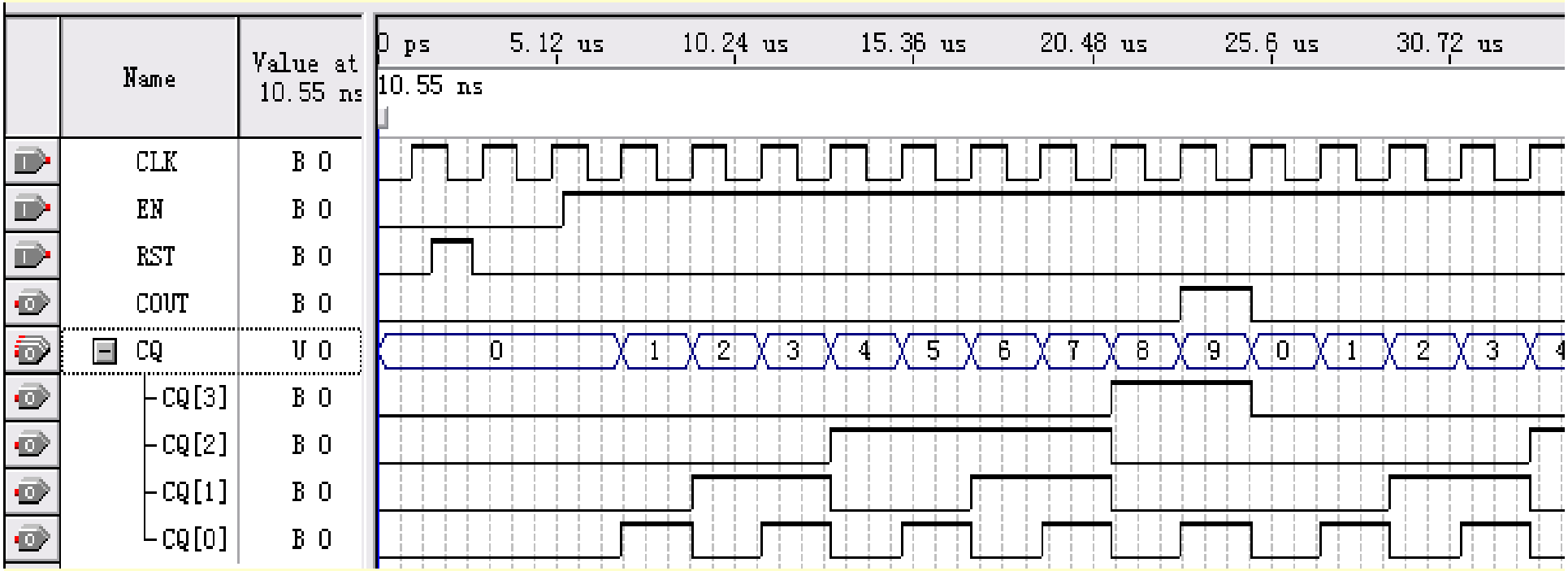


图5-19 仿真波形输出

5.1 基本设计流程

5.1.5 时序仿真

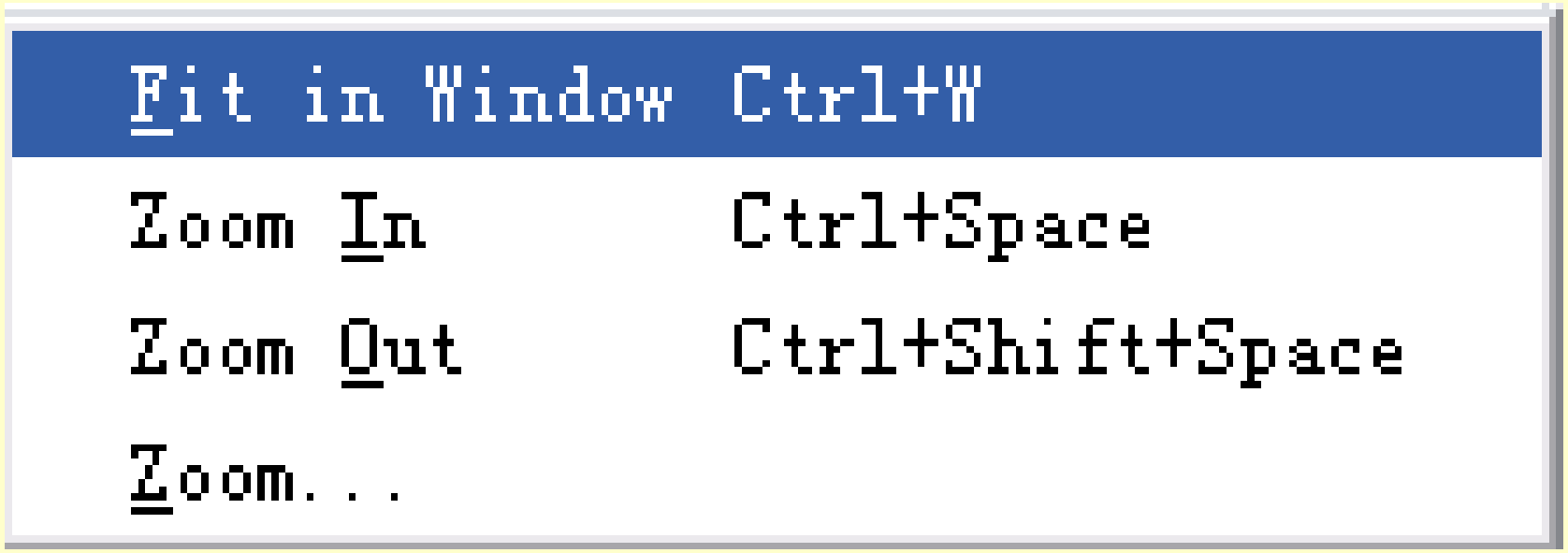


图5-20 选择全时域显示

5.1.6 应用RTL电路图观察器

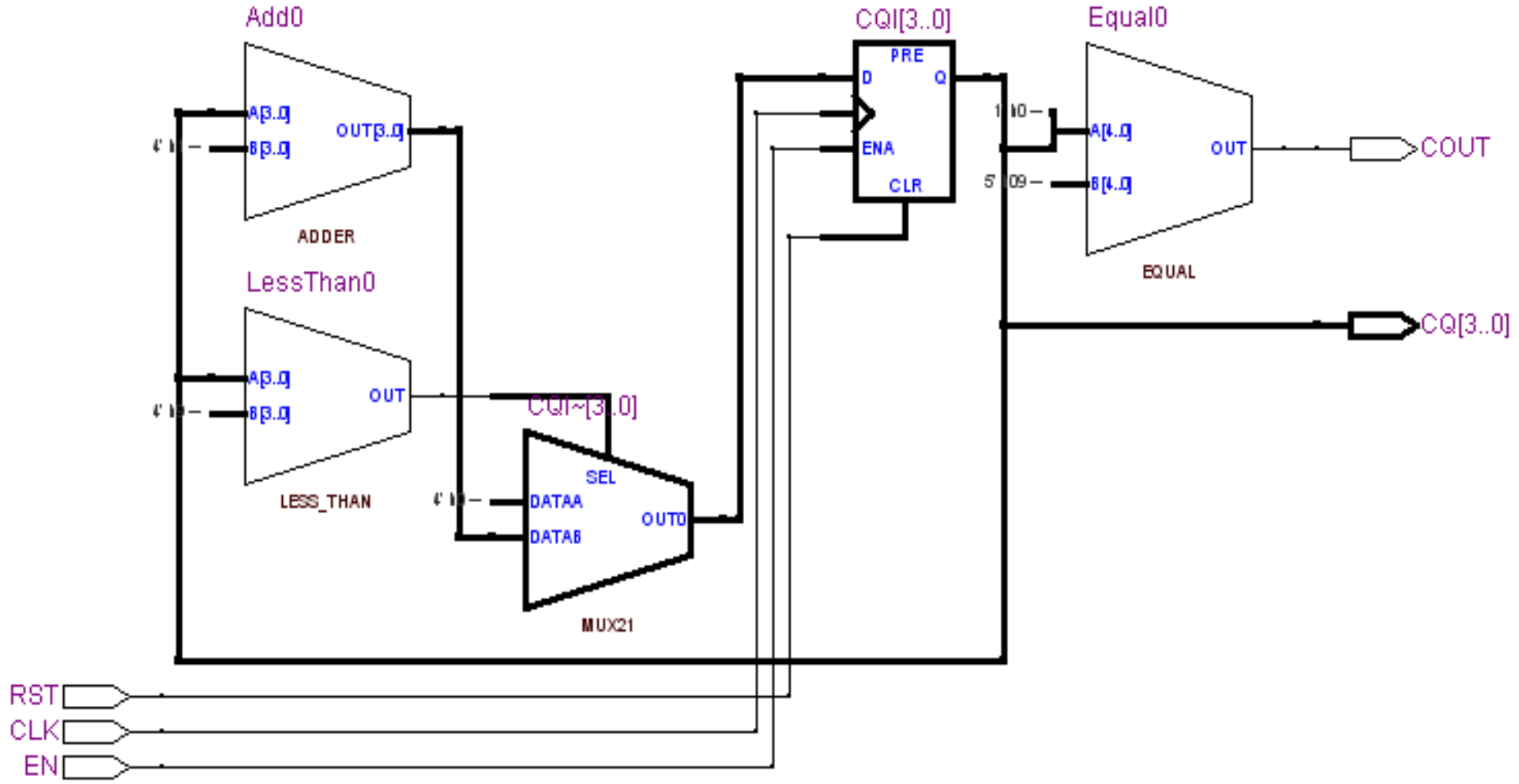


图5-21 cnt10工程的RTL电路图

5.2.1 引脚锁定

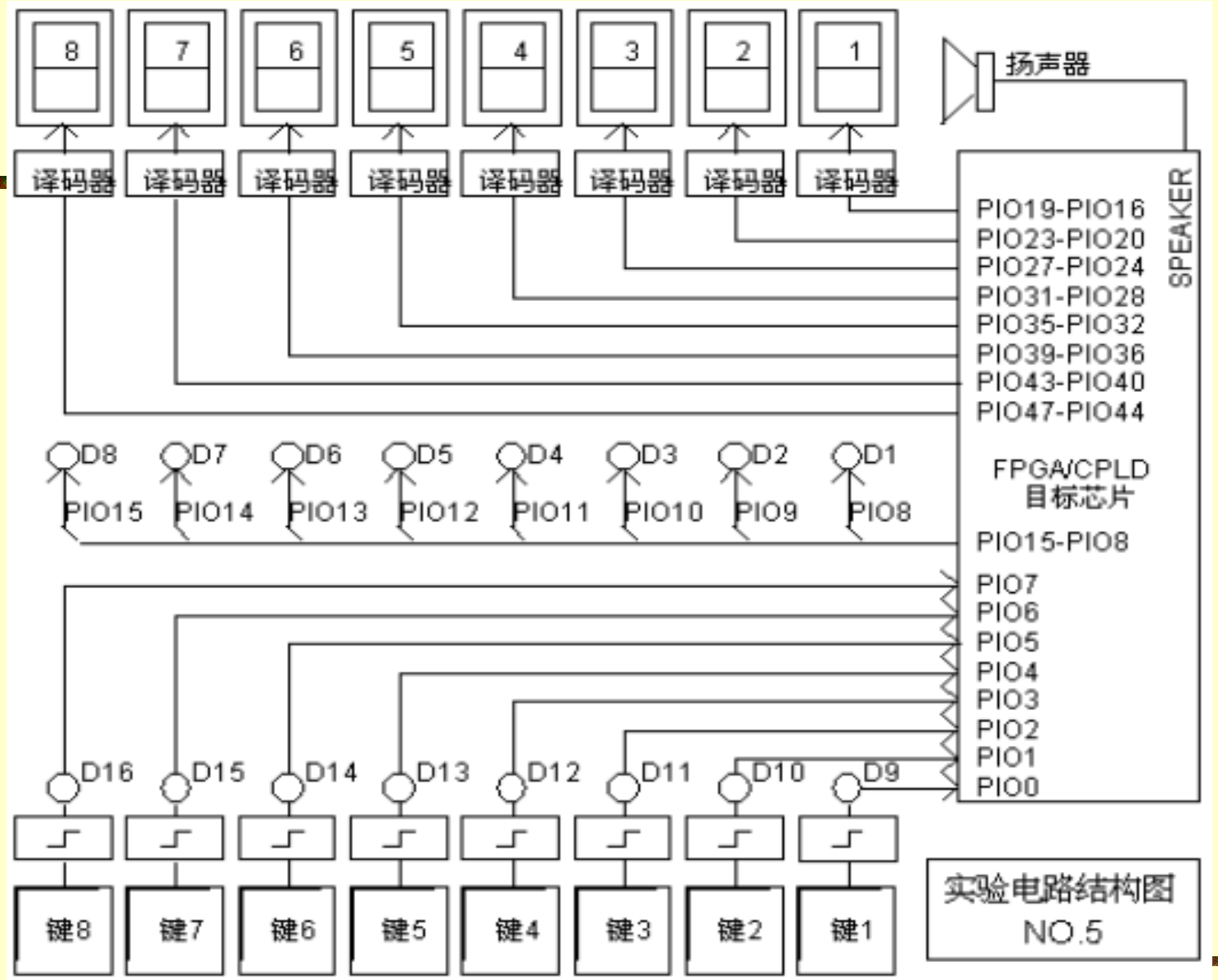


图5-22 GW48实验系统模式5实验电路图

KX 康芯科技

5.2 引脚设置和下载

5.2.1 引脚锁定

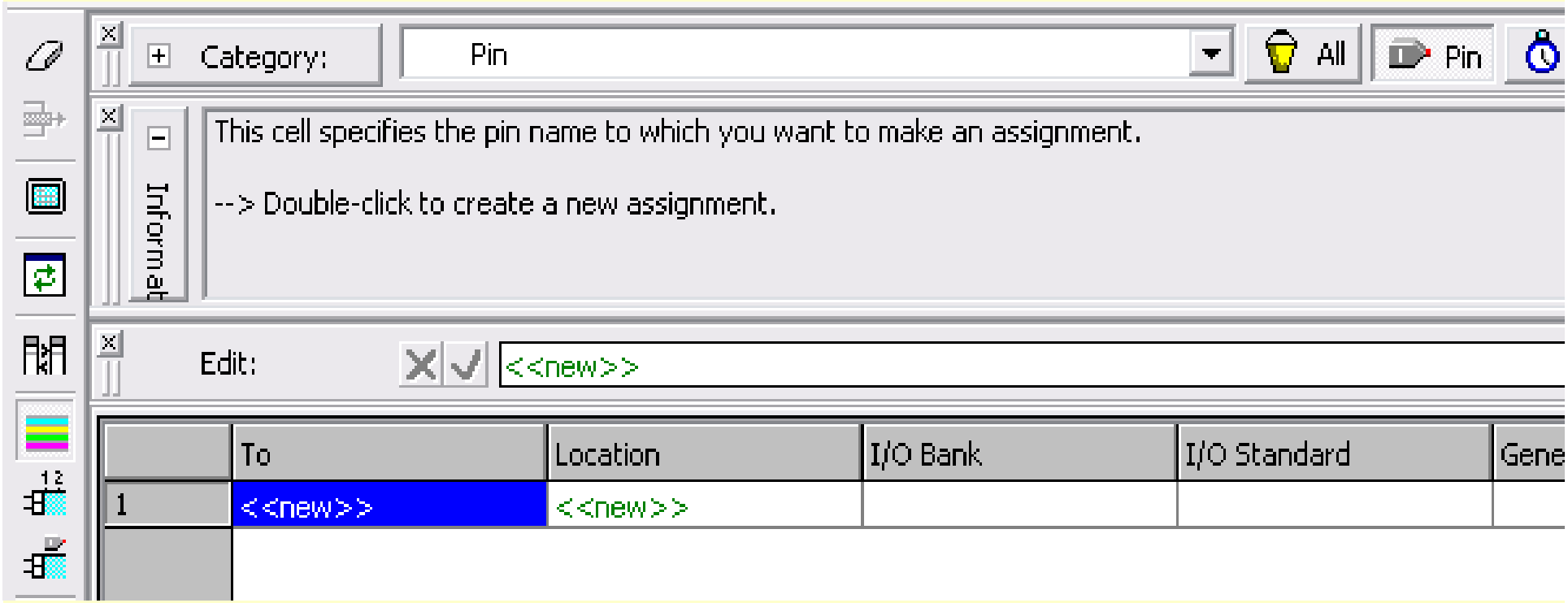


图5-23 Assignment Editor编辑器

5.2 引脚设置和下载

5.2.1 引脚锁定

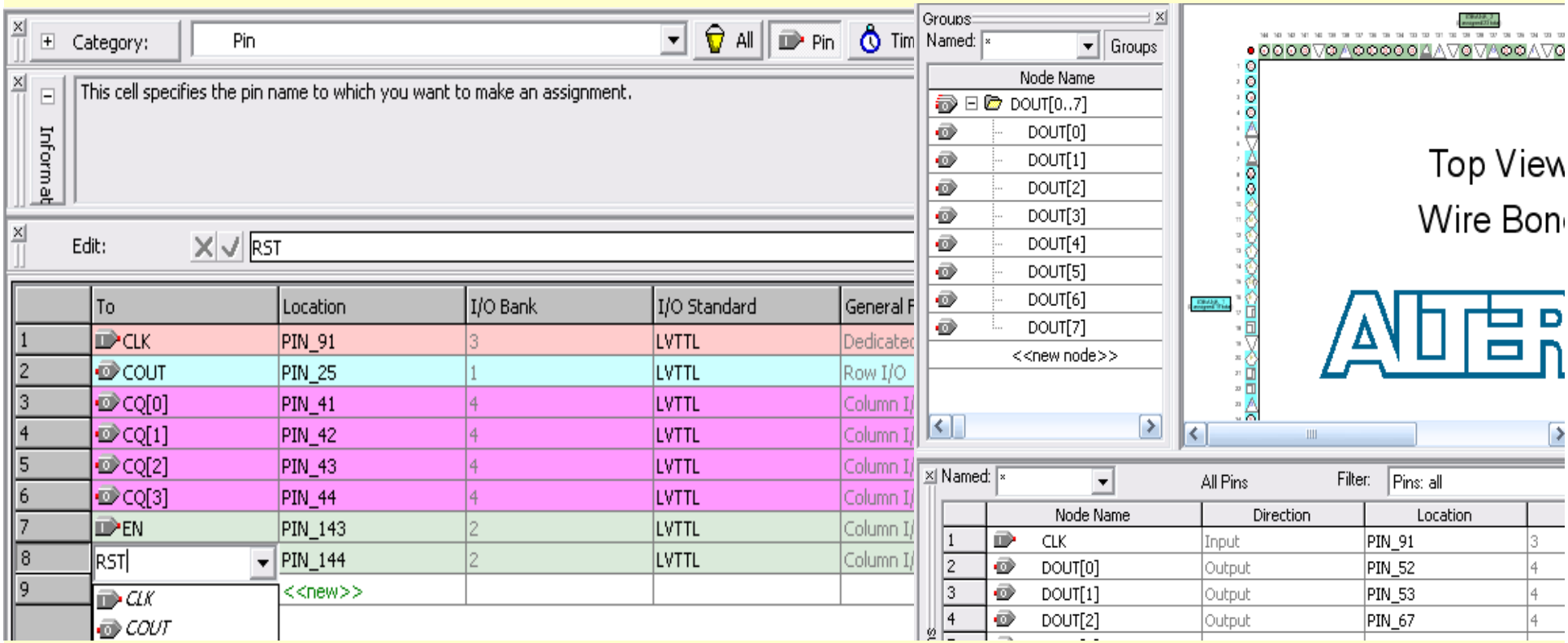


图5-24 两种引脚锁定对话框

5.2 引脚设置和下载

5.2.2 配置文件下载

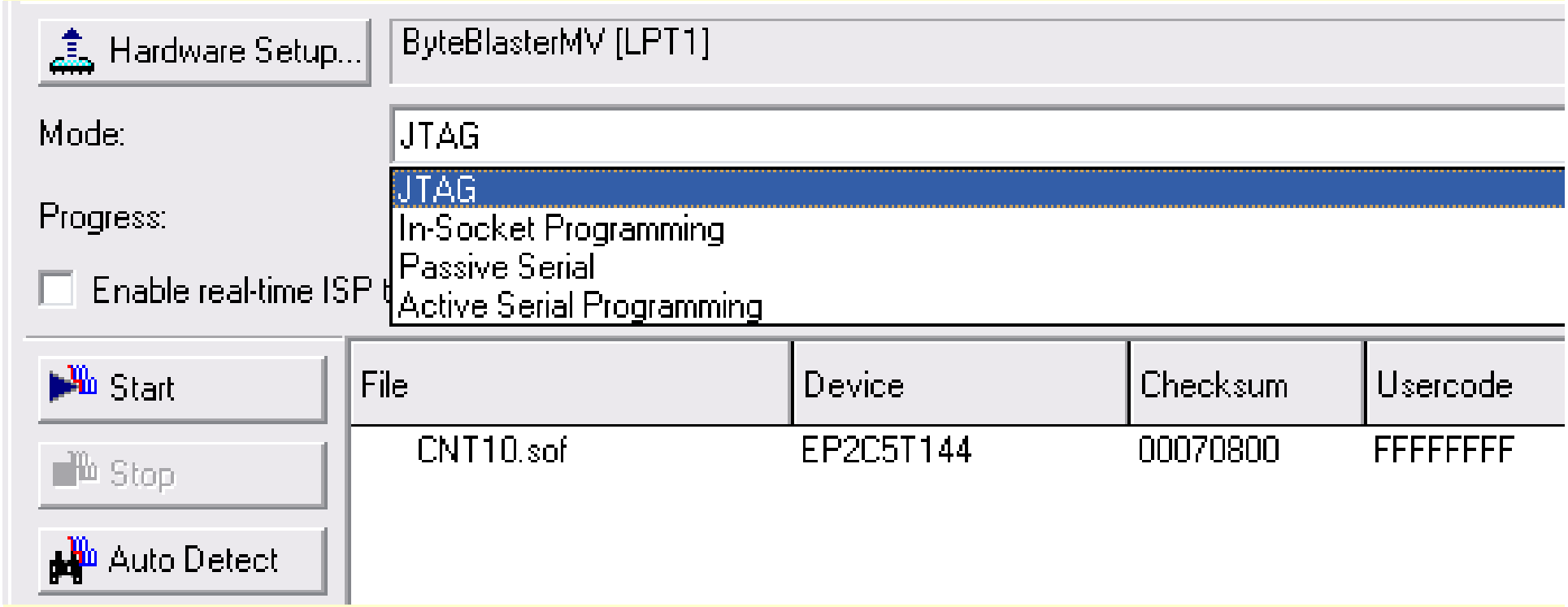


图5-25 选择编程下载文

5.2 引脚设置和下载

5.2.2 配置文件下载

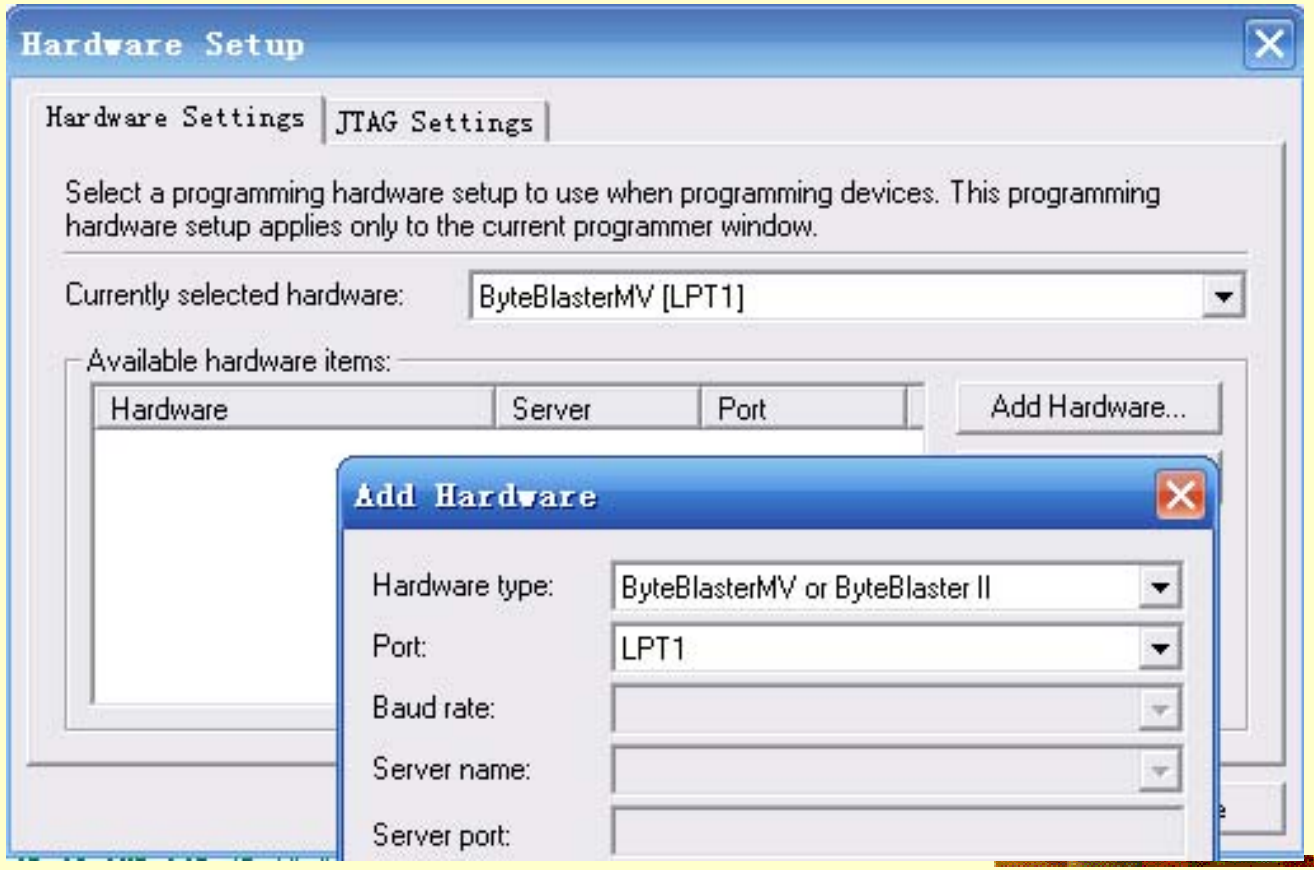


图5-26加入编程下载方式

5.2 引脚设置和下载

5.2.2 配置文件下载

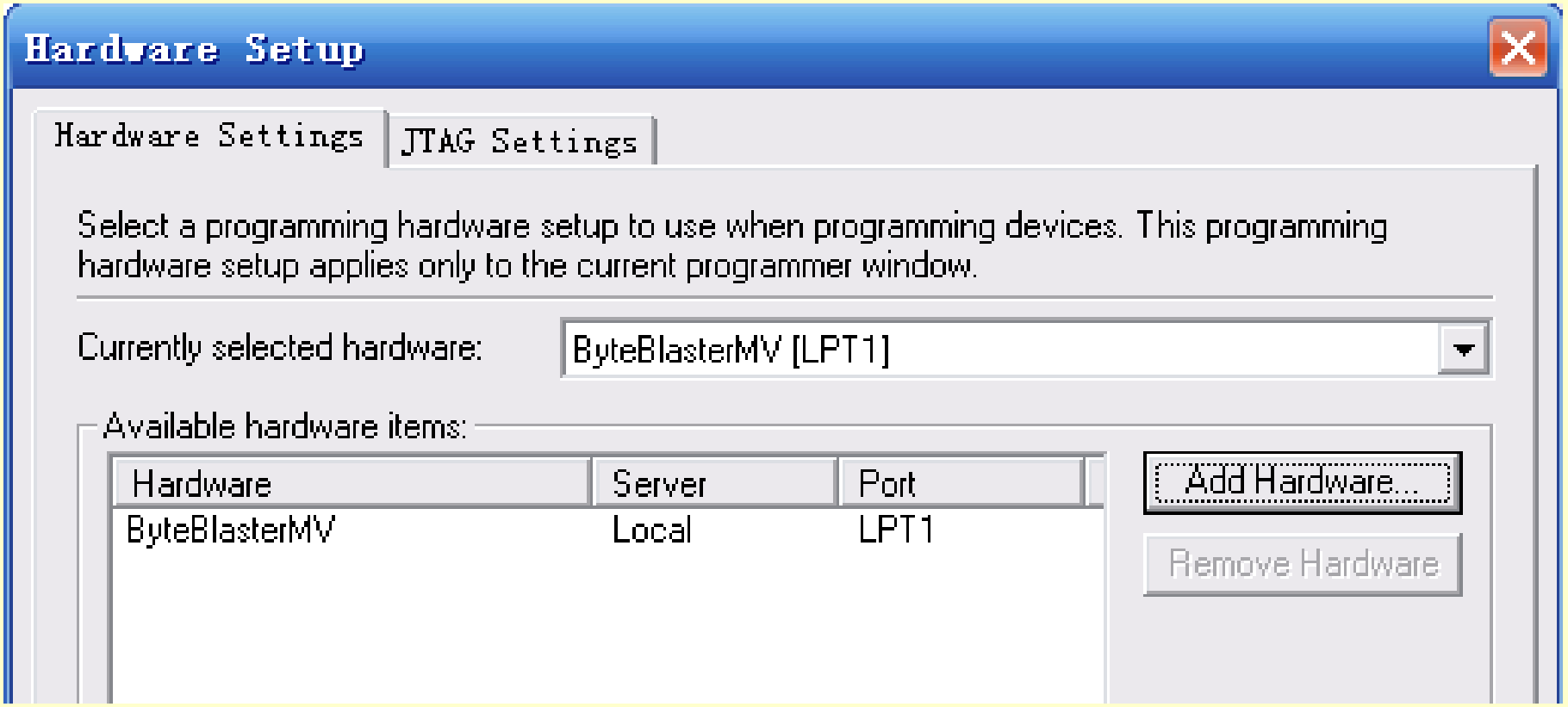


图5-27 双击选中的编程方式名

5.2 引脚设置和下载

5.2.2 配置文件下载

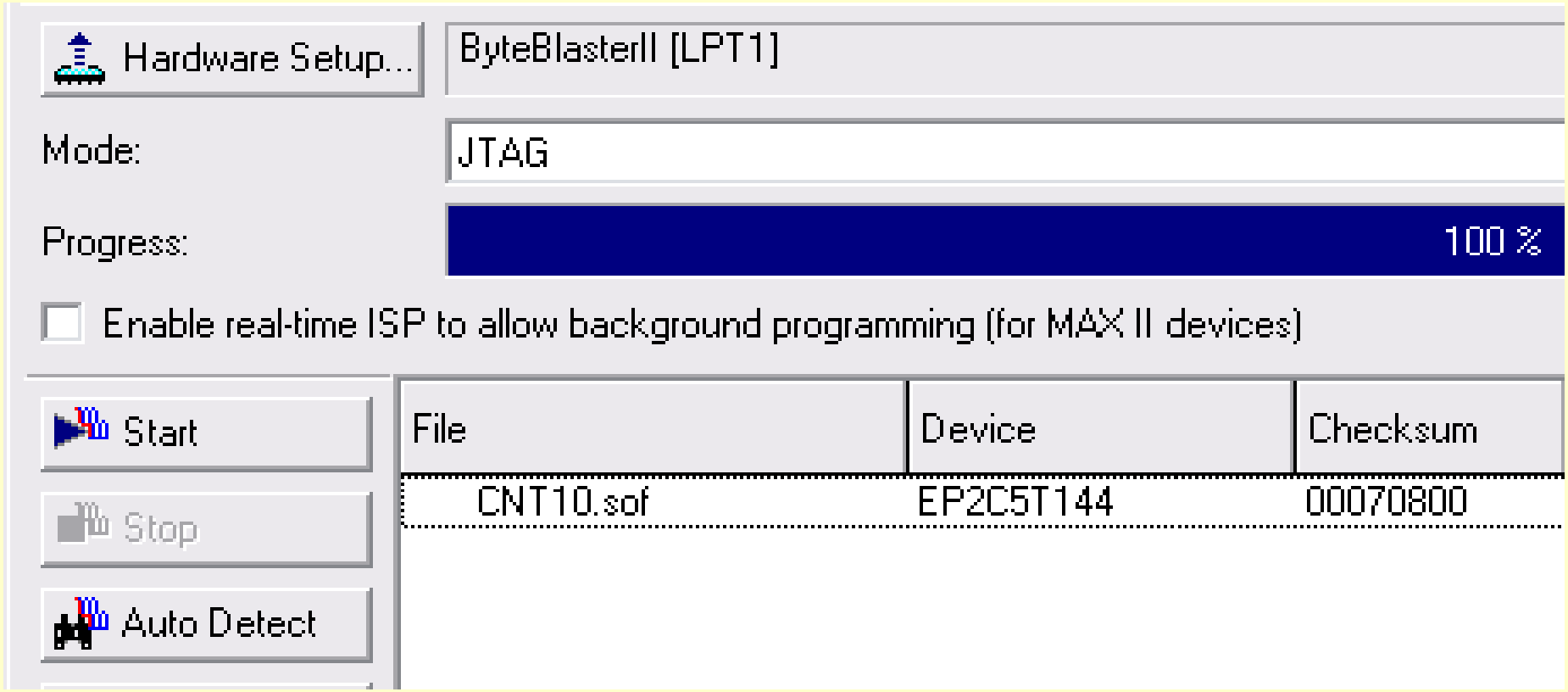


图5-28 ByteBlasterII编程下载窗

5.2 引脚设置和下载

5.2.3 AS模式编程配置器件

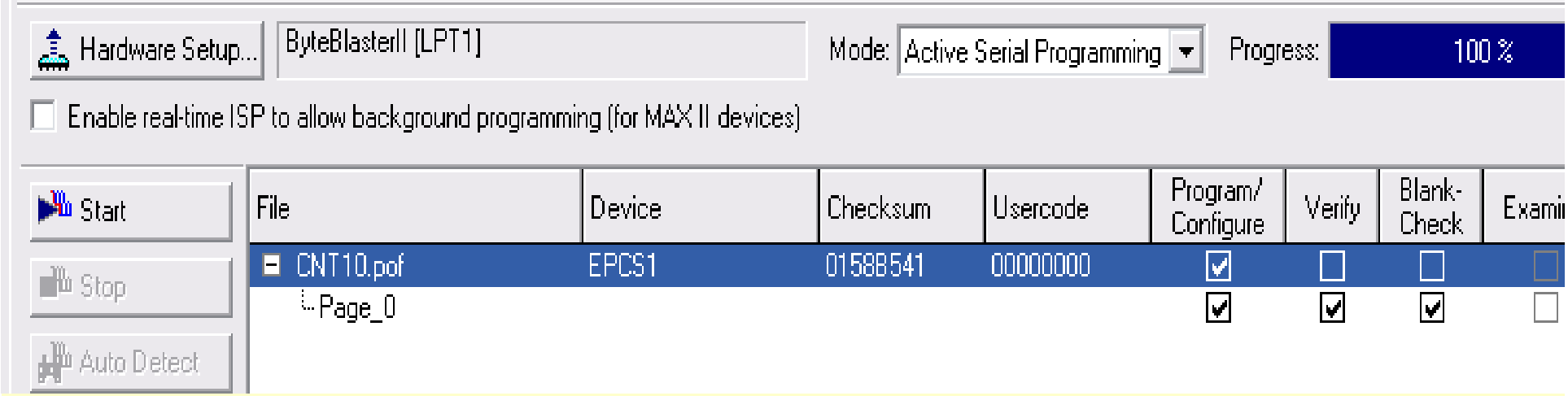


图5-29 ByteBlaster II接口AS模式编程窗口

5.2 引脚设置和下载

5.2.4 JTAG间接模式编程配置器件

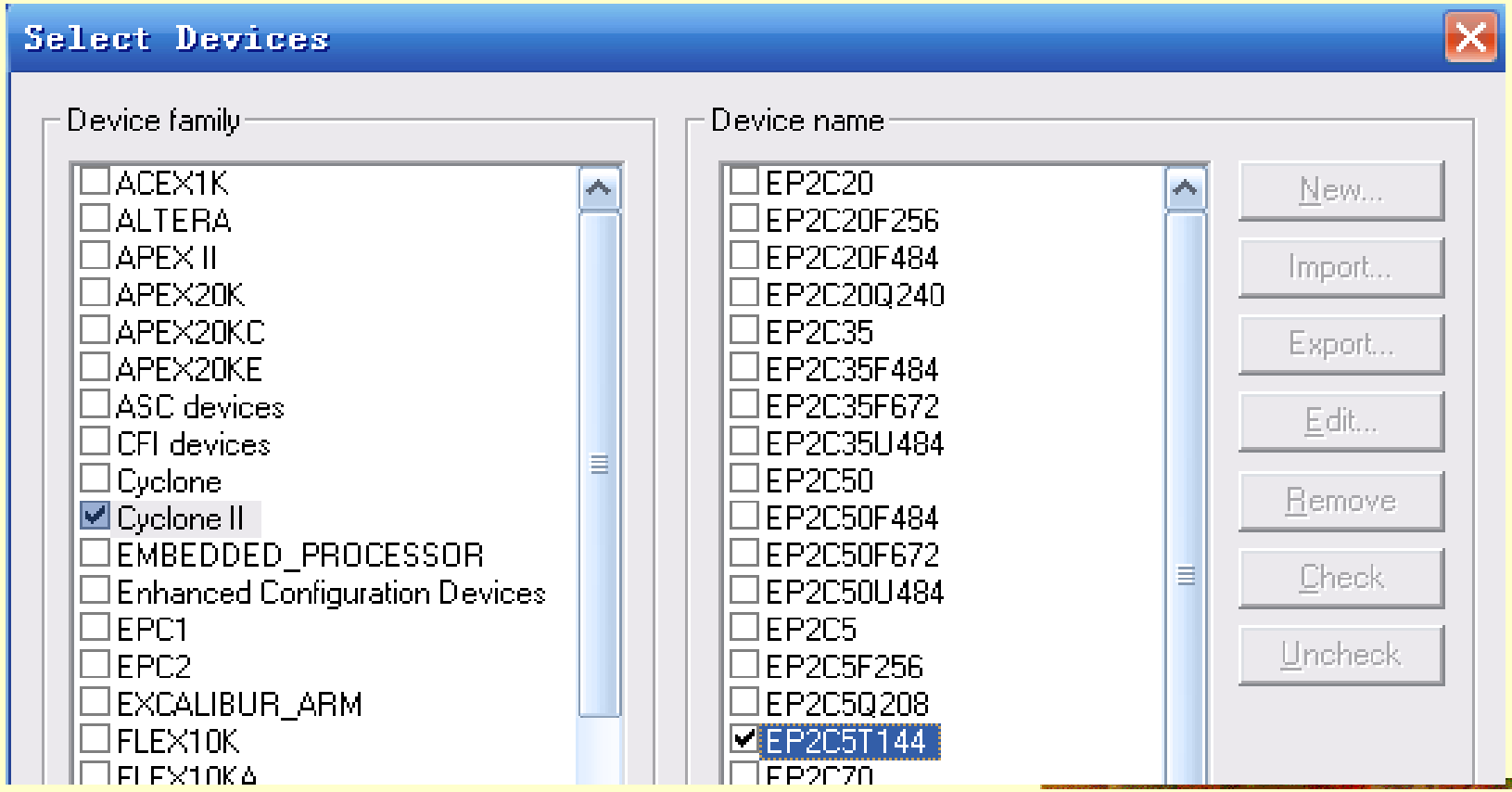


图5-30 选择目标器件EP2C5T144

5.2.4 JTAG间接模式编程配置器件

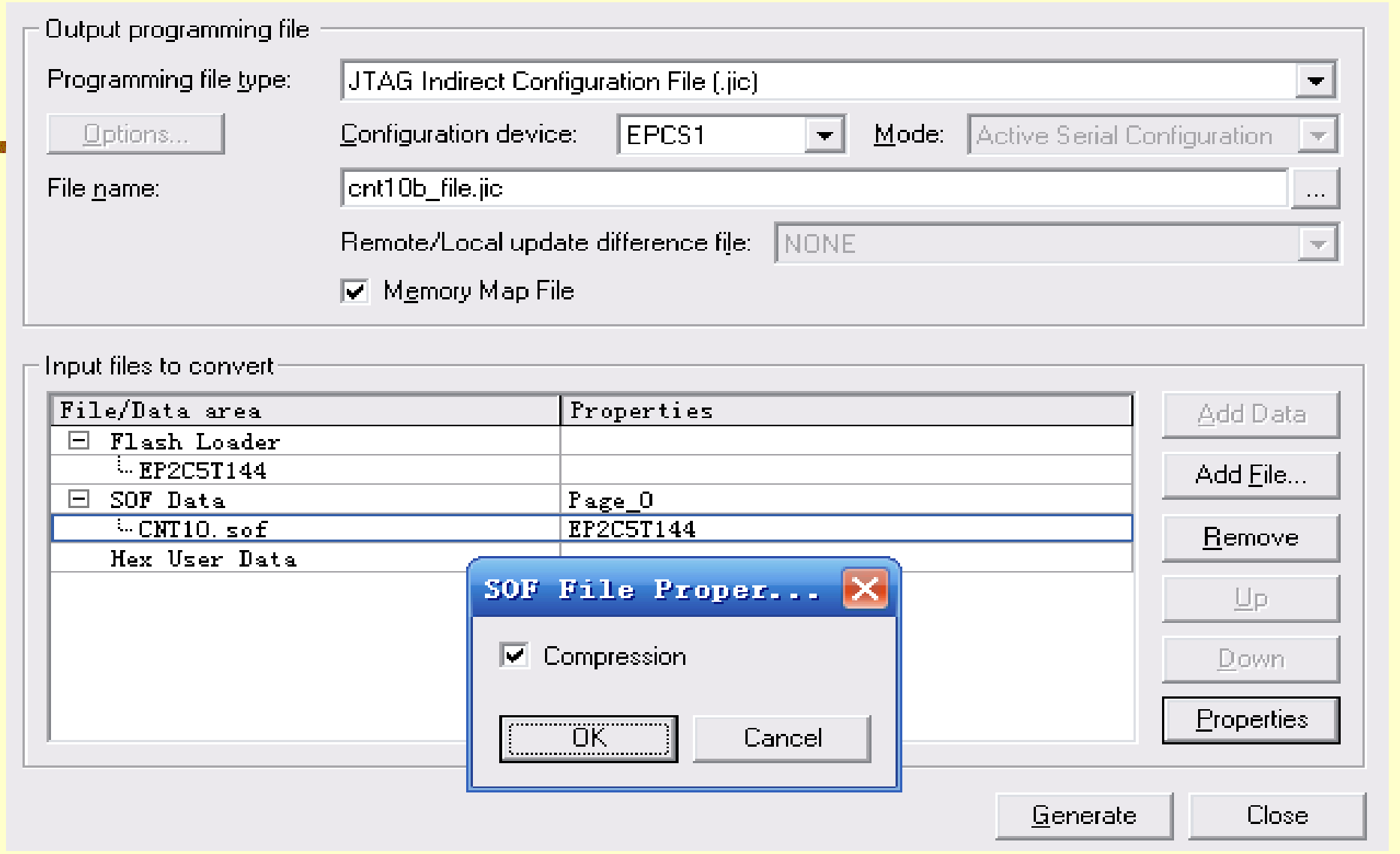


图5-31 选定SOF文件后，选择文件压缩

KX 康芯科技

5.2 引脚设置和下载

5.2.4 JTAG间接模式编程配置器件

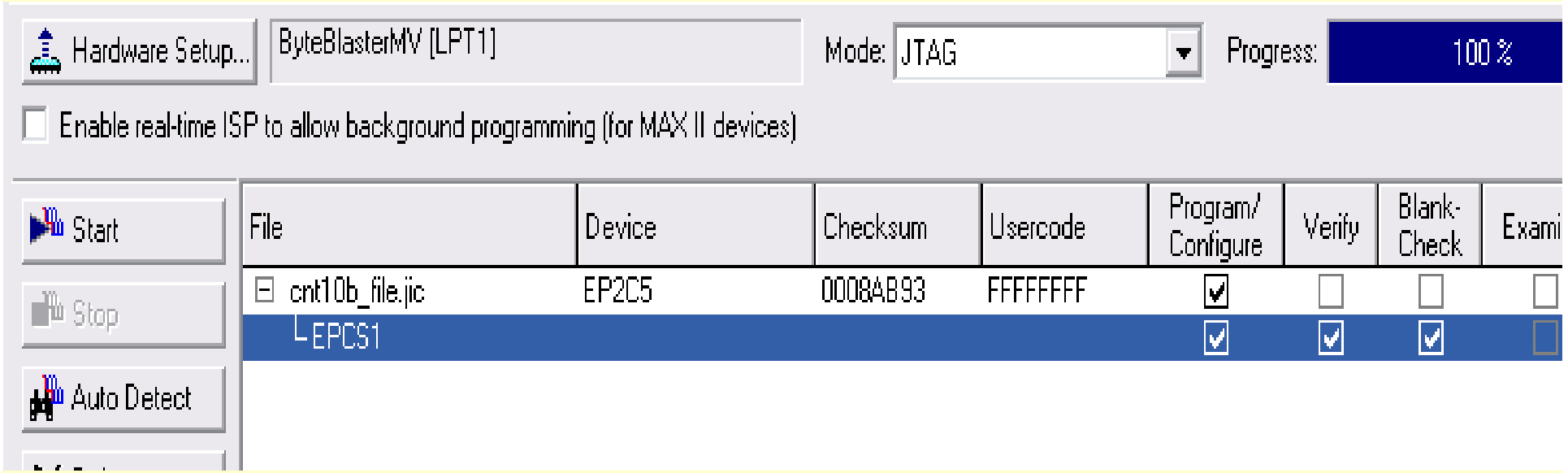


图5-32 用JTAG模式对配置器件EPCS1进行间接编程

5.2.5 USB Blaster编程配置器件使用方法

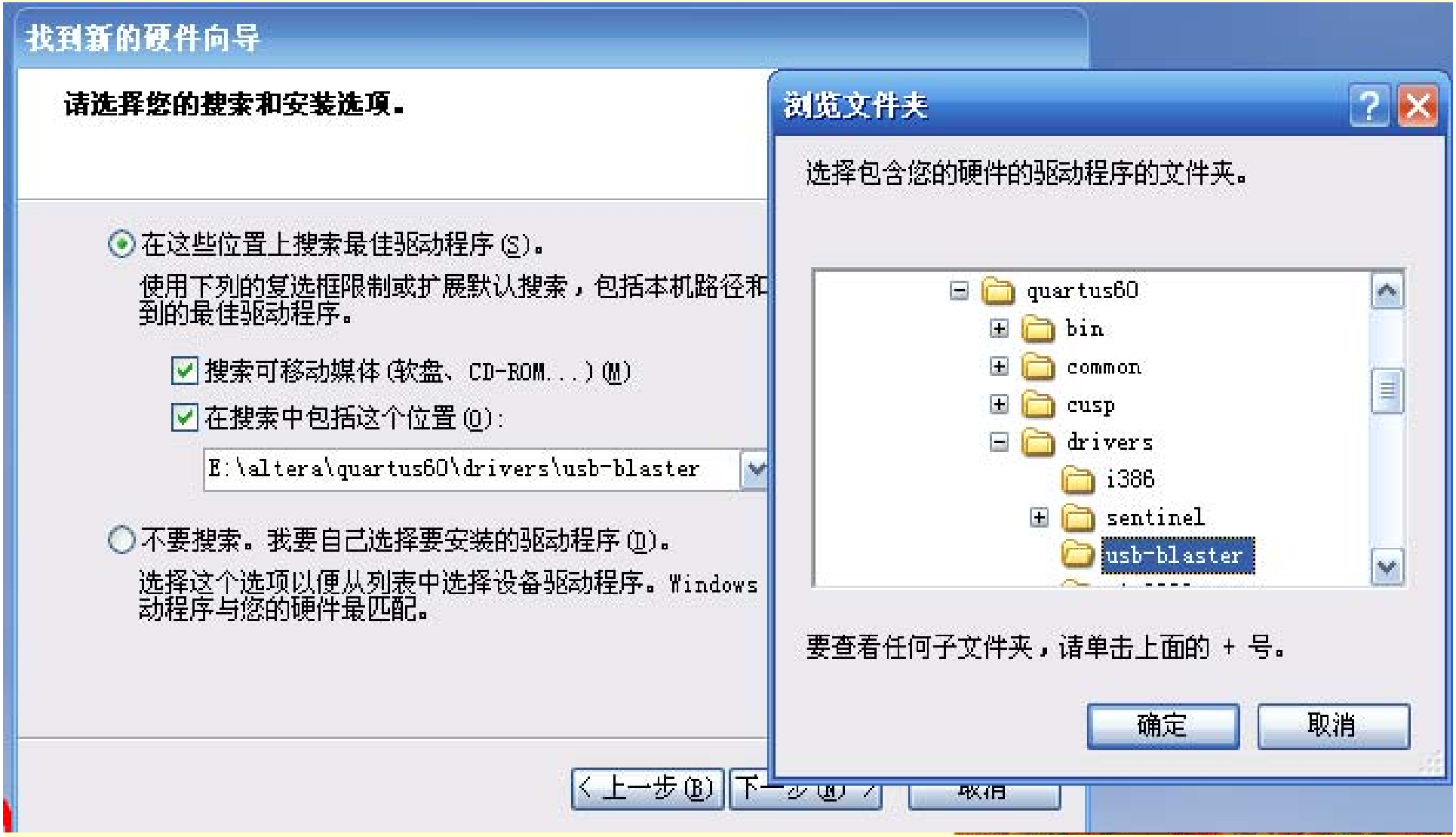


图5-33 安装USB驱动程序

5.2 引脚设置和下载

5.2.5 USB Blaster编程配置器件使用方法

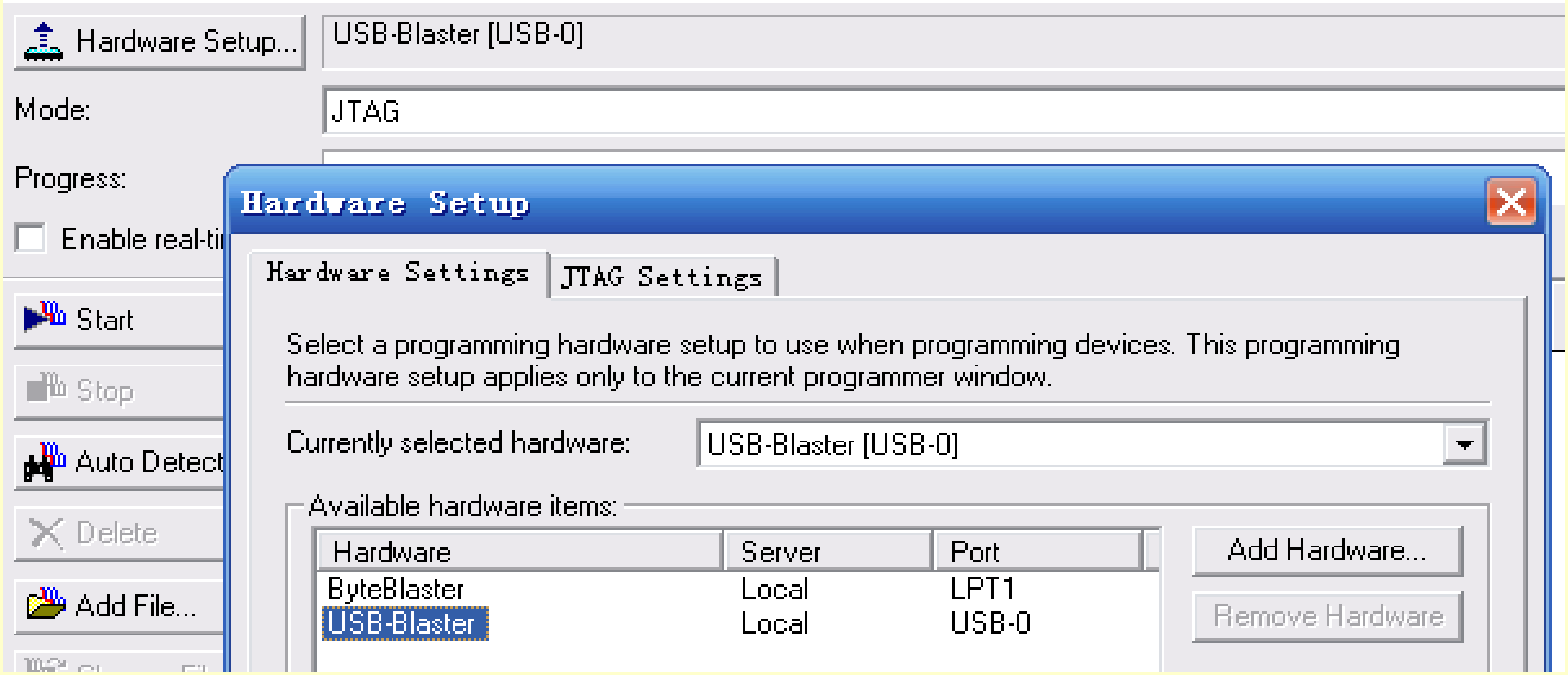


图5-34 设置JTAG硬件功能

5.2 引脚设置和下载

5.2.5 USB Blaster编程配置器件使用方法

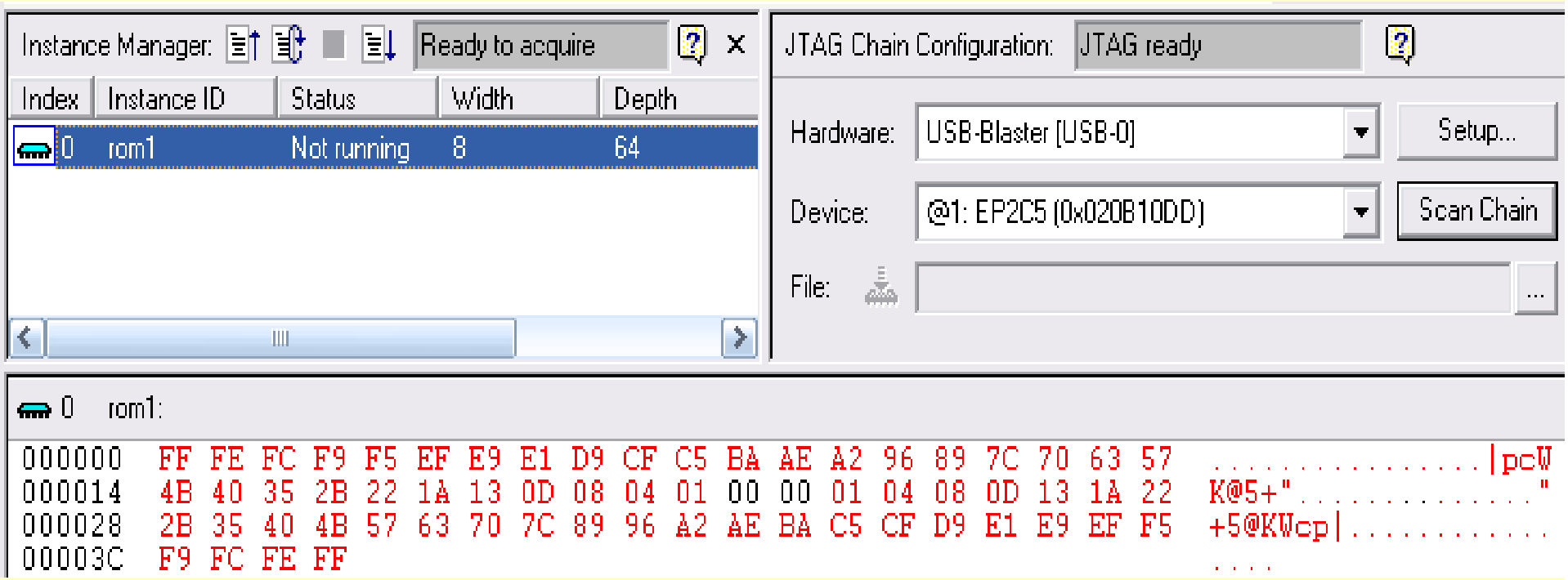


图5-35 在In-System Memory Content Editor中使用USB Blaster

5.3 嵌入式逻辑分析仪使用方法

1. 打开SignalTap II编辑窗

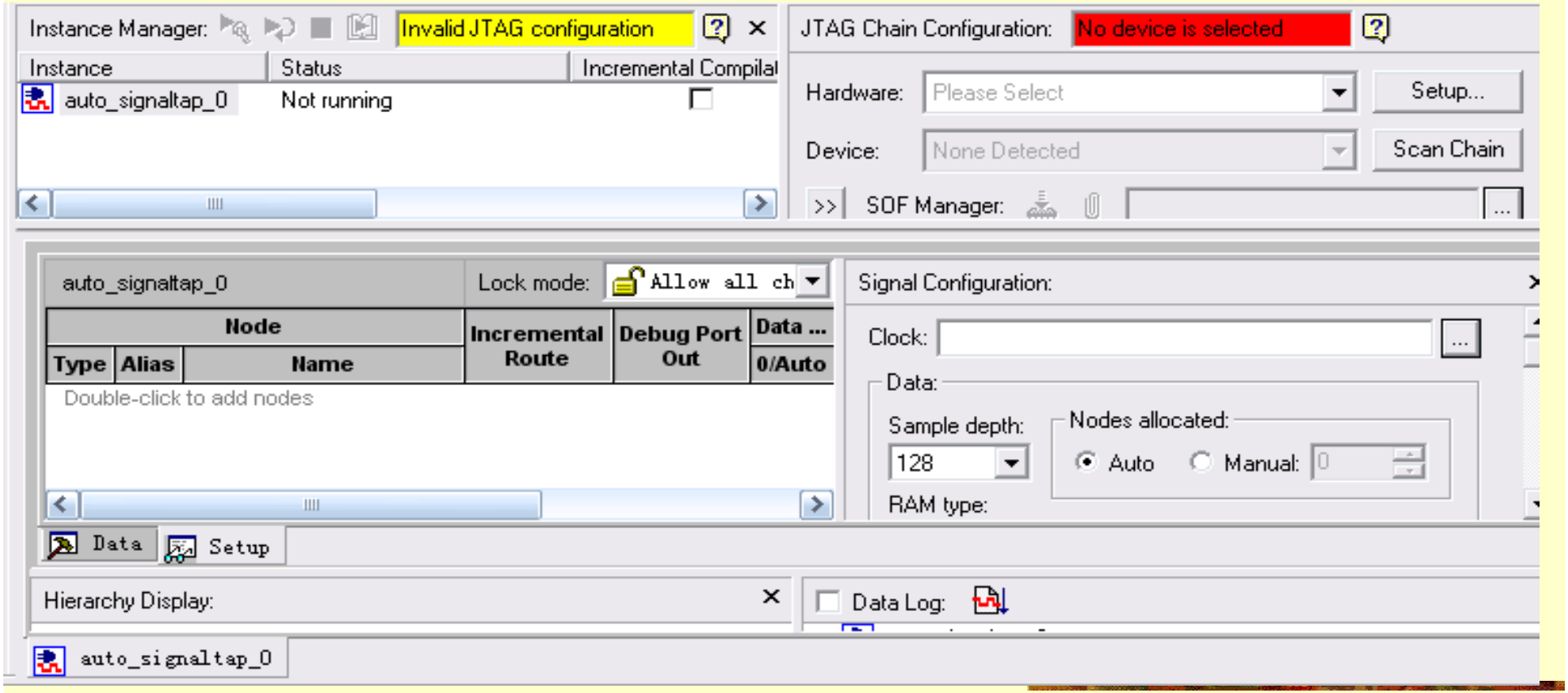


图5-36 SignalTap II编辑窗

5.3 嵌入式逻辑分析仪使用方法

2. 调入待测信号

3. SignalTap II参数设置

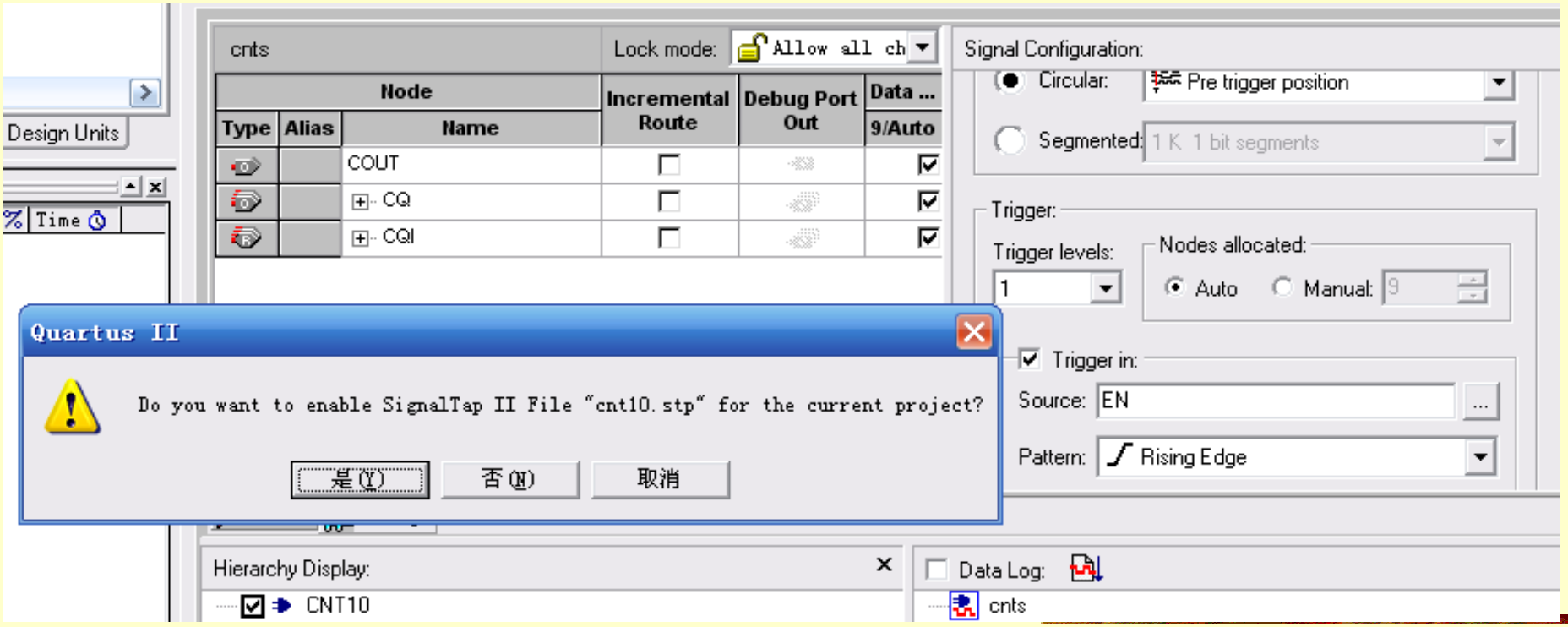


图5-37 SignalTap II编辑窗

KX 康芯科技

5.3 嵌入式逻辑分析仪使用方法

4. 文件存盘

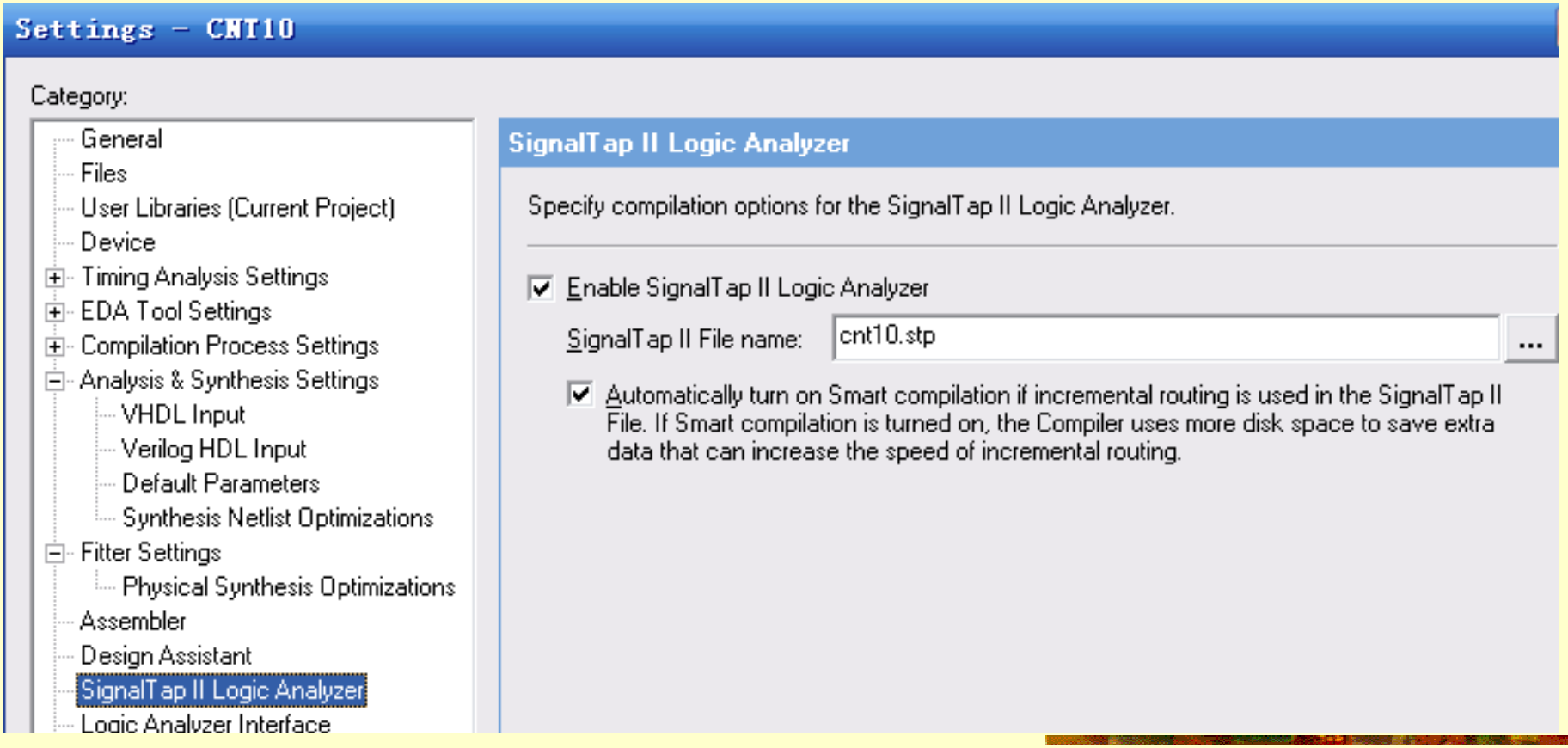


图5-38 设定SignalTap II与工程一同综合适配

5.3 嵌入式逻辑分析仪使用方法

5. 编译下载

6. 启动SignalTap II进行采样与分析

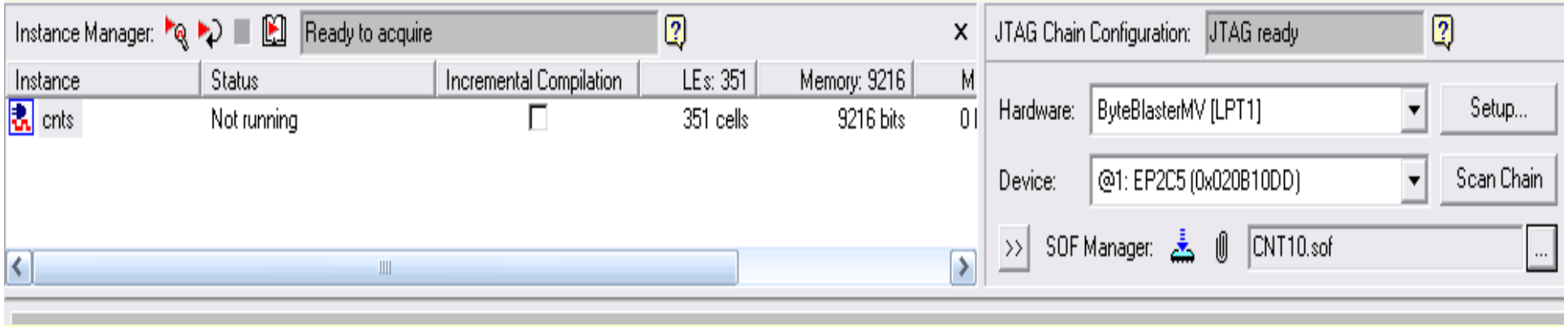


图5-39 下载cnt10.sof并准备启动SignalTap II

5.3 嵌入式逻辑分析仪使用方法

6. 启动SignalTap II进行采样与分析

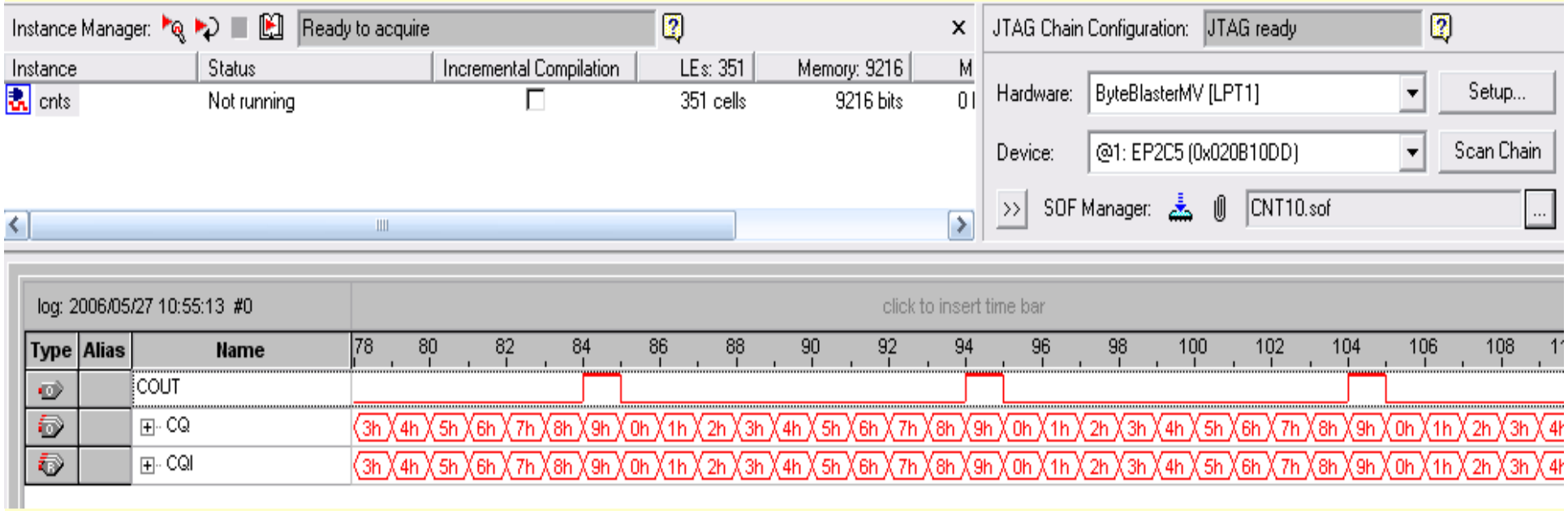


图5-40 SignalTap II采样已被启动

5.3 嵌入式逻辑分析仪使用方法

7. SignalTap II的其他设置和控制方法

图5-41 SignalTap II数据窗设置后的信号波形

5.4 原理图输入设计方法

5.4.1 设计流程

1. 为本项工程设计建立文件夹

假设本项设计的文件夹取名为**adder**，

路径为：**d:\adder**。

5.4 原理图输入设计方法

2. 输入设计项目和存盘

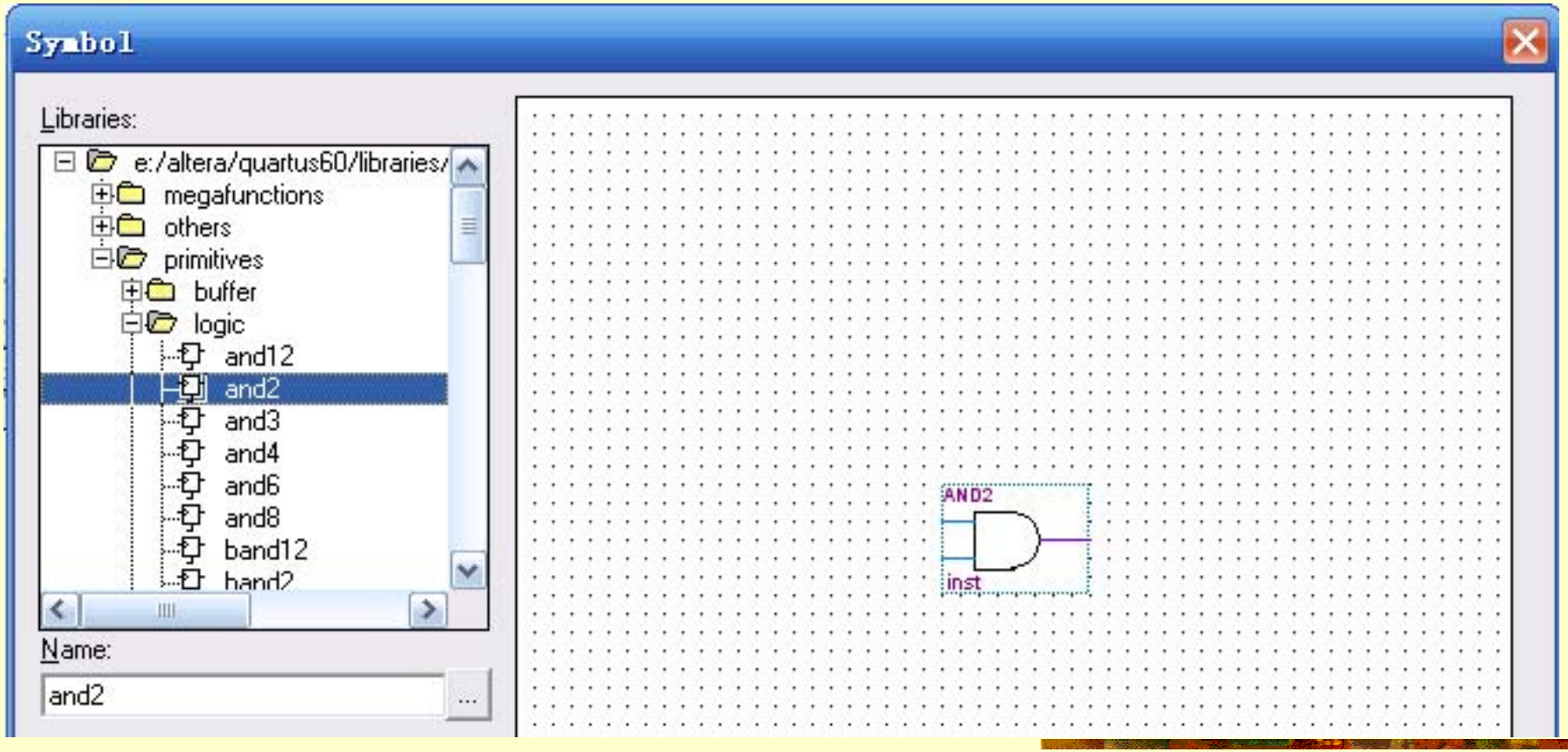


图5-42 元件输入对话框

5.4 原理图输入设计方法

3. 将设计项目设置成可调用的元件

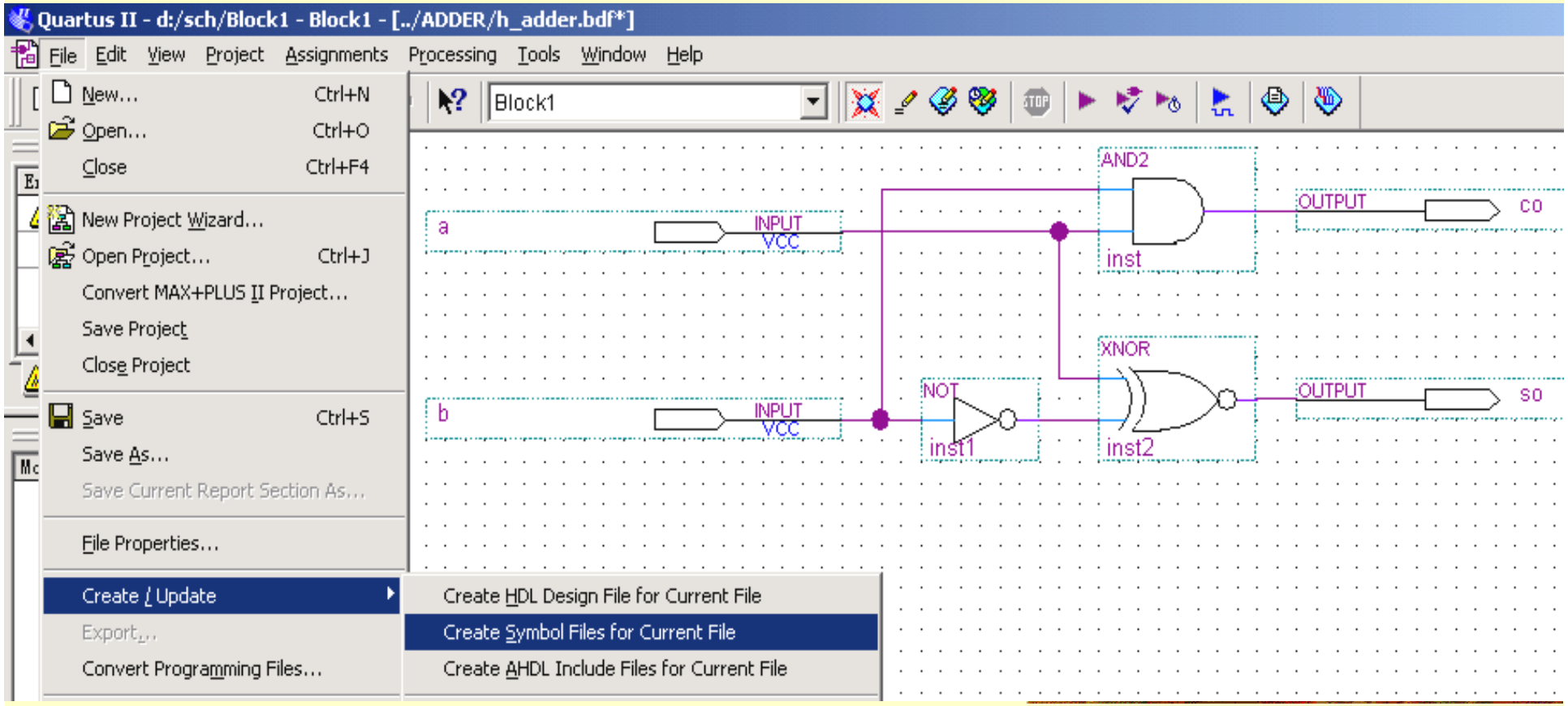


图5-43 将所需元件全部调入原理图编辑窗并连接好

5.4 原理图输入设计方法

4. 设计全加器顶层文件

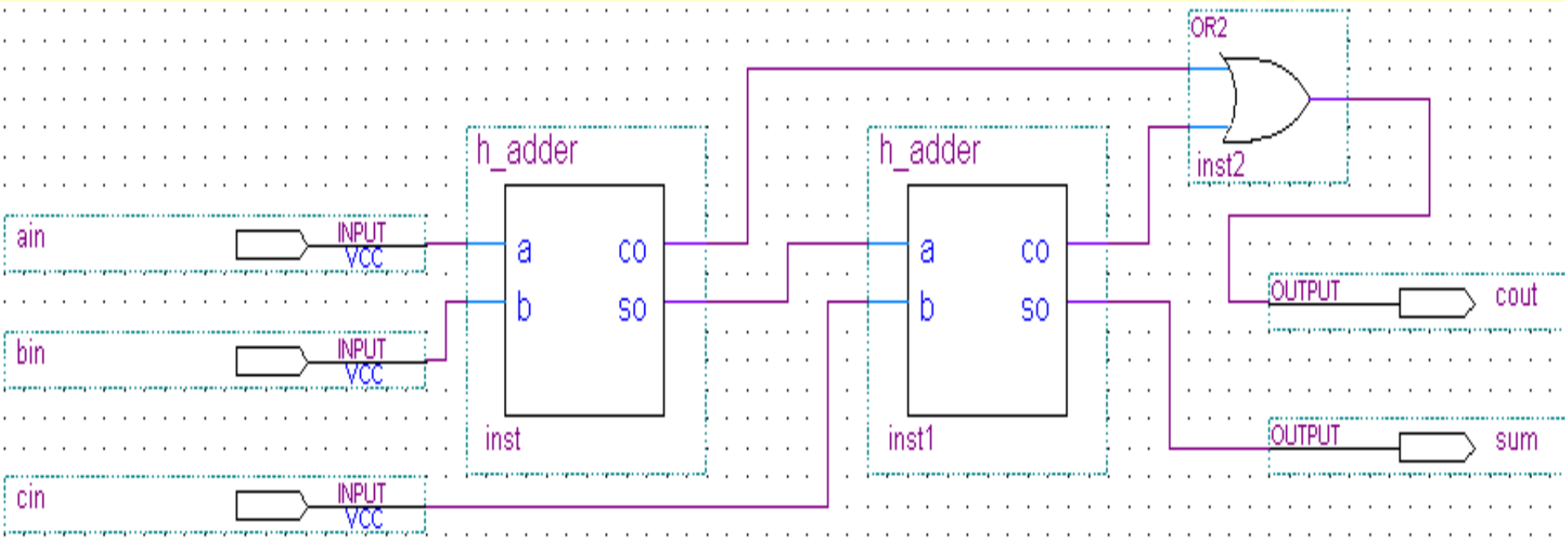


图5-44 连接好的全加器原理图f_adder.bdf

5.4 原理图输入设计方法

5. 将设计项目设置成工程和时序仿真

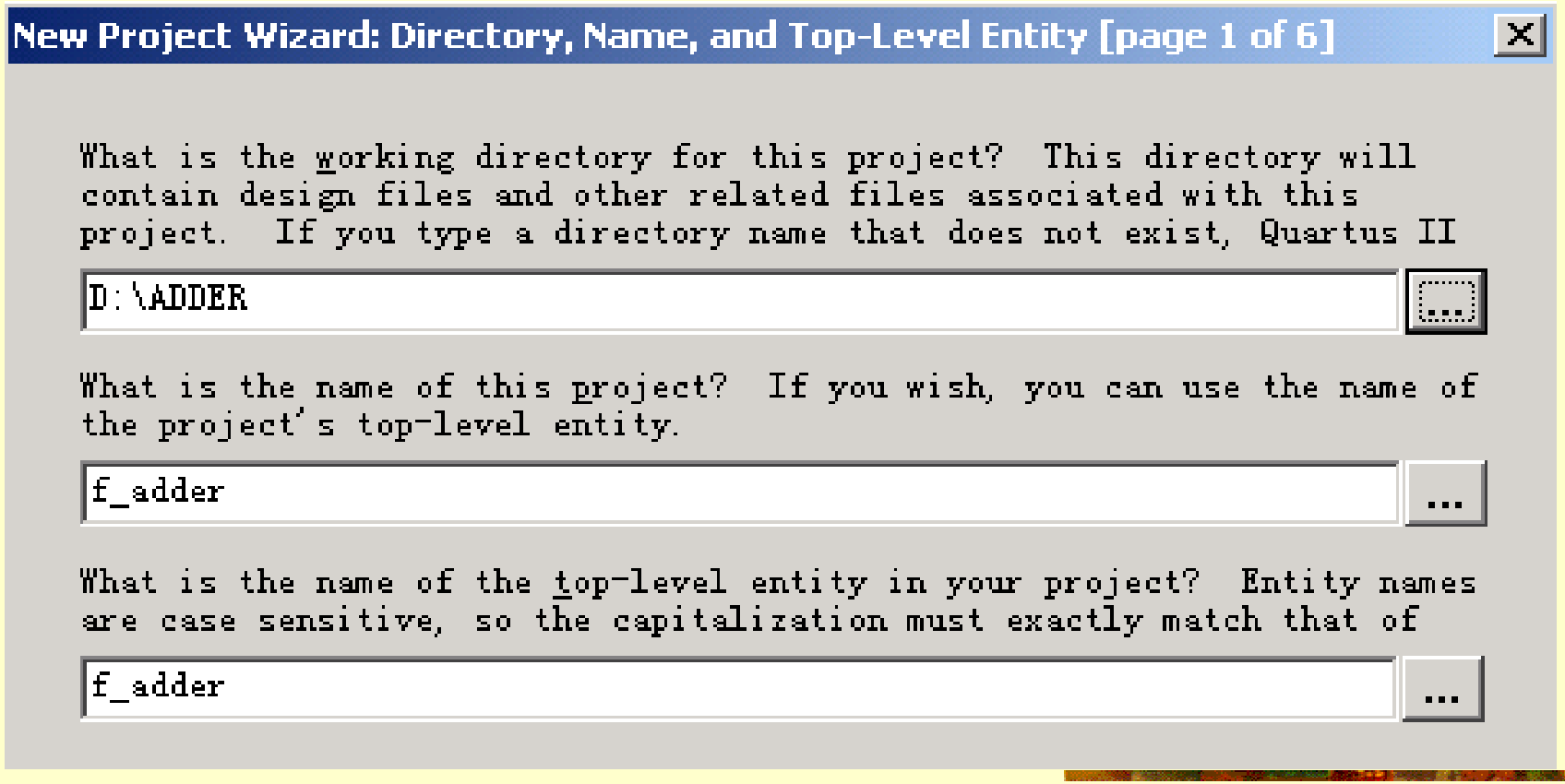


图5-45 f_adder.bdf工程设置窗

5.4 原理图输入设计方法

5. 将设计项目设置成工程和时序仿真

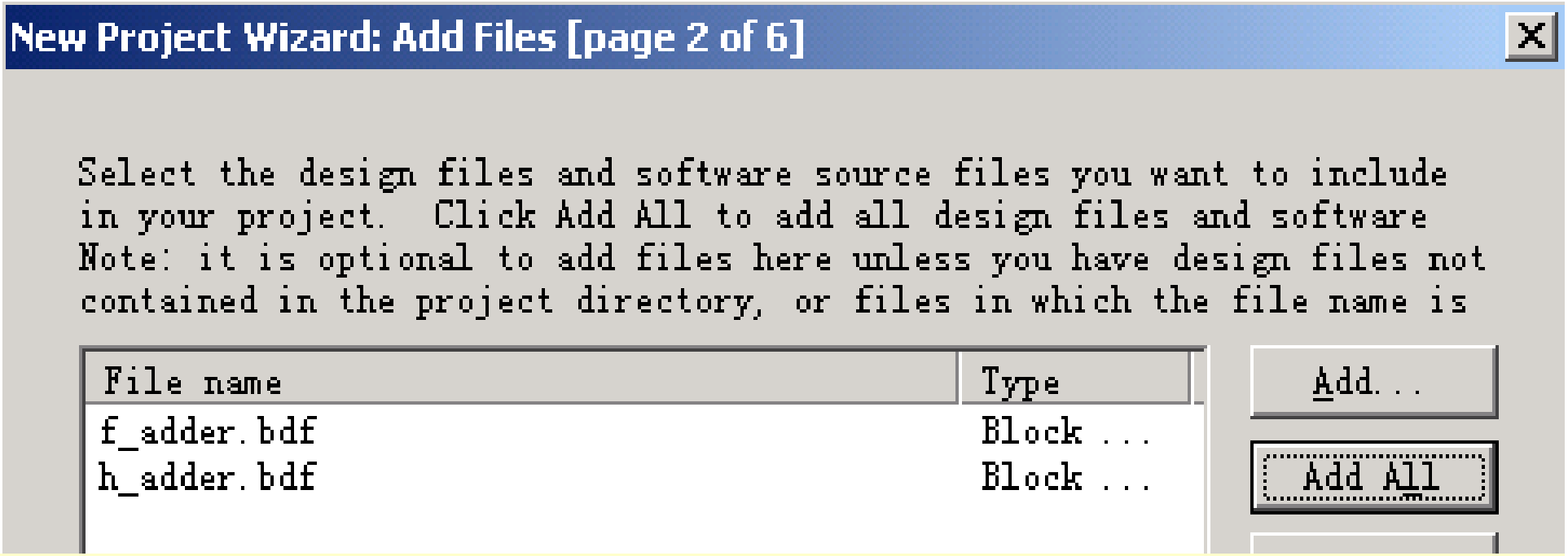


图5-46 加入本工程所有文件

5.4 原理图输入设计方法

5. 将设计项目设置成工程和时序仿真

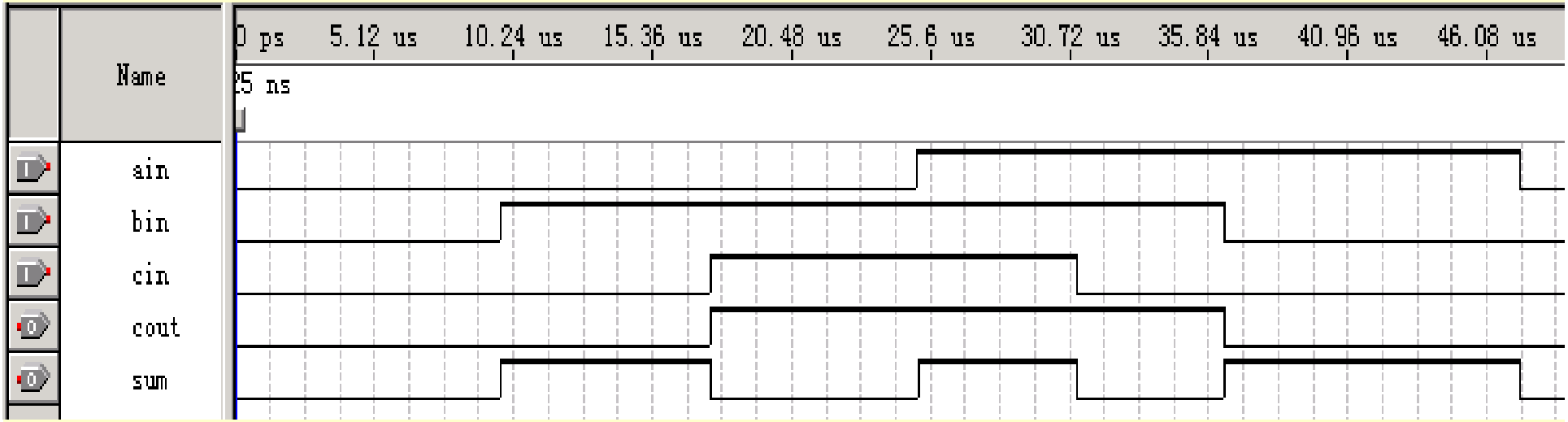


图5-47 全加器工程f_adder的仿真波形

5.4 原理图输入设计方法

5.4.2 应用宏模块的原理图设计

1. 计数器设计

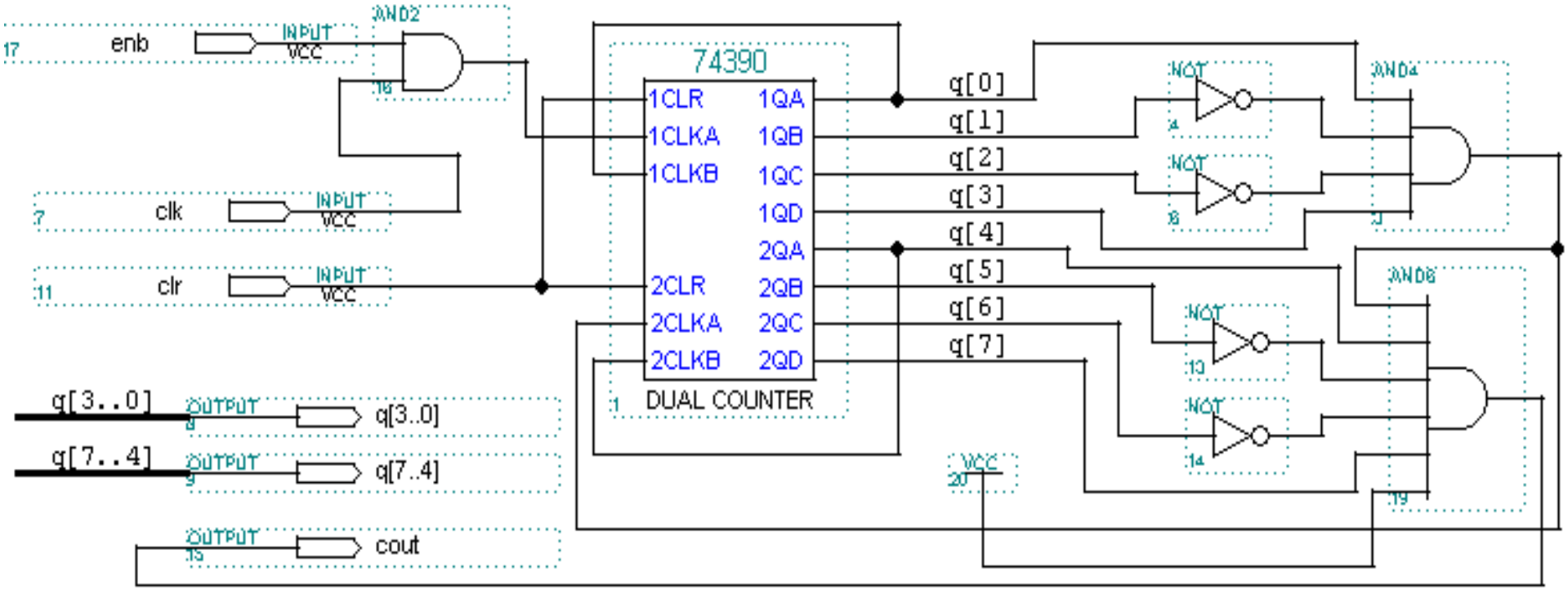


图5-48 含有时钟使能的两位十进制计数器

5.4 原理图输入设计方法

5.4.2 应用宏模块的原理图设计

1. 计数器设计

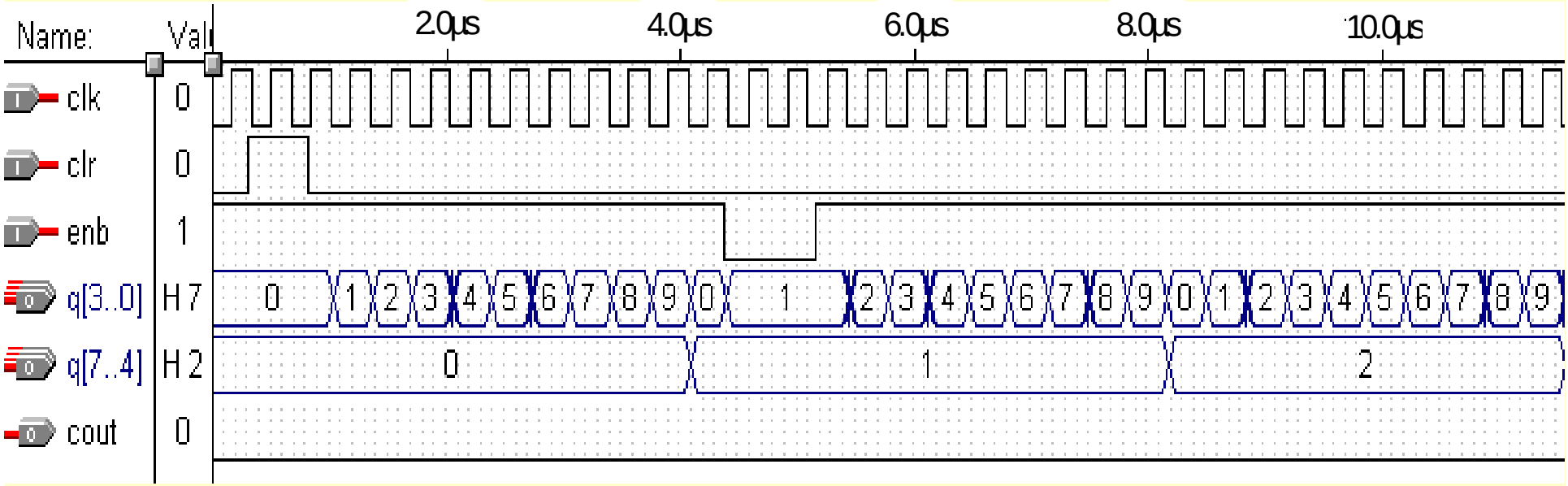


图5-49 两位十进制计数器工作波形

2. 频率计主结构电路设计

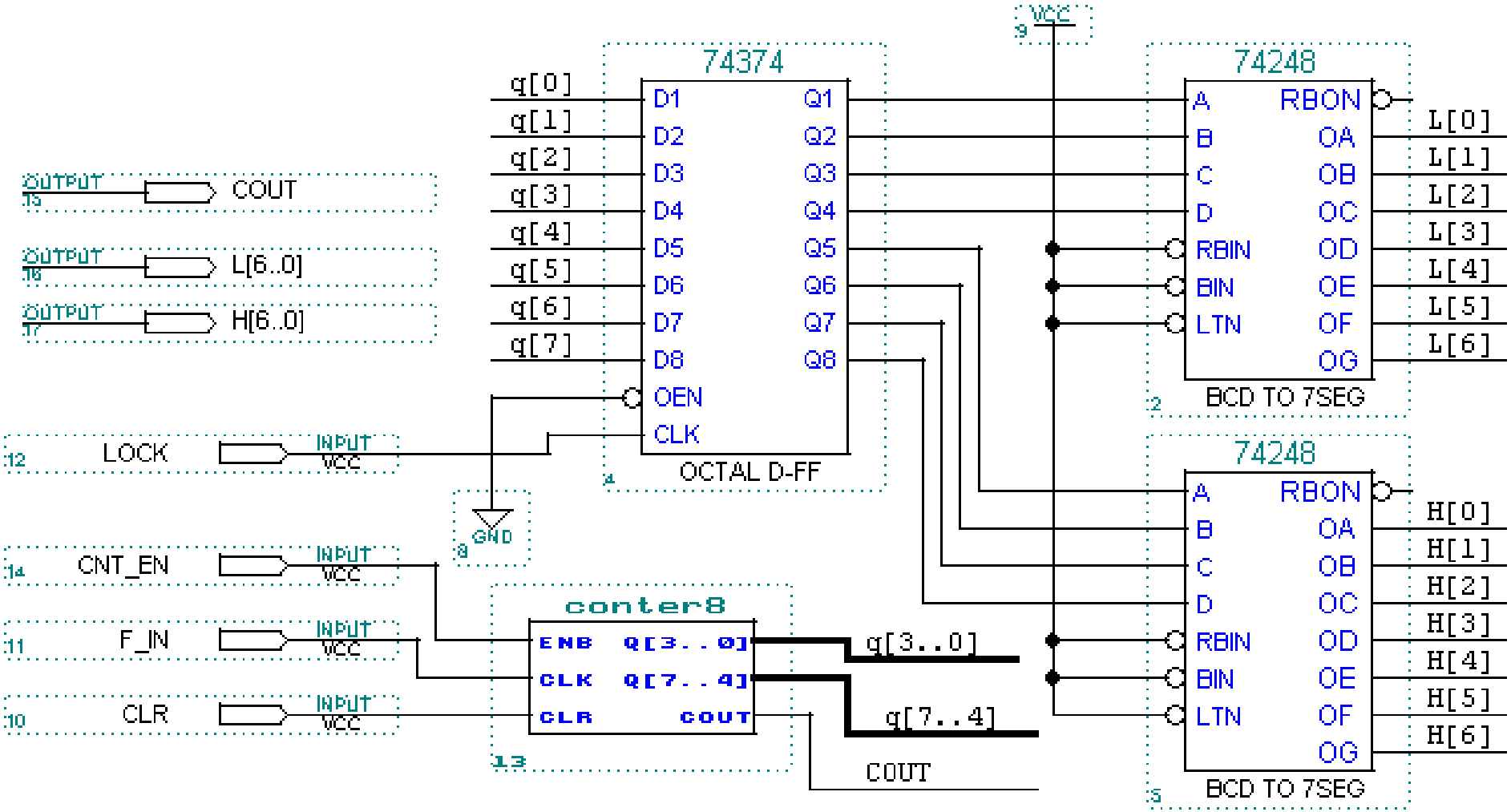


图5-50 两位十进制频率计顶层设计原理图文件

康芯科技

5.4 原理图输入设计方法

5.4.2 应用宏模块的原理图设计

2. 频率计主结构电路设计

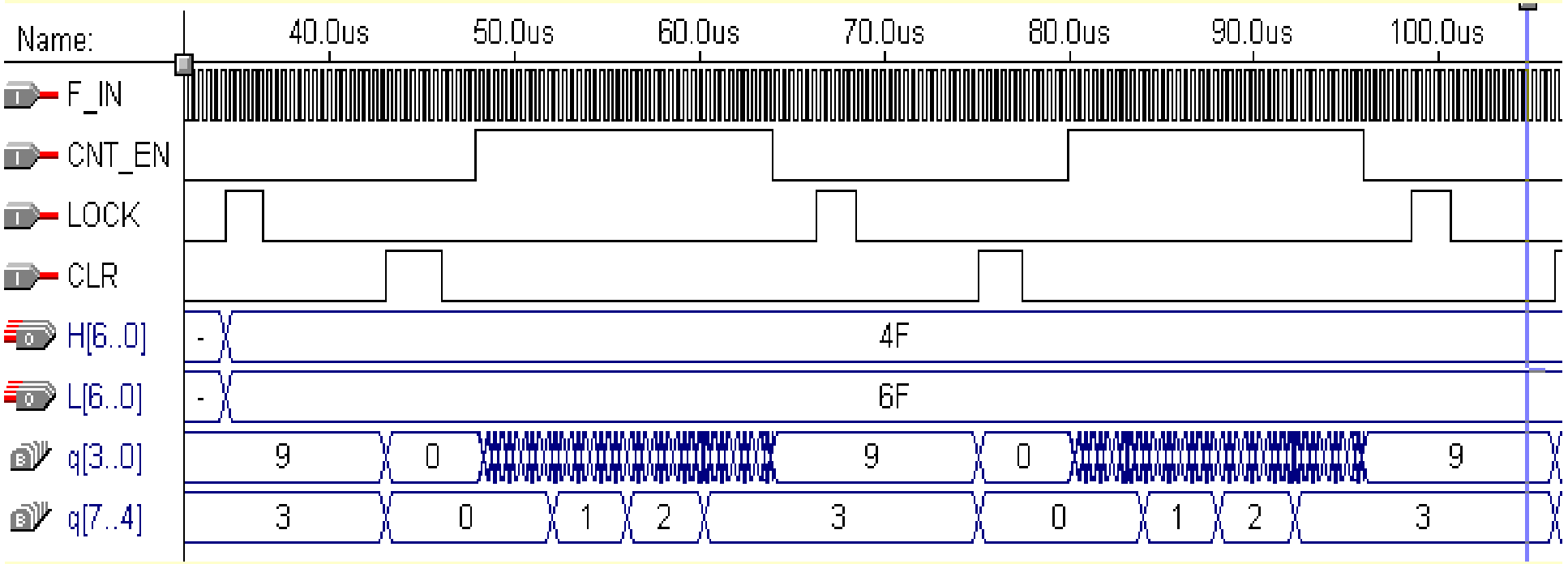


图5-51 两位十进制频率计测频仿真波形

5.4 原理图输入设计方法

3. 时序控制电路设计

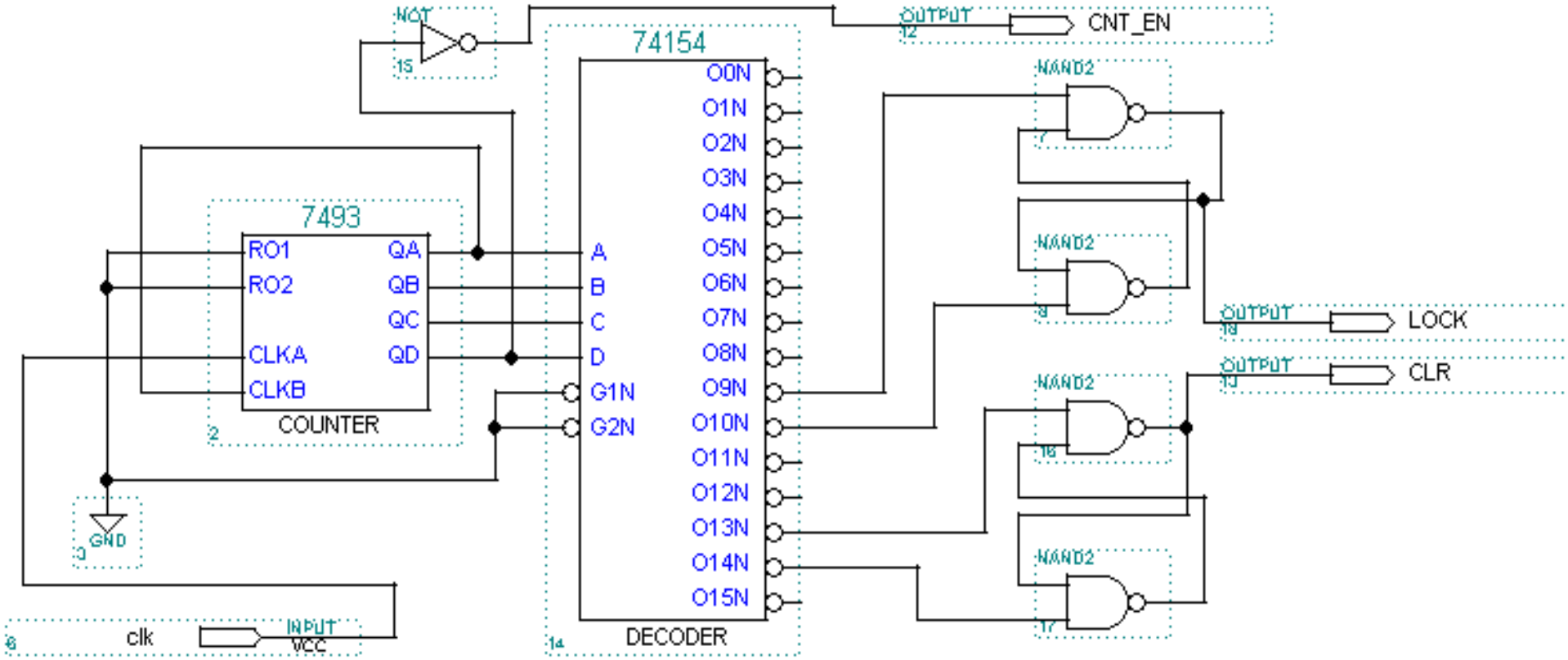


图5-52 测频时序控制电路

5.4 原理图输入设计方法

5.4.2 应用宏模块的原理图设计

3. 时序控制电路设计

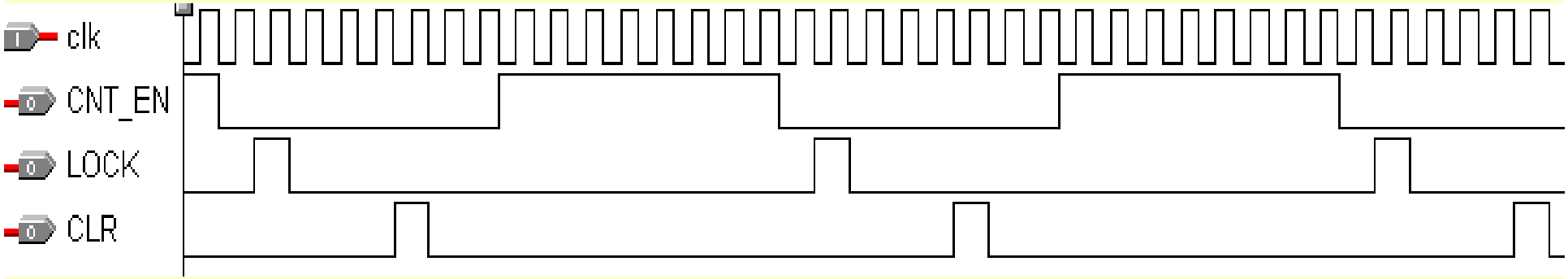


图5-53 测频时序控制电路工作波形

4. 顶层电路设计

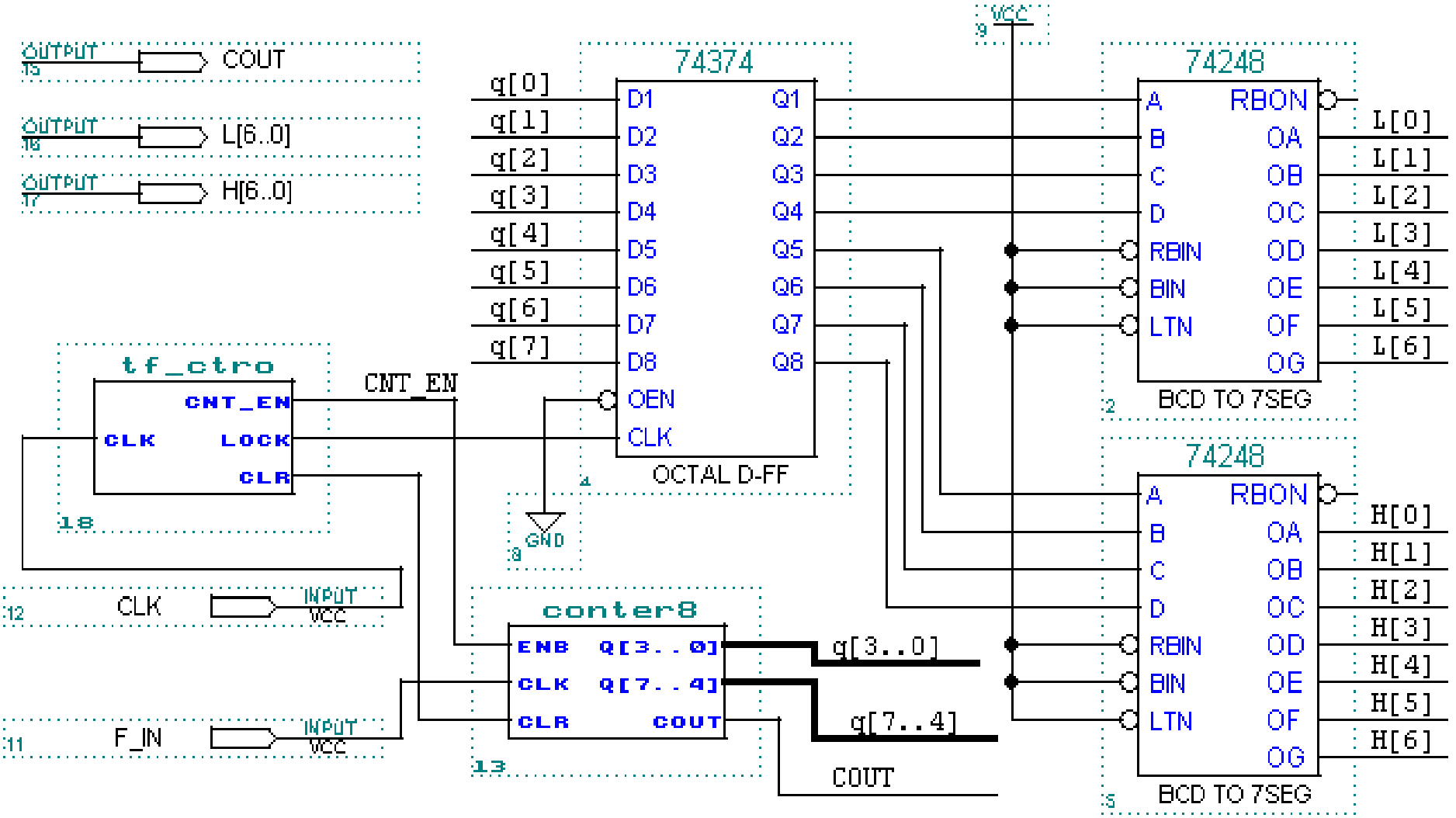


图5-54 频率计顶层电路原理图

康芯科技

5.4 原理图输入设计方法

5.4.2 应用宏模块的原理图设计

4. 顶层电路设计

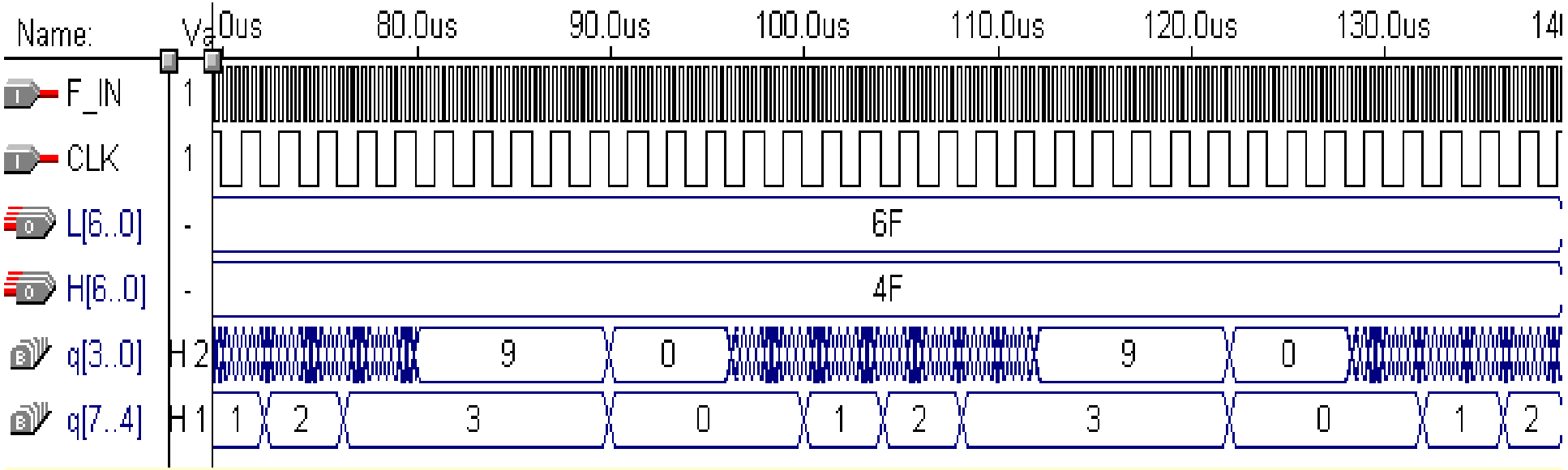


图5-55 频率计工作时序波形

习 题

5-1. 归纳利用**QuartusII**进行**VHDL**文本输入设计的流程：从文件输入一直到**SignalTap II**测试。

5-2. 由图5-40、5-41，详细说明工程设计cnt10的硬件工作情况。

5-3. 如何为设计中的**SignalTap II**加入独立采用时钟？试给出完整的程序和对它的实测结果。

习 题

5-4. 参考**Quartus II**的**Help**，详细说明**Assignments**菜单中**Settings**对话框的功能。

（1）说明其中的**Timing Requirements & Qptions**的功能、使用方法和检测途径。

（2）说明其中的**Compilation Process**的功能和使用方法。

（3）说明**Analysis & Synthesis Setting**的功能和使用方法，以及其中的**Synthesis Netlist Optimization**的功能和使用方法。

（4）说明**Fitter Settings**中的**Design Assistant**和**Simulator**功能，举例说明它们的使用方法。

习 题

5-5. 概述**Assignments**菜单中**Assignment Editor**的功能，举例说明。

5-6. 用**74148**和与非门实现**8421BCD**优先编码器，用**3**片**74139**组成一个**5-24**线译码器。

5-7. 用**74283**加法器和逻辑门设计实现一位**8421BCD**码加法器电路，输入输出均是**BCD**码，**CI**为低位的进位信号，**CO**为高位的进位信号，输入为两个**1**位十进制数**A**，输出用**S**表示。

5-8. 设计一个**7**人表决电路，参加表决者**7**人，同意为**1**，不同意为**0**，同意者过半则表决通过，绿指示灯亮；表决不通过则红指示灯亮。

5-9. 设计一个周期性产生二进制序列**01001011001**的序列发生器，用移位寄存器或用同步时序电路实现，并用时序仿真器验证其功能。

习 题

5-10. 用D触发器构成按循环码(000->001->011->111->101->100->000)规律工作的六进制同步计数器。

5-11. 应用4位全加器和74374构成4位二进制加法计数器。

5-12. 用74194、74273、D触发器等器件组成8位串入并出的转换电路，要求在转换过程中数据不变，只有当8位一组数据全部转换结束后，输出才变化一次。

如果使用74299、74373、D触发器和非门来完成上述功能，应该有怎样的电路？

5-13. 用一片74163和两片74138构成一个具有12路脉冲输出的数据分配器。要求在原理图上标明第1路到第12路输出的位置。若改用一片74195代替以上的74163，试完成同样的设计。

习 题

5-14. 用同步时序电路对串行二进制输入进行奇偶校验，每检测5位输入，输出一个结果。当5位输入中1的数目为奇数时，在最后一位的时刻输出1。

5-15. 用7490设计模为872的计数器，且输出的个位、十位、百位都应符合8421码权重。

5-16. 用74161设计一个97分频电路，用置0和置数两种方法实现。

5-17. 某通信接收机的同步信号为巴克码1110010。设计一个检测器，其输入为串行码x，输出为检测结果y，当检测到巴克码时，输出1。

实验与设计

5-1. 组合电路的设计

(1) 实验目的：熟悉Quartus Ⅱ的VHDL文本设计流程全过程，学习简单组合电路的设计、多层次电路设计、仿真和硬件测试。

(2) 实验内容1：首先利用Quartus Ⅱ完成2选1多路选择器（例4-3）的文本编辑输入(mux21a.vhd)和仿真测试等步骤，给出图4-3所示的仿真波形。最后在实验系统上进行硬件测试，验证本项设计的功能。

(3) 实验内容2：将此多路选择器看成是一个元件mux21a，利用元件例化语句描述图4-18，并将此文件放在同一目录中。以下是部分参考程序：

实验与设计

```
...
   COMPONENT MUX21A
     PORT (  a, b, s :   IN  STD_LOGIC;
                      y :  OUT STD_LOGIC);
   END COMPONENT ;
...
   u1 : MUX21A PORT MAP(a=>a2, b=>a3, s=>s0, y=>tmp);
   u2 : MUX21A PORT MAP(a=>a1, b=>tmp, s=>s1, y=>outy);
  END ARCHITECTURE BHV ;
```

按照本章给出的步骤对上例分别进行编译、综合、仿真。并对其仿真波形作出分析说明。

实验与设计

(4) 实验内容3：引脚锁定以及硬件下载测试。建议选实验电路模式5（附录图8），用键1(PIO0)控制s0；用键2(PIO1)控制s1；a3、a2和a1分别接clock5、clock0和clock2；输出信号outy仍接扬声器spker。通过短路帽选择clock0接256Hz信号，clock5接1024Hz，clock2接8Hz信号。最后进行编译、下载和硬件测试实验（通过选择键1、键2，控制s0、s1，可使扬声器输出不同音调）。

(5) 实验报告：根据以上的实验内容写出实验报告，包括程序设计、软件编译、仿真分析、硬件测试和详细实验过程；给出程序分析报告、仿真波形图及其分析报告。

实验与设计

(6) 附加内容：根据本实验以上提出的各项实验内容和实验要求，设计1位全加器。

首先用Quartus Ⅱ完成4.3节给出的全加器的设计，包括仿真和硬件测试。实验要求分别仿真测试底层硬件或门和半加器，最后完成顶层文件全加器的设计和测试，给出设计原程序，程序分析报告、仿真波形图及其分析报告。

(7) 实验习题：以1位二进制全加器为基本元件，用例化语句写出8位并行二进制全加器的顶层文件，并讨论此加法器的电路特性。

实验与设计

5-2. 时序电路的设计

(1) 实验目的： 熟悉Quartus Ⅱ 的VHDL文本设计过程，学习简单时序电路的设计、仿真和测试。

(2) 实验内容1： 根据实验5-1的步骤和要求，设计触发器(使用例4-6)，给出程序设计、软件编译、仿真分析、硬件测试及详细实验过程。

(3) 实验内容2： 设计锁存器(使用例4-14)，同样给出程序设计、软件编译、仿真分析、硬件测试及详细实验过程。

(4) 实验内容3： 只用一个1位二进制全加器为基本元件和一些辅助的时序电路，设计一个8位串行二进制全加器，要求：

1、能在8-9个时钟脉冲后完成8位二进制数（加数被加数的输入方式为并行）的加法运算，电路须考虑进位输入Cin和进位输出Cout；

实验与设计

2、给出此电路的时序波形，讨论其功能，并就工作速度与并行加法器进行比较；

3、在FPGA中进行实测。对于GW48 EDA实验系统，建议选择电路模式1（附录图3），键2，键1输入8位加数；键4，键3输入8位被加数；键8作为手动单步时钟输入；键7控制进位输入Cin；键9控制清0；数码6和数码5显示相加和；发光管D1显示溢出进位Cout。

4、键8作为相加起始控制，同时兼任清0；工作时钟由clock0自动给出，每当键8发出一次开始相加命令，电路即自动相加，结束后停止工作，并显示相加结果。就外部端口而言，与纯组合电路8位并行加法器相比，此串行加法器仅多出一个加法起始/清0控制输入和工作时钟输入端。

提示：此加法器有并/串和串/并移位寄存器各一。

(5) 实验报告：分析比较实验内容1和2的仿真和实测结果，说明这两种电路的异同点。

实验与设计

5-3. 设计含异步清0和同步时钟使能的加法计数器

(1) 实验目的：学习计数器的设计、仿真和硬件测试，进一步熟悉**VHDL**设计技术。

(2) 实验原理：实验程序为例**4-22**，实验原理参考**4.4**节，设计流程参考本章。

(3) 实验内容1：在**Quartus II**上对例**4-22**进行编辑、编译、综合、适配、仿真。说明例中各语句的作用，详细描述示例的功能特点，给出其所有信号的时序仿真波形。

(4) 实验内容2：引脚锁定以及硬件下载测试（参考**5.2**节）。引脚锁定后进行编译、下载和硬件测试实验。将实验过程和实验结果写进实验报告。

实验与设计

(5) 实验内容3：使用SignalTap II对此计数器进行实时测试，流程与要求参考5.3节。

(6) 实验内容4：从设计中去除SignalTap II，要求全程编译后生成用于配置器件EPCS1编程的压缩POF文件，并使用ByteBlasterII，通过AS模式对实验板上的EPCS1进行编程，最后进行验证。

(7) 实验内容4：为此项设计加入一个可用于SignalTapII采样的独立的时钟输入端（采用时钟选择clock0=12MHz，计数器时钟CLK分别选择256Hz、16384Hz、6MHz），并进行实时测试。

(8) 思考题：在例4-22中是否可以不定义信号 CQI，而直接用输出端口信号完成加法运算，即：

CQ <= CQ + 1？为什么？

(9) 实验报告：将实验原理、设计过程、编译仿真波形和分析结果、硬件测试实验结果写进实验报告。

实验与设计

5-4. 用原理图输入法设计8位全加器

(1) 实验目的：熟悉利用**Quartus II**的原理图输入方法设计简单组合电路，掌握层次化设计的方法，并通过一个**8**位全加器的设计把握利用**EDA**软件进行原理图输入方式的电子线路设计的详细流程。

(2) 实验原理：一个**8**位全加器可以由**8**个**1**位全加器构成，加法器间的进位可以串行方式实现，即将低位加法器的进位输出**cout**与相临的高位加法器的最低进位输入信号**cin**相接。而一个**1**位全加器可以按照**6.1**节介绍的方法来完成。

实验与设计

(3) 实验内容1：完成半加器和全加器的设计，包括原理图输入、编译、综合、适配、仿真、实验板上的硬件测试，并将此全加器电路设置成一个硬件符号入库。键1、键2、键3(PIO0/1/2)分别接ain、bin、cin；发光管D2、D1(PIO9/8)分别接sum和cout。

(4) 实验内容2，建立一个更高层次的原理图设计，利用以上获得的1位全加器构成8位全加器，并完成编译、综合、适配、仿真和硬件测试。建议选择电路模式1（附录图3）；键2、键1输入8位加数；键4、键3输入8位被加数；数码6/5显示加和；D8显示进位cout。

(5) 实验报告：详细叙述8位加法器的设计流程；给出各层次的原理图及其对应的仿真波形图；给出加法器的时序分析情况；最后给出硬件测试流程和结果。

实验与设计

5-5. 用原理图输入法设计较复杂数字系统

(1) 实验目的：熟悉原理图输入法中74系列等宏功能元件的使用方法，掌握更复杂的原理图层次化设计技术和数字系统设计方法。完成8位十进制频率机的设计。

(2) 原理说明：利用第4节介绍的2位计数器模块，连接它们的计数进位，用4个计数模块就能完成一个8位有时钟使能的计数器；对于测频控制器的控制信号，在仿真过程中应该注意它们可能的毛刺现象。最后按照设计流程和方法即可完成全部设计。

(3) 实验内容：首先完成2位频率计的设计，然后进行硬件测试，建议选择电路模式2；数码2和1显示输出频率值，待测频率F_IN接clock0；测频控制时钟CLK接clock2，若选择clock2 = 8Hz，门控信号CNT_EN的脉宽恰好为1秒。然后建立一个新的原理图设计层次，在此基础上将其扩展为8位频率计，仿真测试该频率计待测信号的最高频率，并与硬件实测的结果进行比较。

(4) 实验报告：给出各层次的原理图、工作原理、仿真波形图和分析，详述硬件实验过程和实验结果。

EDA 技术实用教程

第 6 章

VHDL设计进阶

6.1 数据对象

6.1.1 常数

常数定义的一般表述如下：

CONSTANT 常数名：数据类型 **:=** 表达式 **;**

```
CONSTANT  FBT : STD_LOGIC_VECTOR := "010110" ; -- 标准位矢类型
    CONSTANT  DATAIN : INTEGER := 15 ;          -- 整数类型
```

6.1 数据对象

6.1.2 变量

定义变量的一般表述如下：

VARIABLE 变量名 **:** 数据类型 **:=** 初始值 **;**

```
VARIABLE  a : INTEGER RANGE 0 TO 15 ;--变量a定义为常数，取值范围是0到5
VARIABLE  d : STD_LOGIC := '1' ;--变量a定义为标准逻辑位数据类型, 初始值是1
```

变量赋值的一般表述如下：

目标变量名 **:=** 表达式

6.1 数据对象

6.1.3 信号

SIGNAL 信号名： 数据类型 := 初始值 ;

目标信号名 <= 表达式 AFTER 时间量;

```
SIGNAL a, b, c, y, z: INTEGER ;
    ...
PROCESS (a, b, c)
BEGIN
   y <= a + b ;
   z <= c – a ;
   y <= b ;
END PROCESS ;
```

6.1 数据对象

6.1.4 进程中的信号与变量赋值

表6-1　信号与变量赋值语句功能的比较

	信号SIGNAL	变量VARIABLE
基本用法	用于作为电路中的信号连线	用于作为进程中局部数据存储单元
适用范围	在整个结构体内的任何地方都能适用	只能在所定义的进程中使用
行为特性	在进程的最后才对信号赋值	立即赋值

6.1 数据对象

6.1.4 进程中的信号与变量赋值

【例6-1】

```
. . .
  ARCHITECTURE bhv OF DFF3 IS
   BEGIN
   PROCESS (CLK)
    VARIABLE QQ : STD_LOGIC ;
    BEGIN
     IF  CLK'EVENT AND CLK = '1'  THEN QQ := D1 ;
     END IF;
  END PROCESS ;
       Q1 <= QQ;
END;
```

6.1 数据对象

6.1.4 进程中的信号与变量赋值

【例6-2】

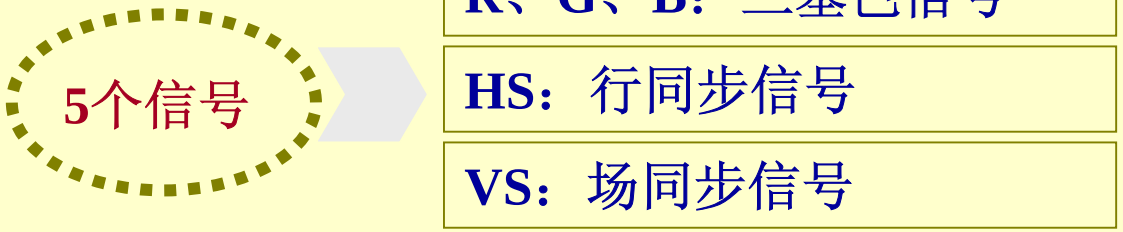
```
. . .
ARCHITECTURE bhv OF DFF3 IS
   SIGNAL QQ : STD_LOGIC ;
  BEGIN
  PROCESS (CLK)
   BEGIN
    IF  CLK'EVENT AND CLK = '1'  THEN  QQ <= D1 ;
    END IF;
 END PROCESS ;
      Q1 <= QQ;
END;
```

EDA 技术实用教程

第 13 章

电子系统设计实践

13.1 VGA彩条信号显示控制器设计



VGA工业标准要求的频率：

时钟频率(Clock frequency)：25.175 MHz (像素输出的频率)

行频(Line frequency) ： 31469 Hz

场频(Field frequency) ： 59.94 Hz (每秒图像刷新频率)

13.1 VGA彩条信号显示控制器设计

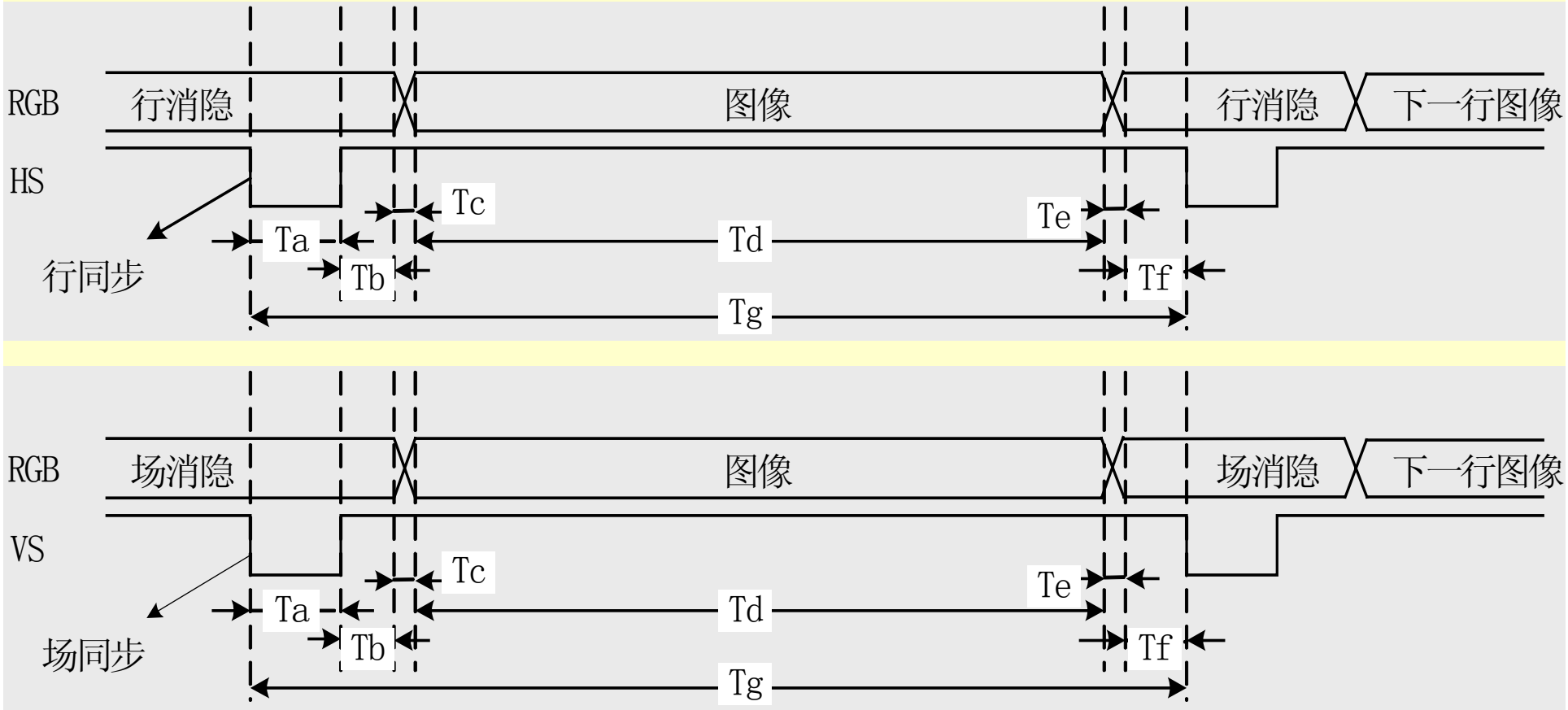


图13-1 VGA行扫描、场扫描时序示意图

KX 康芯科技

13.1 VGA彩条信号显示控制器设计

表13-1 行扫描时序要求：(单位：像素，即输出一个像素Pixel的时间间隔)

		行同步头			行图像		行周期
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg
时间(Pixels)	8	96	40	8	640	8	800

表13-1 行扫描时序要求：(单位：像素，即输出一个像素Pixel的时间间隔)

		行同步头			行图像		行周期
对应位置	Tf	Ta	Tb	Tc	Td	Te	Tg
时间(Lines)	2	2	25	8	480	8	525

13.1 VGA彩条信号显示控制器设计

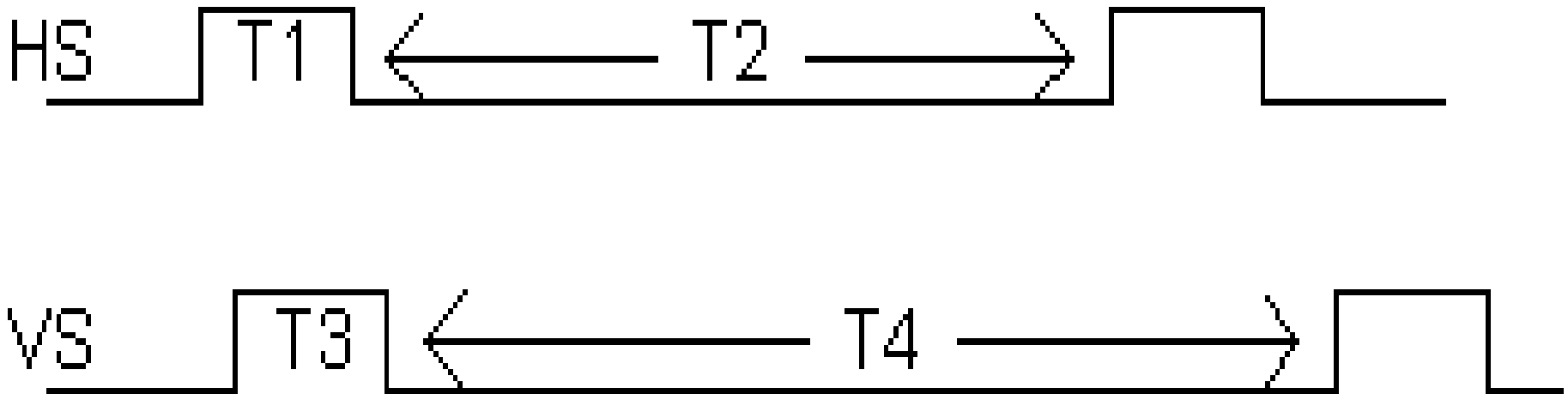


图13-2 HS和VS的时序图

13.1 VGA彩条信号显示控制器设计

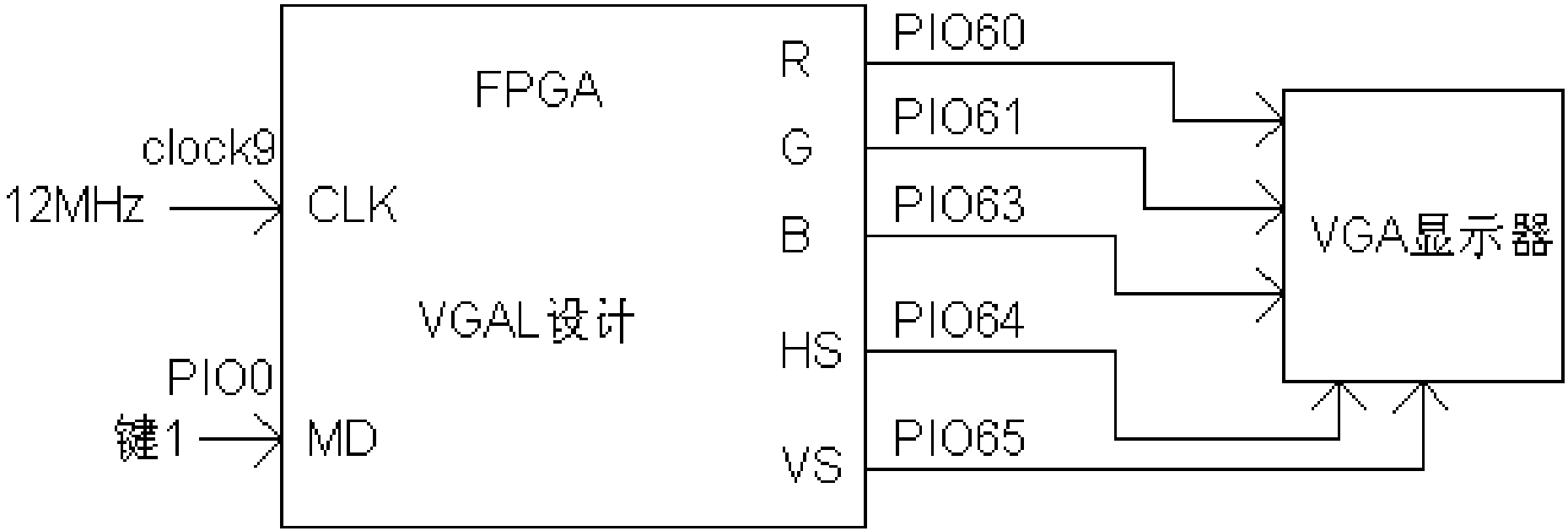


图13-3 例13-7实现电路

13.1 VGA彩条信号显示控制器设计

表13-3 颜色编码：

颜色	黑	蓝	红	品	绿	青	黄	白
R	0	0	0	0	1	1	1	1
G	0	0	1	1	0	0	1	1
B	0	1	0	1	0	1	0	1

表13-4彩条信号发生器3种显示模式

1	横彩条	1：白黄青绿品红蓝黑	2：黑蓝红品绿青黄白
2	竖彩条	1：白黄青绿品红蓝黑	2：黑蓝红品绿青黄白
3	棋盘格	1：棋盘格显示模式1	2：棋盘格显示模式2

13.1 VGA彩条信号显示控制器设计

【例13-1】

```
LIBRARY IEEE;    -- VGA显示器 彩条 发生器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY COLOR IS
    PORT (      CLK, MD : IN STD_LOGIC;
        HS, VS, R, G, B : OUT STD_LOGIC  ); -- 行场同步/红，绿，兰
END COLOR;
ARCHITECTURE behav OF COLOR IS
    SIGNAL HS1,VS1,FCLK,CCLK      : STD_LOGIC;
    SIGNAL MMD : STD_LOGIC_VECTOR(1 DOWNTO 0);-- 方式选择
    SIGNAL FS : STD_LOGIC_VECTOR (3 DOWNTO 0);
    SIGNAL CC : STD_LOGIC_VECTOR(4 DOWNTO 0); --行同步/横彩条生成
    SIGNAL LL : STD_LOGIC_VECTOR(8 DOWNTO 0); --场同步/竖彩条生成
    SIGNAL GRBX : STD_LOGIC_VECTOR(3 DOWNTO 1);-- X横彩条
    SIGNAL GRBY : STD_LOGIC_VECTOR(3 DOWNTO 1);-- Y竖彩条
    SIGNAL GRBP : STD_LOGIC_VECTOR(3 DOWNTO 1);
    SIGNAL GRB  : STD_LOGIC_VECTOR(3 DOWNTO 1);
BEGIN
```

（接下页）

```
GRB(2) <= (GRBP(2) XOR MD) AND HS1 AND VS1;
    GRB(3) <= (GRBP(3) XOR MD) AND HS1 AND VS1;
    GRB(1) <= (GRBP(1) XOR MD) AND HS1 AND VS1;
    PROCESS( MD )
    BEGIN
        IF MD'EVENT AND MD = '0' THEN
           IF MMD = "10" THEN  MMD <= "00";
            ELSE   MMD <= MMD + 1; END IF;        --三种模式
        END IF;
    END PROCESS;
    PROCESS( MMD )
    BEGIN
        IF MMD = "00" THEN     GRBP <= GRBX;      -- 选择横彩条
        ELSIF MMD = "01" THEN  GRBP <= GRBY;      -- 选择竖彩条
        ELSIF MMD = "10" THEN  GRBP <= GRBX XOR GRBY; --产生棋盘格
         ELSE  GRBP <= "000";  END IF;
    END PROCESS;
    PROCESS( CLK )
    BEGIN
        IF CLK'EVENT AND CLK = '1' THEN -- 13MHz 13分频
            IF FS = 13 THEN FS <= "0000";
            ELSE FS <= (FS + 1); END IF;
        END IF;
    END PROCESS;
    FCLK <= FS(3); CCLK <= CC(4);
    PROCESS( FCLK )
    BEGIN
```

（接下页）

```
IF FCLK'EVENT AND FCLK = '1' THEN
            IF CC = 29 THEN  CC <= "00000";
            ELSE  CC <= CC + 1; END IF;
        END IF;
    END PROCESS;
    PROCESS( CCLK )
    BEGIN
        IF CCLK'EVENT AND CCLK = '0' THEN
            IF LL = 481 THEN  LL <= "000000000";
            ELSE  LL <= LL + 1; END IF;
        END IF;
    END PROCESS;
    PROCESS( CC,LL )
    BEGIN
        IF CC > 23 THEN  HS1 <= '0';  --行同步
        ELSE HS1 <= '1'; END IF;
        IF LL > 479 THEN  VS1 <= '0'; --场同步
        ELSE  VS1 <= '1'; END IF;
    END PROCESS;
    PROCESS(CC, LL)
    BEGIN
           IF CC < 3  THEN GRBX <= "111";  -- 横彩条
        ELSIF CC < 6  THEN GRBX <= "110";
        ELSIF CC < 9  THEN GRBX <= "101";
        ELSIF CC < 13 THEN GRBX <= "100";
        ELSIF CC < 15 THEN GRBX <= "011";
```

（接下页）

```
ELSIF CC < 18 THEN GRBX <= "010";
        ELSIF CC < 21 THEN GRBX <= "001";
         ELSE GRBX <= "000";
           END IF;
           IF LL <  60 THEN GRBY <= "111";   -- 竖彩条
        ELSIF LL < 130 THEN GRBY <= "110";
        ELSIF LL < 180 THEN GRBY <= "101";
        ELSIF LL < 240 THEN GRBY <= "100";
        ELSIF LL < 300 THEN GRBY <= "011";
        ELSIF LL < 360 THEN GRBY <= "010";
        ELSIF LL < 420 THEN GRBY <= "001";
        ELSE GRBY <= "000";
           END IF;
    END PROCESS;
    HS <= HS1 ; VS <= VS1 ;R <= GRB(2) ;G <= GRB(3) ; B <= GRB(1);
END behav;
```

13.2 VGA图象显示控制器设计

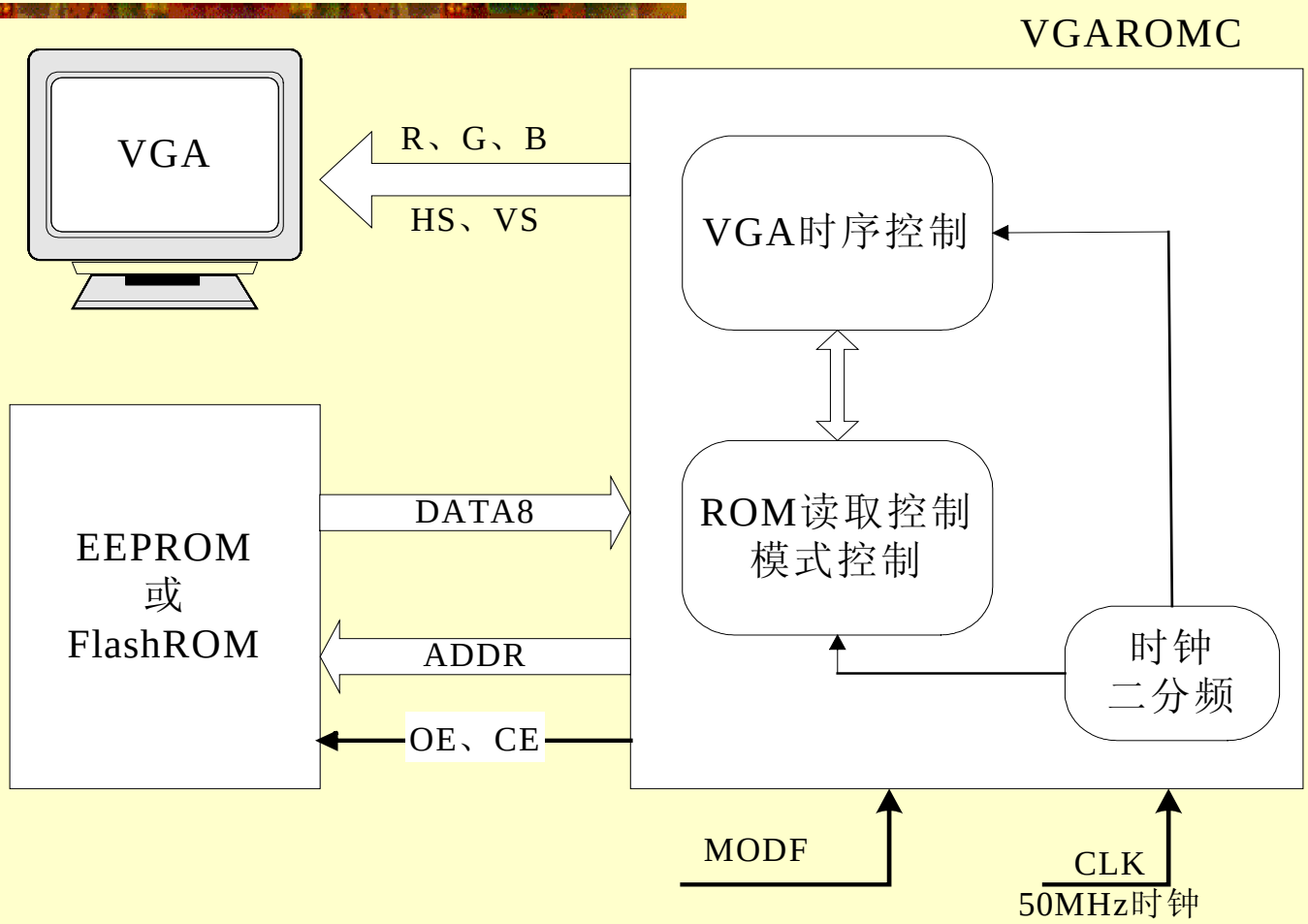


图13-4 VGA图像控制器框图

【例13-2】

```
LIBRARY ieee;  --图象显示顶层程序
USE ieee.std_logic_1164.all;
ENTITY img IS
       port
       ( clk50MHz :  IN  STD_LOGIC;
                 hs,       vs,       r, g, b :  OUT  STD_LOGIC );
END img;
ARCHITECTURE modelstru OF img IS
 component vga640480                --VGA显示控制模块
       PORT(clk : IN STD_LOGIC;
                 rgbin : IN STD_LOGIC_VECTOR(2 downto 0);
                 hs, vs, r, g, b : OUT STD_LOGIC;
                 hcntout, vcntout : OUT STD_LOGIC_VECTOR(9 downto 0)       );
end component;
component imgrom                   --图象数据ROM，数据线3位；地址线13位
       PORT(inclock : IN STD_LOGIC;
                 address : IN STD_LOGIC_VECTOR(11 downto 0);
                 q : OUT STD_LOGIC_VECTOR(2 downto 0)   );
end component;
signal rgb :  STD_LOGIC_VECTOR(2 downto 0);
signal clk25MHz : std_logic;
signal romaddr :  STD_LOGIC_VECTOR(11 downto 0);
signal hpos, vpos          :  std_logic_vector(9 downto 0);
BEGIN
romaddr <= vpos(5 downto 0) & hpos(5 downto 0);
process(clk50MHz) begin
if clk50MHz'event and clk50MHz = '1' then clk25MHz <= not clk25MHz ; end if;
end process;
i_vga640480 : vga640480 PORT MAP(clk => clk25MHz, rgbin => rgb, hs => hs,
vs => vs, r => r, g => g, b => b, hcntout => hpos, vcntout => vpos);
i_rom : imgrom PORT MAP(inclock => clk25MHz, address => romaddr, q => rgb);
END;
```

【例13-3】

```
LIBRARY IEEE
use IEEE.std_logic_1164.all;
use  IEEE.STD_LOGIC_UNSIGNED.ALL;
entity vga640480 is
     port (  clk     : in STD_LOGIC;
             hs,     vs,     r,g,b   : out STD_LOGIC;
             rgbin   : in std_logic_vector(2 downto 0);
             hcntout,vcntout        :  out  std_logic_vector(9  downto  0)
     );
end vga640480;
architecture ONE of vga640480 is
signal hcnt, vcnt   : std_logic_vector(9 downto 0);
begin
hcntout <= hcnt; vcntout <= vcnt;
process(clk) begin
     if (rising_edge(clk)) then
             if(hcnt < 800) then   hcnt <= hcnt + 1;
             else hcnt <= (others => '0'); end if;
     end if;
end process;
process(clk) begin
     if (rising_edge(clk)) then
             if (hcnt = 640+8 ) then
                     if(vcnt < 525) then   vcnt <= vcnt + 1;
                       else vcnt <= (others => '0');      end if;
```

（接下页）

```
        end if;
     end if;
end process;
process(clk) begin
     if (rising_edge(clk)) then
            if((hcnt>=640+8+8) and (hcnt<640+8+8+96 )) then hs<='0';
            else hs <= '1';        end if;
     end if;
end process;
process(vcnt) begin
     if ((vcnt >= 480+8+2) and (vcnt<480+8+2+2)) then vs <= '0';
            else vs<='1'; end if;
end process;
process(clk) begin
     if (rising_edge(clk)) then
            if (hcnt<640 and vcnt<480) then
                  r<=rgbin(2); g<=rgbin(1); b<=rgbin(0);
            else r<='0'; g<='0'; b<='0'; end if;
     end if;
end process;
end ONE;
```

KX 康芯科技

13.3 步进电机细分驱动控制

1、步进电机细分驱动原理

2、步距细分的系统构成

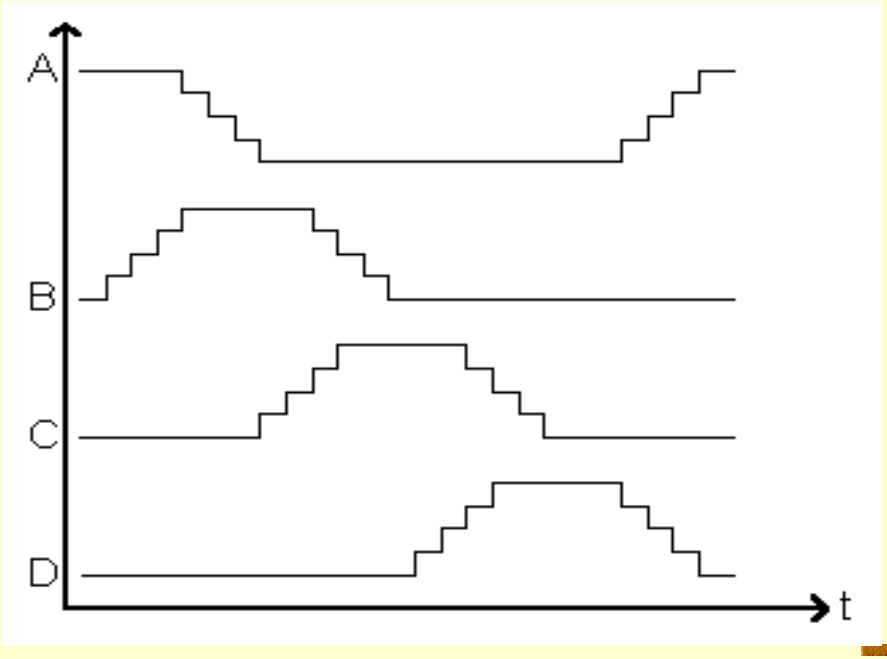


图13-5 四相步进电机8细分电流波形

13.3 步进电机细分驱动控制

2、步距细分的系统构成

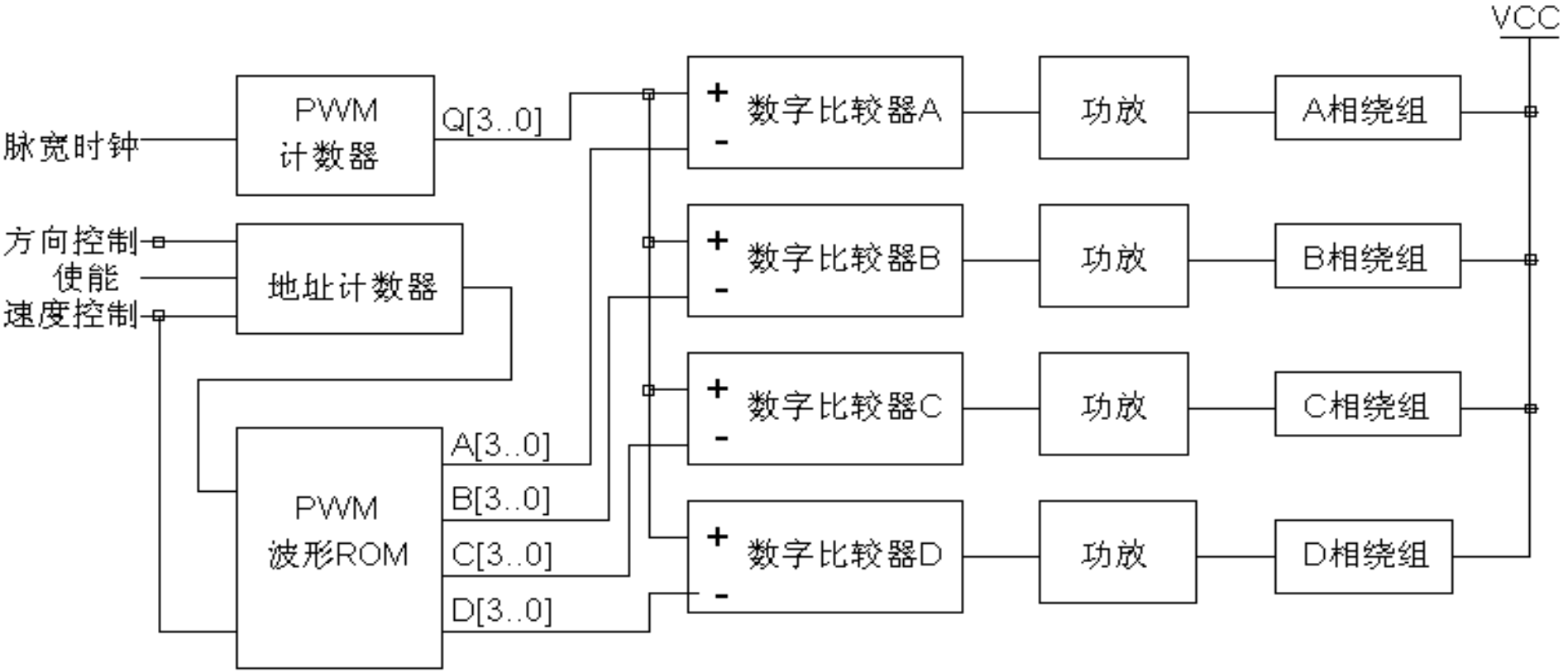


图13-6 步进电机细分驱动电路结构图

13.3 步进电机细分驱动控制

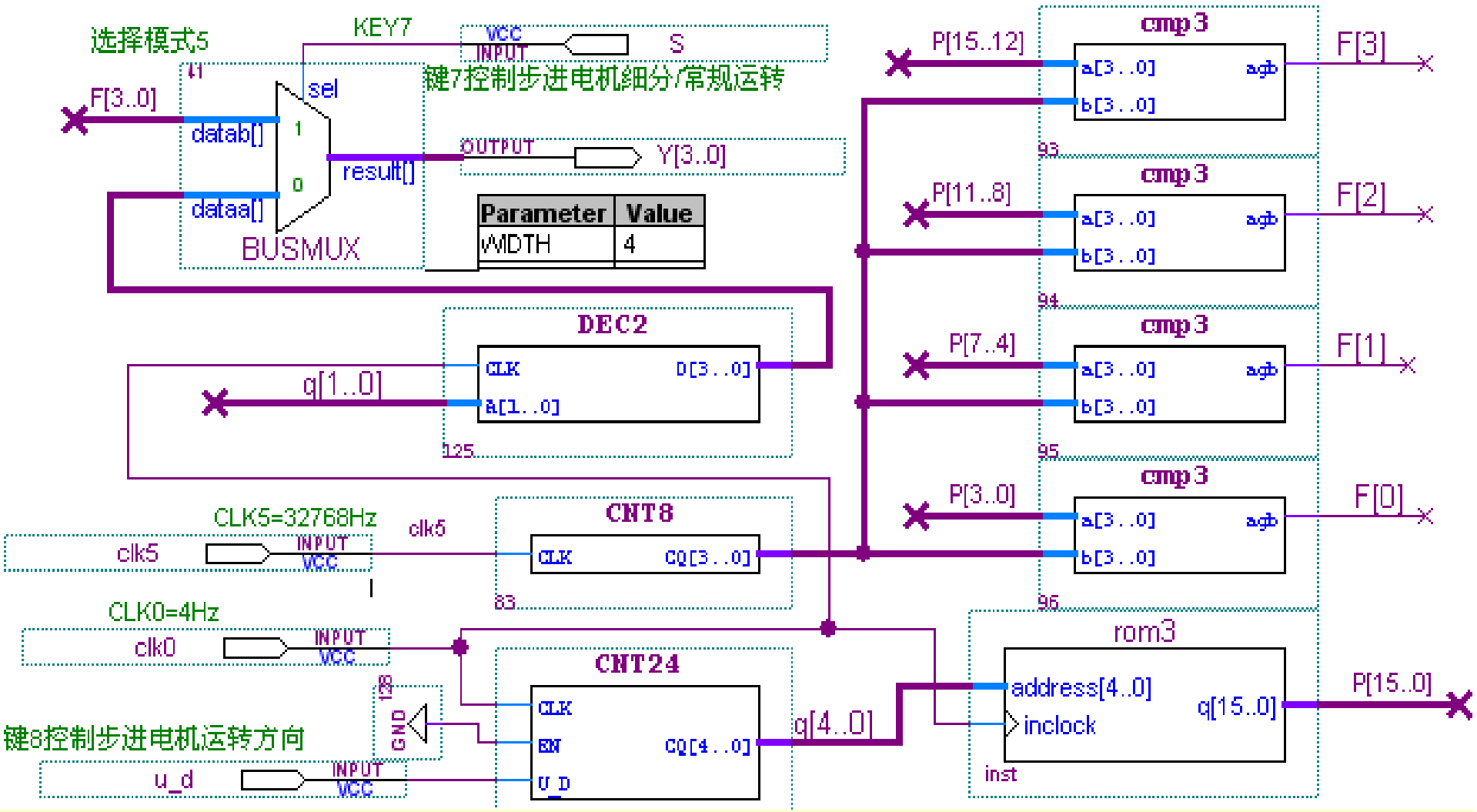


图13-7 步进电机PWM细分控制控制电路图

KX 康芯科技

13.3 步进电机细分驱动控制

2、步距细分的系统构成

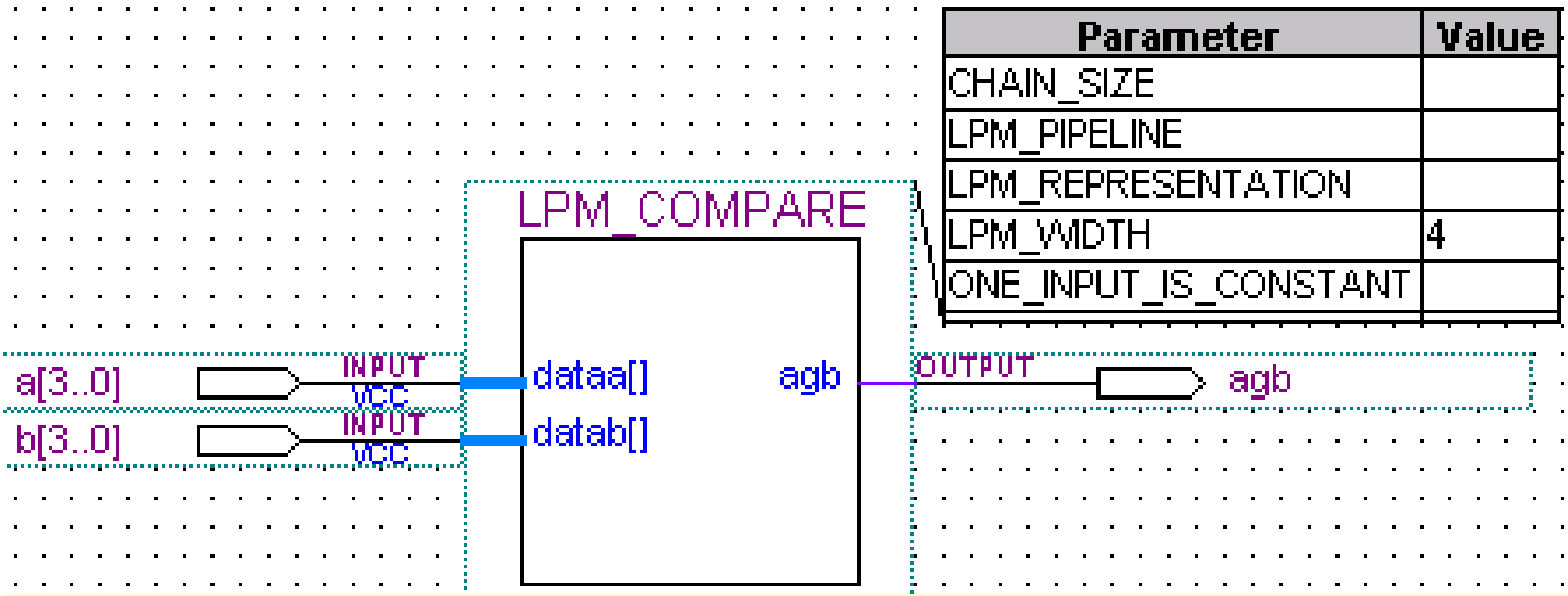


图13-8 图13-7中的cmp3模块

13.3 步进电机细分驱动控制

2、步距细分的系统构成

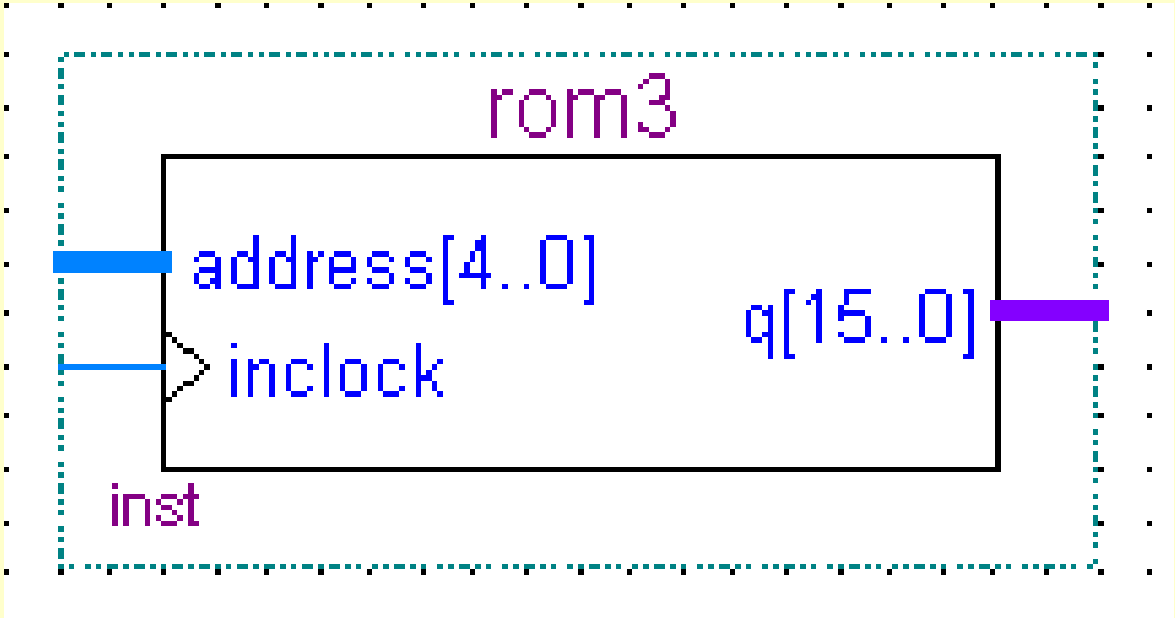


图13-9 PWM波形ROM存储器

13.3 步进电机细分驱动控制

3、细分电流信号的实现

4、细分驱动性能的改善

5、细工作时序分析

13.3 步进电机细分驱动控制

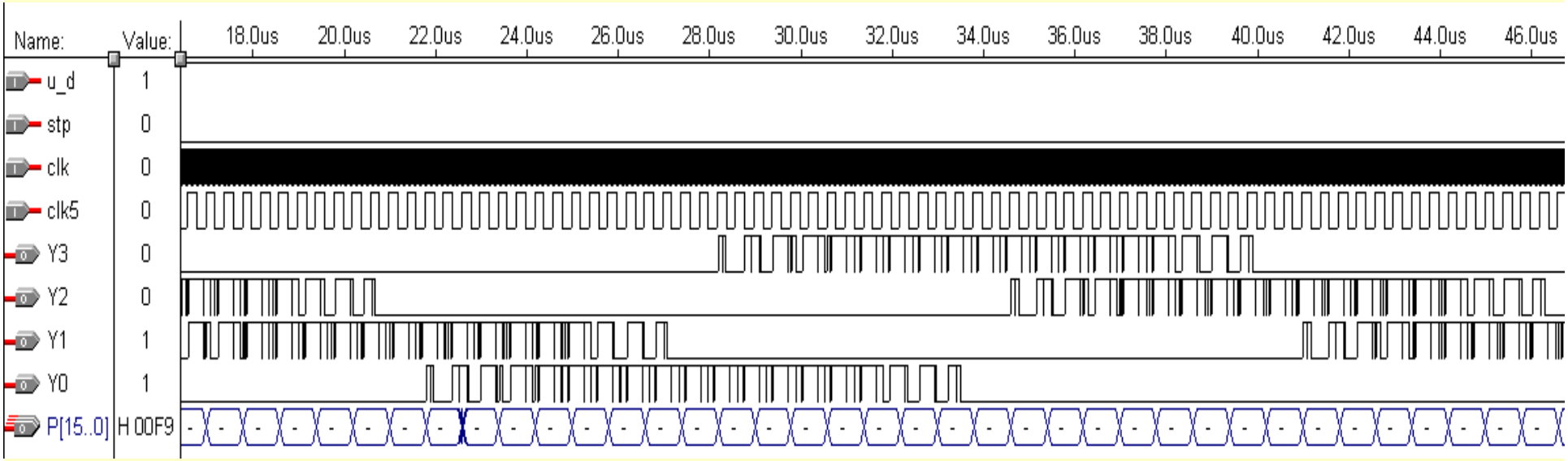


图13-10 步进电机PWM仿真波形图（注意，图中clk与clk5交换）

13.3 步进电机细分驱动控制

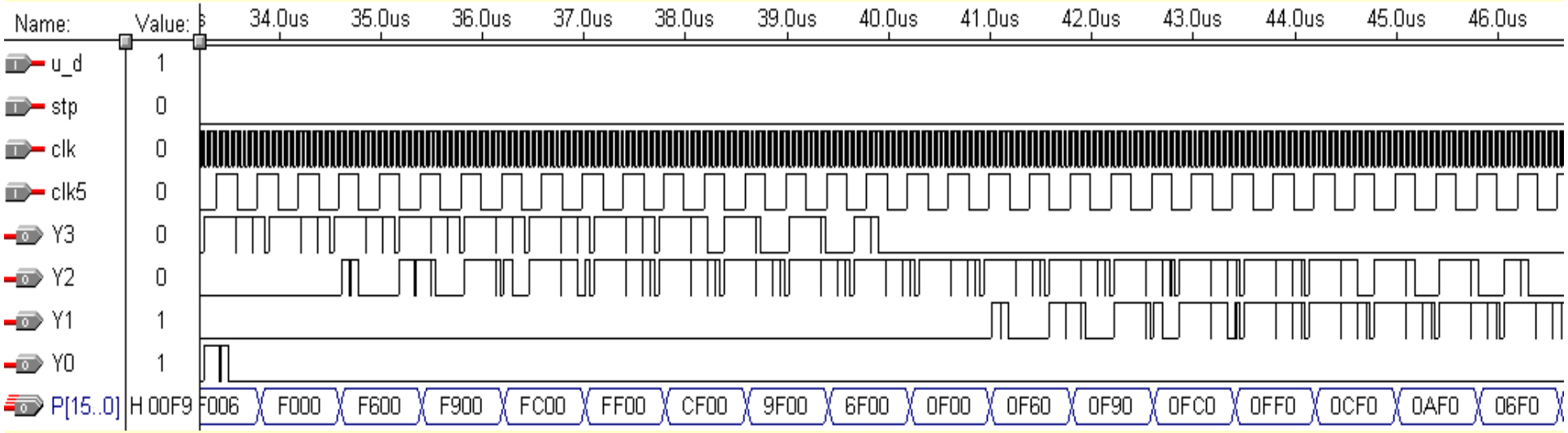


图13-11 展开后的步进电机PWM仿真波形图（注意，图中clk与clk5交换）

13.3 步进电机细分驱动控制

6、硬件验证

【例13-4】--元件CNT8

```
LIBRARY IEEE;                                                    -- 8进制计数器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT8 IS
    PORT ( CLK : IN STD_LOGIC;
           CQ : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END CNT8;
ARCHITECTURE behav OF CNT8 IS
    SIGNAL CQI : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
    PROCESS(CLK)
    BEGIN
       IF CLK'EVENT AND CLK = '1' THEN  CQI <= CQI + 1;  END IF;
    END PROCESS;
  CQ <= CQI(4 DOWNTO 1);
END behav;
```

【例13-5】--元件DEC2

```
LIBRARY IEEE ;
 USE IEEE.STD_LOGIC_1164.ALL ;
  ENTITY Dec2 IS
    PORT ( CLK : IN STD_LOGIC;
           A  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           D  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)  ) ;
 END ;
 ARCHITECTURE one OF Dec2 IS
 SIGNAL CQ : STD_LOGIC_VECTOR(1 DOWNTO 0);
 BEGIN
   PROCESS( CQ )
    BEGIN
        CASE  CQ  IS
          WHEN "00" =>  D <= "1001" ;
          WHEN "01" =>  D <= "1100" ;
          WHEN "10" =>  D <= "0110" ;
          WHEN "11" =>  D <= "0011" ;
          WHEN OTHERS =>  NULL ;
        END CASE ;
    END PROCESS ;
 PROCESS(CLK)
    BEGIN
       IF CLK'EVENT AND CLK = '1' THEN CQ <= A; END IF;
    END PROCESS;
END ;
```

【例13-6】 --元件CNT24

```
LIBRARY IEEE;                                          -- 24进制计数器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT24 IS
    PORT ( CLK,EN, U_D : IN STD_LOGIC;
              CQ : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END CNT24;
ARCHITECTURE behav OF CNT24 IS
    SIGNAL CQI : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
    PROCESS(CLK, EN, U_D)
    BEGIN
        IF EN = '1' THEN   CQI <= CQI;
        ELSIF CLK'EVENT AND CLK = '1' THEN
            IF U_D = '1' THEN  CQI <= CQI + 1;
                          ELSE CQI<= CQI-1;  END IF;
        END IF;
    END PROCESS;
  CQ(4 DOWNTO 0) <= CQI;
END behav;
```

13.3 步进电机细分驱动控制

6、硬件验证

【例13-7】 --5位地址线ROM3中的数据: pwm_1.mif，注意，实用每一组数据要占一行

```
WIDTH = 16;
DEPTH = 32;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
0:f000; 1:f600; 2:f900; 3:fc00; 4:ff00; 5:cf00; 6:9f00; 7:6f00; 8:0f00;
9:0f60; a:0f90; b:0fc0; c:0ff0; d:0cf0; e:0af0; f:06f0; 10:00f0; 11:00f6;
13:00f9;13:00fc; 14:00ff; 15:00cf; 16:009f; 17:006f; 18:000f; 19:600f;
1a:900f;1b:c00f; 1c:f00f; 1d:f00c; 1e:f009; 1f:f006;
END;
```

13.4 直流电机的PWM控制

直流电机控制电路主要由三部分组成：

- FPGA中PWM脉宽调制信号产生电路
- FPGA中的工作/停止控制和正/反转方向控制电路
- 由功率放大电路和H桥组成的正反转功率驱动电路

13.4 直流电机的PWM控制

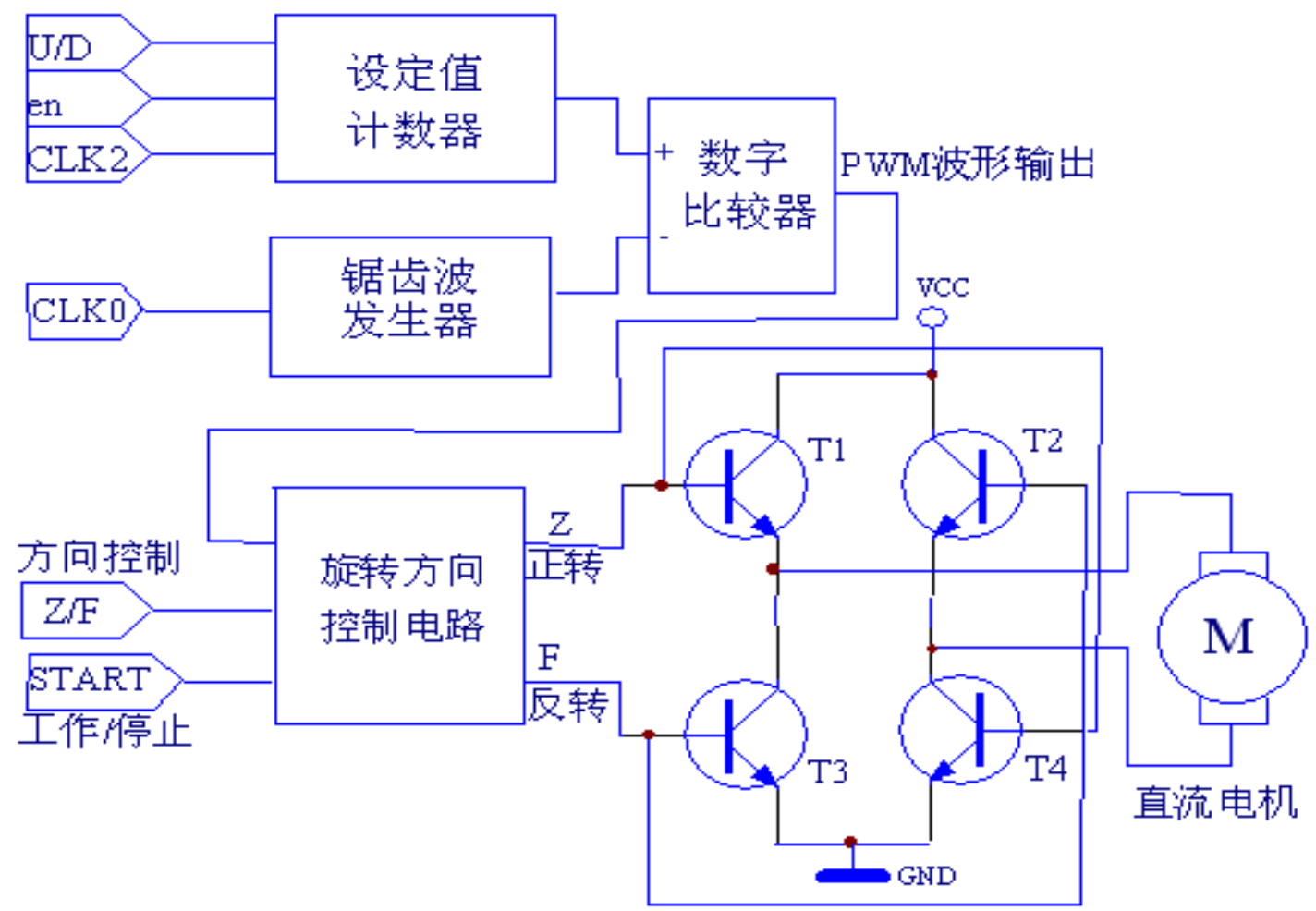


图13-12 FPGA直流电机驱动控制电路

13.4 直流电机的PWM控制

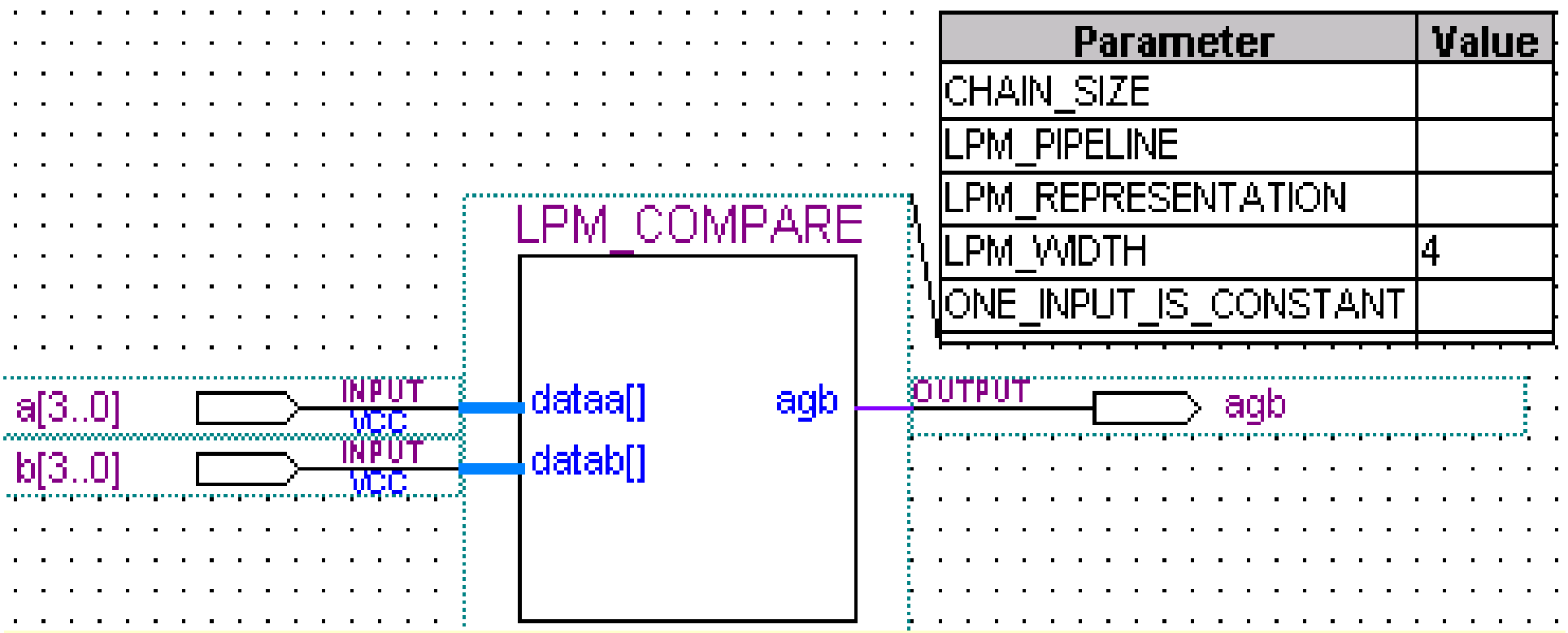


图13-13图13-14中的cmp3模块

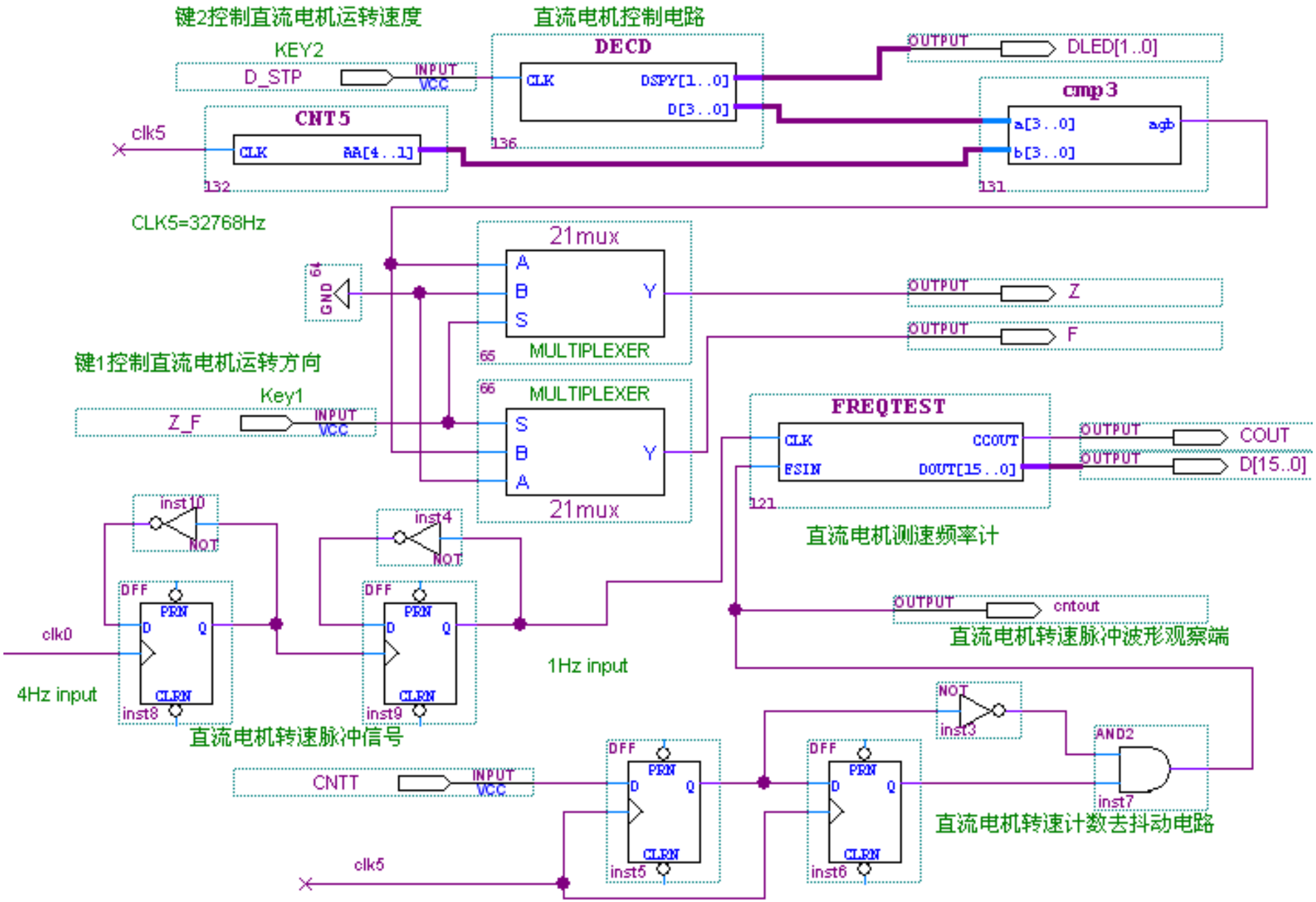


图13-14 FPGA直流电机控制模块

KX 康芯科技

【例13-8】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
  ENTITY DECD IS
    PORT ( CLK : IN STD_LOGIC;
         DSPY  : OUT STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            D  : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)  ) ;
 END ;
 ARCHITECTURE one OF DECD IS
 SIGNAL CQ : STD_LOGIC_VECTOR(1 DOWNTO 0);
 BEGIN
   PROCESS( CQ )
    BEGIN
        CASE  CQ  IS
          WHEN "00" =>  D <= "0100" ;
          WHEN "01" =>  D <= "0111" ;
          WHEN "10" =>  D <= "1011" ;
          WHEN "11" =>  D <= "1111" ;
          WHEN OTHERS =>  NULL ;
        END CASE ;
    END PROCESS ;
 PROCESS(CLK)
    BEGIN
       IF CLK'EVENT AND CLK = '1'  then CQ <= CQ + 1; END IF;
    END PROCESS;
   DSPY<=CQ;
END ;
```

13.4 直流电机的PWM控制

【例13-9】

```
LIBRARY IEEE;                                                    -- 4进制计数器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT5 IS
    PORT ( CLK : IN STD_LOGIC;
           AA : OUT STD_LOGIC_VECTOR(4 DOWNTO 1));
END CNT5;
ARCHITECTURE behav OF CNT5 IS
    SIGNAL CQI : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
    PROCESS(CLK)
    BEGIN
       IF CLK'EVENT AND CLK = '1'  then  CQI <= CQI + 1; END IF;
    END PROCESS;
  AA <= CQI(4 DOWNTO 1);
END behav;
```

习 题

13-1. 详述**VGA**显示控制原理。

13-2. 试在通用异步收发器**UART**中加入**FIFO**，以缓冲接收，发生数据。

13-3. 把**VGA**控制器模块与**UART**模块连接起来，实现**VGA**显示图像的动态更新。

13-4. 简述步进电机转角细分的工作原理，有哪些方法可以实现步进转角细分控制？

13-5. 步进电机相电流的细分与步进转角细分是一回事吗？有何区别？要提高步进电机转角细分的控制精度，可以采取哪些方法？

习 题

13-6. 要使步进电机按预先设定的角度转动，控制电路应如何设计？

13-7. 有哪些方法可以对直流电机进行调速控制？如何用FPGA对直流电机进行调速控制？

13-8. 若要使电机转速设置更精确，可以采取哪些措施，控制电路应如何修改？

13-9. 要使直流电机精确地达到设定转速，可以通过检测电机的转速，采用速度闭环控制。如何通过实验台上的光电检测装置检测电机转速，如何用FPGA实现速度闭环控制？

实 验 与 设 计

13-1. VGA彩条信号显示控制器设计

(1) 实验目的：学习VGA图像显示控制器的设计。

(2) 实验内容1：根据图13-3和程序13-1，完成VGA彩条信号显示的验证性实验。根据图13-3引脚锁定：R、G、B分别接PIO60、PIO61、PIO63；HS、VS分别接PIO64、PIO65；CLK接clock9（13MHz），MD接PIO0，控制显示模式。

接上VGA显示器，选择模式5，下载COLOR.SOF；控制键1，观察显示器工作（如果显示不正常，将GW48系统右侧开关拨以下，最后再拨回到“TO_MCU”）。

(3) 实验内容2：设计可显示横彩条与棋盘格相间的VGA彩条信号发生器。

(4) 实验内容3：设计可显示英语字母的VGA信号发生器电路。

(5) 实验内容4：设计可显示移动彩色斑点的VGA信号发生器电路。

实 验 与 设 计

13-2. VGA图像显示控制器设计

(1) 实验内容1：根据图13-4和程序例13-2/3，设计与生成图象数据；根据例13-2中imgrom元件的接口，定制放置图象数据的ROM。

(2) 实验内容2：硬件验证例13-2/3，选择模式5，引脚连接方式仍同图13-4，只是时钟输入clk50MHz接clock0，选择频率50MHz的时钟信号。在EDA系统上接上VGA显示器，下载后观察图形显示情况。

(3) 实验内容3：为此设计增加一个键，控制输出图象的正色与补色。

(4) 实验内容4：为了显示更大的图象，用外部ROM取代FPGA的内部ROM，即imgrom元件，电路结构参考图13-4，引脚锁定参考电路结构图NO.5图中的ROM 27C020/27C040与FPGA的引脚连接情况。示例程序可下载 ./VGA88/vgarom.sof，或 ./VGAbb/vgarom.sof，clock0接50MHz，GW48 EDA系统左下角的拨码开关的“ROM使能”拨向下（如果显示不正常，将EDA系统右侧开关拨以下，最后再拨回到“TO_MCU”）。选择模式5，键1控制图象的正色与补色显示。注意，实验结束后将拨码开关的“ROM使能”拨向上还原。

实 验 与 设 计

13-3. 步进电机细分驱动控制实验

(1) 实验目的：学习用FPGA实现步进电机的驱动和细分控制。

(2) 实验内容1：完成以图13-7所示的步进电机控制电路的验证性实验。首先引脚锁定：

步进电机的4个相：Ap、Bp、Cp、Dp（对应程序中的Y0、Y1、Y2、Y3）分别与PIO65、PIO64、PIO63、PIO62（见GW48主系统左侧的标注）相接。

CLK0接clock0，选择4Hz；CLK5接clock5，选择32768Hz；S接PIO6（键7），控制步进电机细分旋转（1/8细分，2.25度/步），或不细分旋转（18度/步）；U_D接PIO7（键8），控制旋转方向。

用短路帽将系统左侧的“步进允许(JM0)”短路（注意，电机实验结束后，短路帽插回“禁止”端！

选择模式No.5，用Quartus下载step_1c3中的step_a.sof到EP1C3中，观察电机工作情况。

给出电机的驱动仿真波形，与示波器中观察到的电机控制波形进行比较。

实 验 与 设 计

13-3. 步进电机细分驱动控制实验

(3) 实验内容2： 设计2个电路：1、要求能按给定细分要求，采用PWM方法，用FPGA对步进电机转角进行细分控制(利用QuartusII的EAB在系统编辑器实时在系统编辑调试ROM3中的细分控制数据)；2、用FPGA实现对步进电机的匀加速和匀减速控制。

(4) 实验内容3： 为使步进电机能平稳地运行，并尽快从起点到达终点，步进电机应按照以下控制方式运行：启动→匀加速→匀速→匀减速→停止。当给定终点位置（转角）以后，试用FPGA实现此控制。

(5) 实验内容4： 步进电机在步距角 8 细分的基础上，试通过修改控制电路对步距角进一步细分。

实 验 与 设 计

13-4. 直流电机PWM控制实验

(1) 实验目的：学习直流电机PWM的FPGA控制。掌握PWM控制的工作原理，对直流电机进行速度控制、旋转方向控制、变速控制。

(2) 实验内容1：完成以图13-14所示的直流电机控制电路的验证性实验。首先引脚锁定：

直流电机模块中的MA2、MA1（对应程序中的Z、F）分别与EP1C3的PIO60/61相接，用于控制直流电机；测直流电机转速的MA-CNT端接PIO66，即CNTT端（见主系统左侧的标注）；

用短路帽分别将主系统左侧的“直流允许（JM1）”和“计数允许（JM2）”短路；CLK5接clock5，选择32768Hz；F1HZ接clock2，选择1Hz，作为转速测量的频率计的门控时钟；

键1（PIO0，接Z_F）控制旋转方向；键2（PIO1，D_STP）控制旋转速度。连续按动此键时，由数码管7显示0、1、2、3指示4个速度级别；转速由数码管4、3、2、1显示。

选择模式No.5，用Quartus下载step_1c3中的step_a.sof到EP1C3中，观察电机工作情况。

给出电机的驱动仿真波形，与示波器中观察到的电机控制波形进行比较。

(3) 实验内容2：实现直流电机的闭环控制，旋转速度可设置。

(4) 实验内容3：说明图13-14中的去抖动电路的工作原理，为了加强去抖动效果，改进图13-14中的去抖动电路和工作时钟频率的选择，如设计一个含4个D触发器的去抖动电路，实测它的性能。并说明此电路的工作时钟频率与被测信号频率的关系。

EDA 技术实用教程

第 12 章

系统仿真

12.1 仿真

仿真也称模拟（**Simulation**）

是对电路设计的一种间接的检测方法，是利用计算机对整个硬件系统进行模拟检测，但却可以不接触具体的硬件系统。

12.2 VHDL源程序仿真

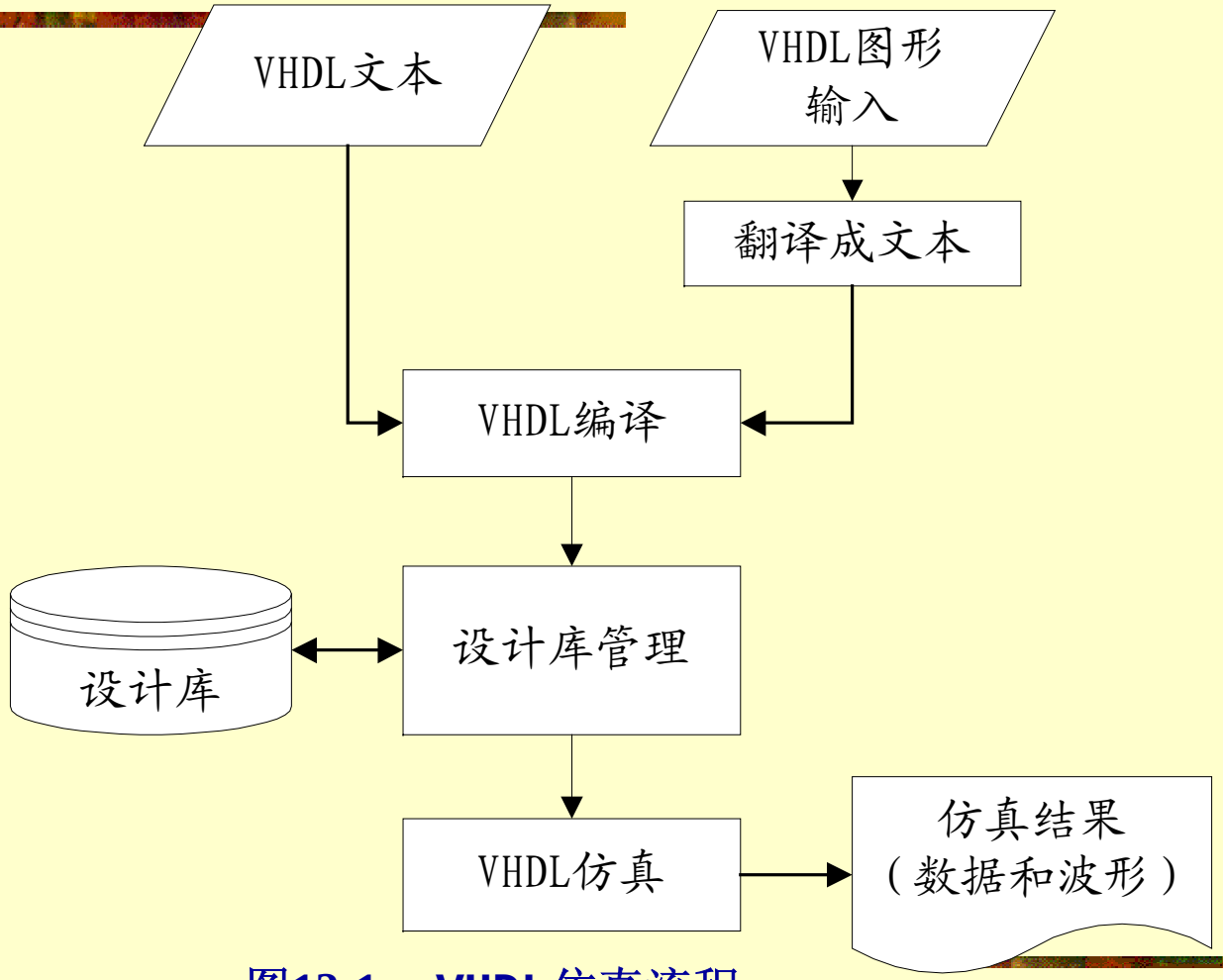


图12-1　VHDL仿真流程

12.2 VHDL源程序仿真

【例12-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY and1 IS
PORT(aaa,bbb : IN STD_LOGIC; ccc: OUT STD_LOGIC);
END and1;
ARCHITECTURE one OF and1 IS
BEGIN
ccc <= aaa AND bbb;
END;
```

12.2 VHDL源程序仿真

【例12-2】

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY TRIBUF_and1 IS
      GENERIC (
          ttri: TIME := 1 ns;
          ttxz: TIME := 1 ns;
          ttzx: TIME := 1 ns);
      PORT (
          in1 : IN std_logic;
          oe  : IN std_logic;
          y   : OUT std_logic);
END TRIBUF_and1;
ARCHITECTURE behavior OF TRIBUF_and1 IS
BEGIN
      PROCESS (in1, oe)
  BEGIN
     IF oe'EVENT THEN
```

（接下页）

```
IF oe = '0' THEN
                    y <= TRANSPORT 'Z' AFTER ttxz;
                ELSIF oe = '1' THEN
                    y <= TRANSPORT in1 AFTER ttzx;
                END IF;
            ELSIF oe = '1' THEN
                y <= TRANSPORT in1 AFTER ttri;
            ELSIF oe = '0' THEN
                y <= TRANSPORT 'Z' AFTER ttxz;
            END IF;
        END PROCESS;
END behavior;
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.tribuf_and1;
ENTITY and1 IS
    PORT (
      aaa : IN std_logic;
      bbb : IN std_logic;
      ccc : OUT std_logic);
END and1;
ARCHITECTURE EPF10K10LC84_a3 OF and1 IS
  . . . . . .
END EPF10K10LC84_a3;
```

12.3 仿真激励信号的产生

第一种方法 ：

【例12-3】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ADDER4 IS
    PORT ( a, b : IN INTEGER RANGE 0 TO 15;
                  c : OUT INTEGER RANGE 0 TO 15 );
END ADDER4;
ARCHITECTURE one OF ADDER4 IS
BEGIN
    c <= a + b;
END one;
```

12.3 仿真激励信号的产生

【例12-4】

```
ENTITY SIGGEN IS
    PORT ( sig1 : OUT INTEGER RANGE 0 TO 15;
           sig2 : OUT INTEGER RANGE 0 TO 15 );
END;
ARCHITECTURE Sim OF SIGGEN IS
BEGIN
    sig1 <= 10, 5 AFTER 200 ns, 8 AFTER 400 ns;
    sig2 <= 3, 4 AFTER 100 ns, 6 AFTER 300 ns;
END;
```

12.3 仿真激励信号的产生

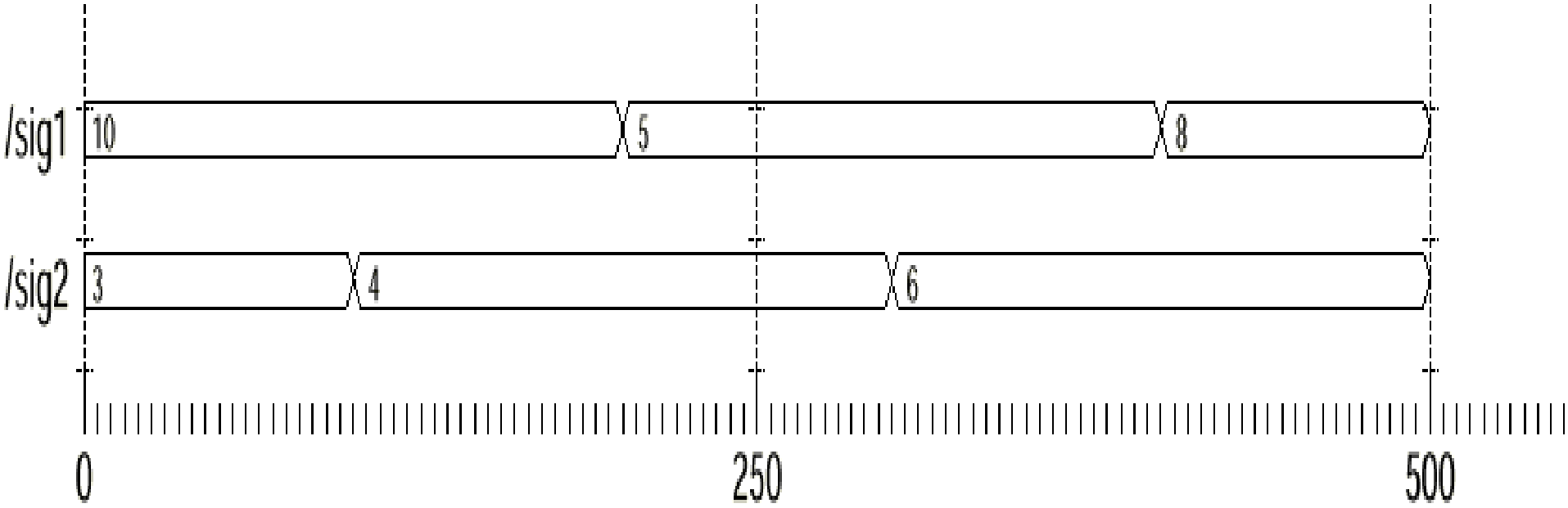


图12-2 SIGGEN的仿真输出波形

12.3 仿真激励信号的产生

【例12-5】

```
ENTITY BENCH IS
END;
ARCHITECTURE one OF BENCH IS
    COMPONENT ADDER4
        PORT ( a, b : integer range 0 to 15;
                  c : OUT INTEGER RANGE 0 TO 15 );
    END COMPONENT;
    COMPONENT SIGGEN
        PORT ( sig1 : OUT INTEGER RANGE 0 TO 15;
                 sig2 : OUT INTEGER RANGE 0 TO 15 );
    END COMPONENT;
        SIGNAL a, b, c : INTEGER RANGE 0 TO 15;
BEGIN
    U1 : ADDER4 PORT MAP (a, b, c);
    U2 : SIGGEN PORT MAP (sig1=>a, sig2=>b);
END;
```

12.3 仿真激励信号的产生

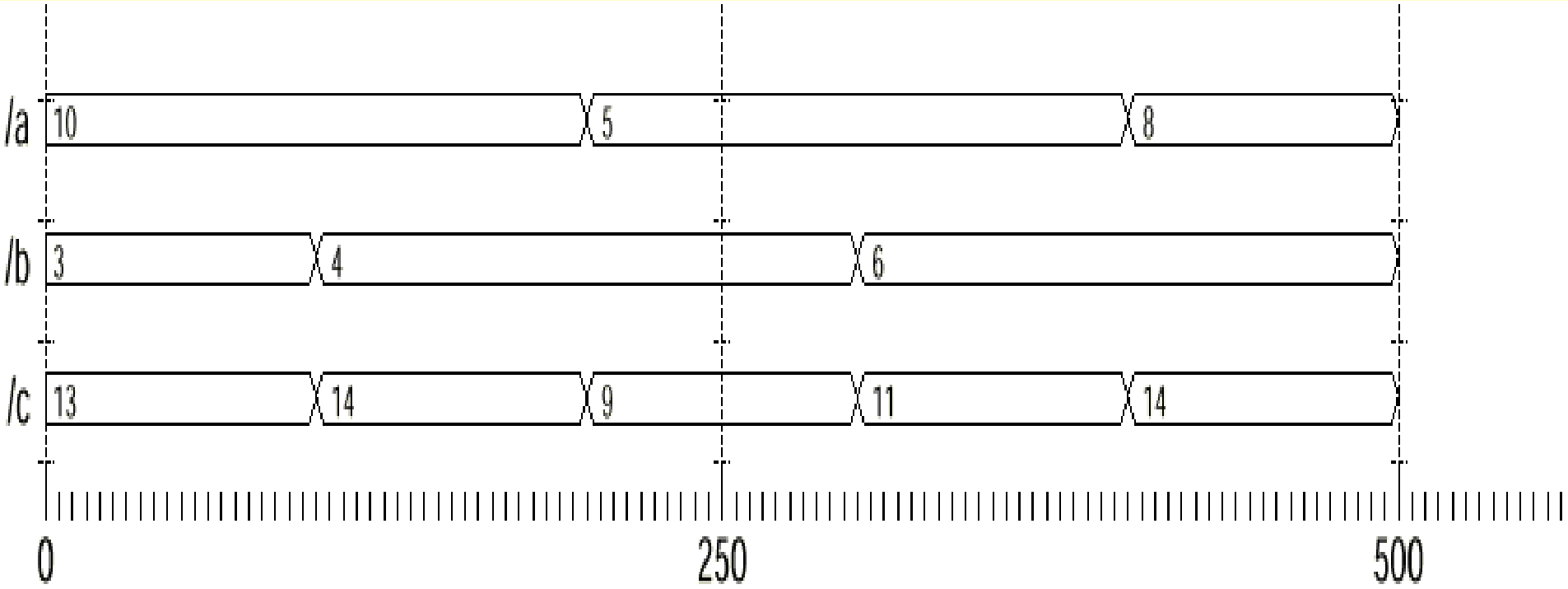


图12-3 BENCH仿真波形图

12.3 仿真激励信号的产生

第二种方法 ：

```
force <信号名> <值> [<时间>][, <值> <时间> …] [-repeat <周期>]
```

```
force a 0                          （强制信号的当前值为0）
force b 0 0, 1 10          （强制信号b在时刻0的值为0，在时刻10的值为1）
  force clk 0 0, 1 15 –repeat 20  （clk为周期信号，周期为20
```

```
force a 10 0, 5 200, 8 400
force b 3 0, 4 100, 6 300
```

12.4 VHDL测试基准

【例12-6】

```
Library IEEE;
use IEEE.std_logic_1164.all;
entity counter8 is
  port (
          CLK, CE, LOAD, DIR, RESET: in STD_LOGIC;
          DIN: in INTEGER range 0 to 255;
          COUNT: out INTEGER range 0 to 255 );
end counter8;
architecture counter8_arch of counter8 is
begin
process (CLK, RESET)
   variable COUNTER: INTEGER range 0 to 255;
begin
   if RESET='1' then  COUNTER := 0;
   elsif CLK='1' and CLK'event then
      if LOAD='1' then  COUNTER := DIN;
```

（接下页）

```
Else
         if CE='1' then
            if DIR='1' then
             if COUNTER =255 then COUNTER := 0;
                Else   COUNTER := COUNTER + 1;
              end if;
            else
              if COUNTER =0 then  COUNTER := 255;
                Else   COUNTER := COUNTER - 1;
              end if;
            end if;
         end if;
      end if;
   end if;
   COUNT <= COUNTER;
end process;
end counter8_arch;
```

12.4 VHDL测试基准

【例12-7】

```
Entity testbench is end testbench;
Architecture testbench_arch of testbench is
File RESULTS: TEXT open WRITE_MODE is "results.txt";
Component counter8
   port (   CLK: in STD_LOGIC;
         RESET: in STD_LOGIC;
         CE, LOAD, DIR: in STD_LOGIC;
         DIN: in INTEGER range 0 to 255;
         COUNT: out INTEGER range 0 to 255 );
 end component;
shared variable end_sim : BOOLEAN := false;
signal CLK, RESET, CE, LOAD, DIR: STD_LOGIC;
signal DIN: INTEGER range 0 to 255;
signal COUNT: INTEGER range 0 to 255;
procedure WRITE_RESULTS  (
            CLK，CE，LOAD，LOAD，RESET : STD_LOGIC;
```

（接下页）

```
DIN, COUNT : INTEGER  ) is
Variable V_OUT : LINE;
Begin
     write(V_OUT, now, right, 16, ps);                    -- 输入时间
     write(V_OUT, CLK, right, 2);
     write(V_OUT, RESET, right, 2);
     write(V_OUT, CE, right, 2);
     write(V_OUT, LOAD, right, 2);
     write(V_OUT, DIR, right, 2);
     write(V_OUT, DIN, right, 257);
     --write outputs
     write(V_OUT, COUNT, right, 257);
     writeline(RESULTS,V_OUT);
end WRITE_RESULTS;
begin
     UUT: COUNTER8
     port map (CLK => CLK,RESET => RESET,
                  CE => CE,   LOAD => LOAD,
                   DIR => DIR,   DIN => DIN,
                COUNT => COUNT );
CLK_IN:     process
     Begin                                                （接下页）
```

```
if end_sim = false then  CLK <= '0';
                    Wait for 15 ns;
                    CLk <='1';
                    Wait for 15 ns;
               Else
                  Wait;
              end if;
     end process;
STIMULUS: process
     Begin
           RESET  <= '1';
           CE           <= '1';                -- 计数使能
           DIR          <= '1';                -- 加法计数
           DIN          <= 250;                -- 输入数据
           LOAD   <= '0';                      -- 禁止加载输入的数据
       wait for 15 ns;
       RESET      <= '0';
       wait for 1 us;
       CE         <= '0';            -- 禁止计数脉冲信号进入
       wait for 200 ns;
       CE         <= '1';
       wait for 200 ns;                              （接下页）
```

```
DIR         <= '0';
        wait for 500 ns;
        LOAD<= '1';
        wait for 60 ns;
        LOAD        <= '0';
        wait for 500 ns;
        DIN  <= 60;
        DIR  <= '1';
        LOAD<= '1';
        wait for 60 ns;
        LOAD<= '0';
        wait for 1 us;
        CE          <= '0';
        wait for 500 ns;
        CE          <= '1';
        wait for 500 ns;
        end_sim :=true;
     wait;
end process;
WRITE_TO_FILE: WRITE_RESULTS(CLK,RESET,CE,LOAD,DIR,DIN,COUNT);
End testbench_arch;
```

KX 康芯科技

12.4 VHDL测试基准

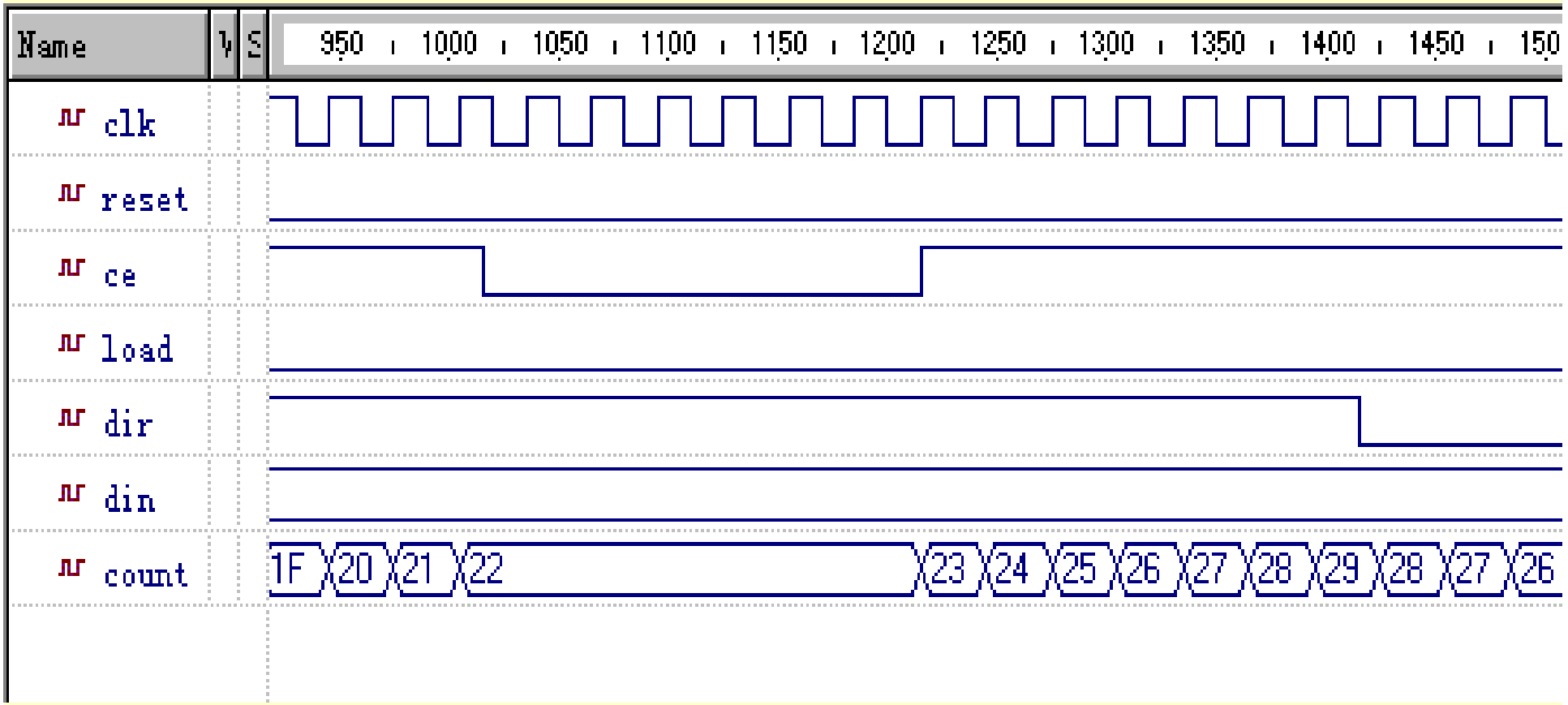


图12-4 8位计数器测试基准仿真部分波形图

12.5 VHDL系统级仿真

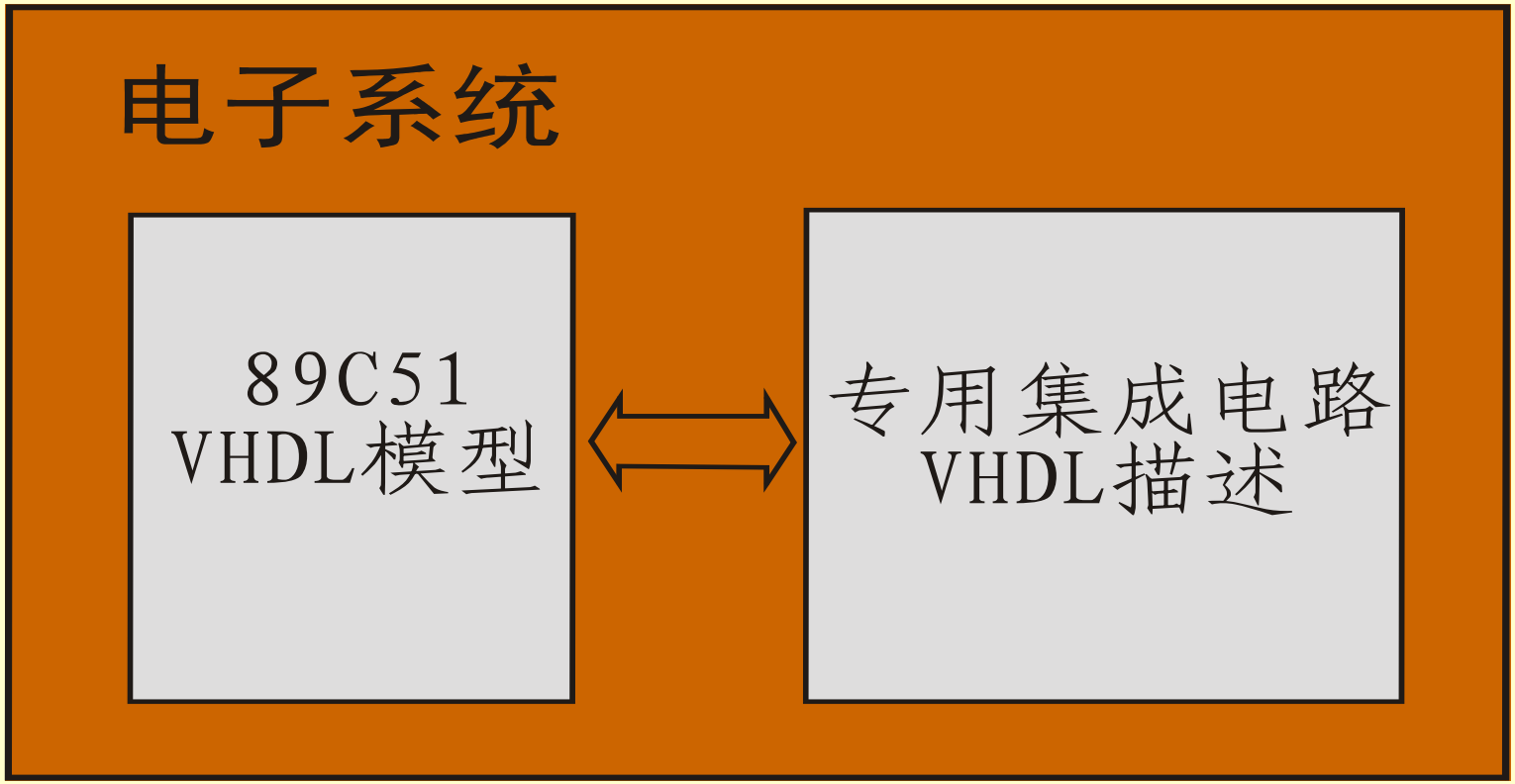


图12-5 VHDL系统仿真模型示意图

12.6 使用ModelSim进行仿真

【例12-8】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY cnt4 IS
    PORT
    (           rst : IN STD_LOGIC;
          d : IN STD_LOGIC_VECTOR(3 downto 0);
          load : IN STD_LOGIC;
          clk,ce : IN STD_LOGIC;
          q : OUT STD_LOGIC_VECTOR(3 downto 0);
          cout : OUT STD_LOGIC    );
END cnt4;
ARCHITECTURE syn OF cnt4 IS
    signal count : std_logic_vector(3 downto 0);
  BEGIN
```

（接下页）

```
cntproc: process(clk,rst) begin
     if rst = '1' then
            count <= (others => '0');
     elsif rising_edge(clk) then
            if ce = '1' then
                   if load = '1' then
                          count <= d;                 else
                          count <= count + 1;
                   end if;
            end if;
     end if;
end process;
coutproc : process(clk,rst) begin
     if rst = '1' then
            cout <= '0';
     elsif rising_edge(clk) then
            if count = "1111" then
                   cout <= '1';          else
                   cout <= '0';
            end if;
     end if;
end process;
q <= count;
END syn;
```

12.6 使用ModelSim进行仿真

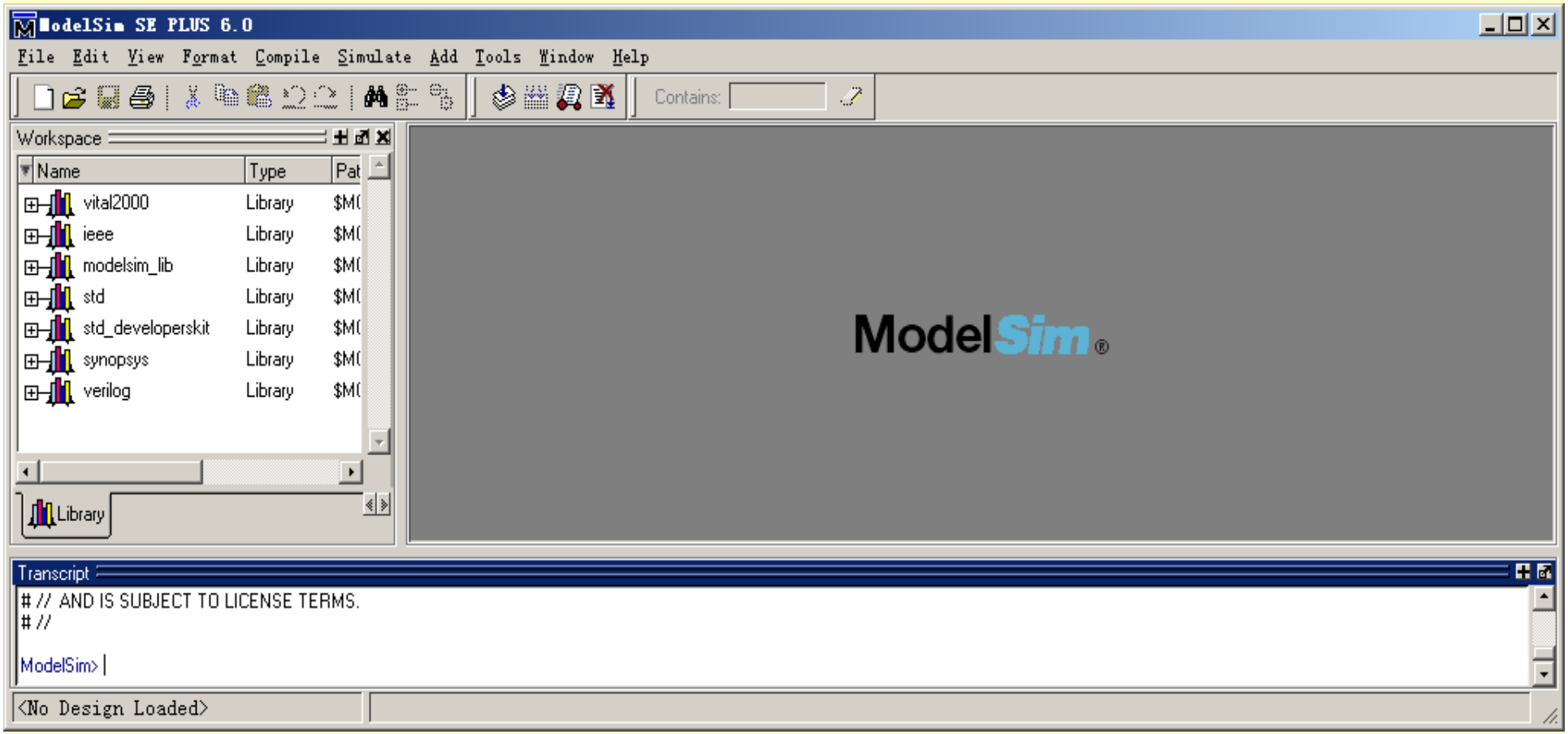


图12-6 ModelSim的启动界面

12.6 使用ModelSim进行仿真

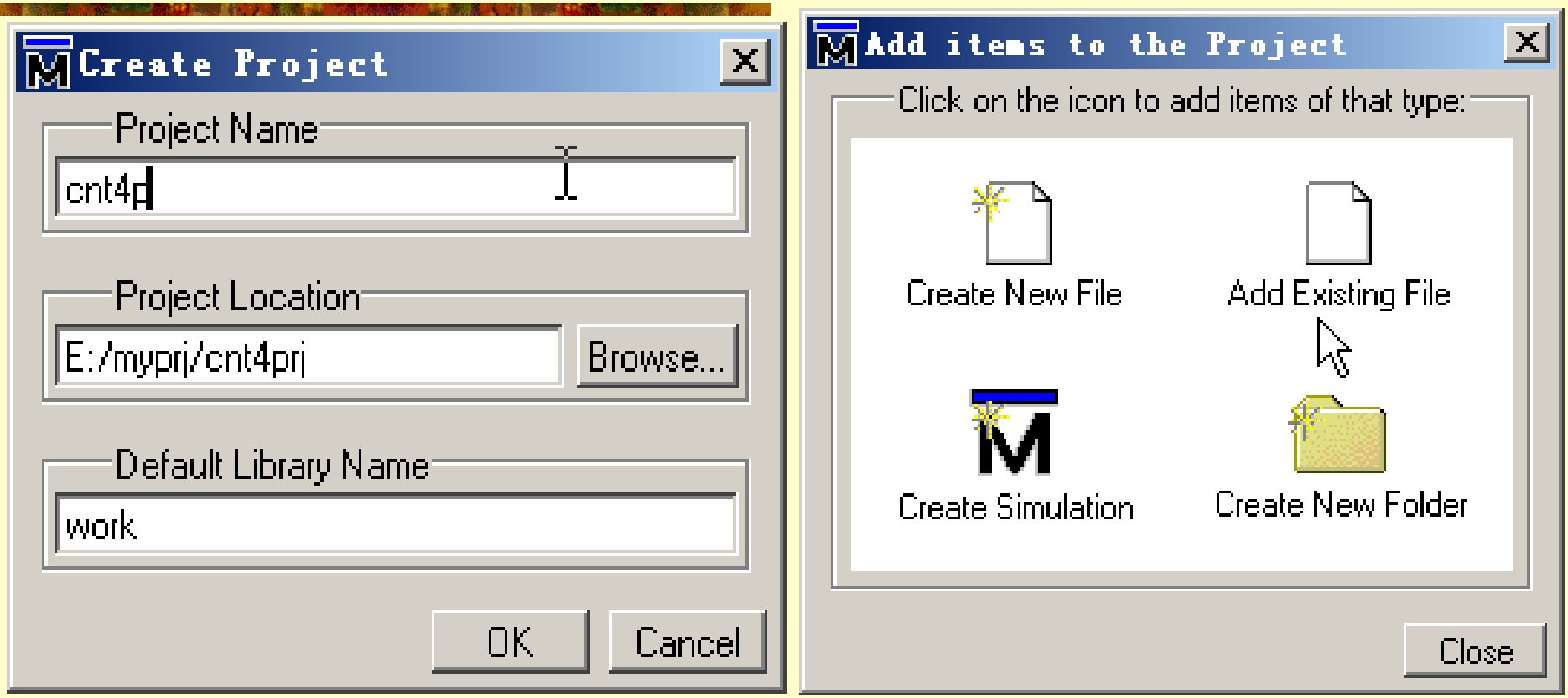


图12-7 建立工程建立项目

12.6 使用ModelSim进行仿真

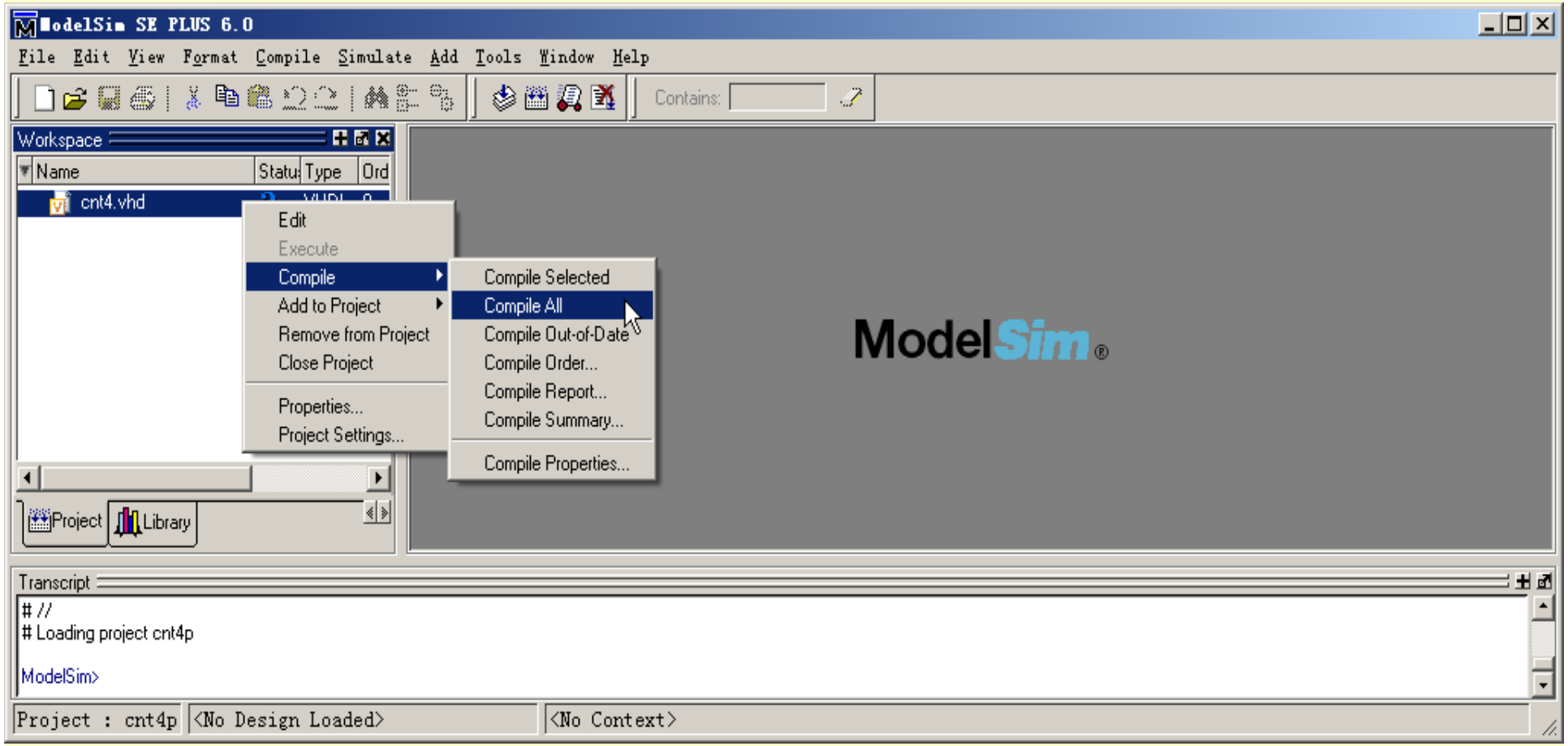


图12-8 开始编译仿真文件

12.6 使用ModelSim进行仿真

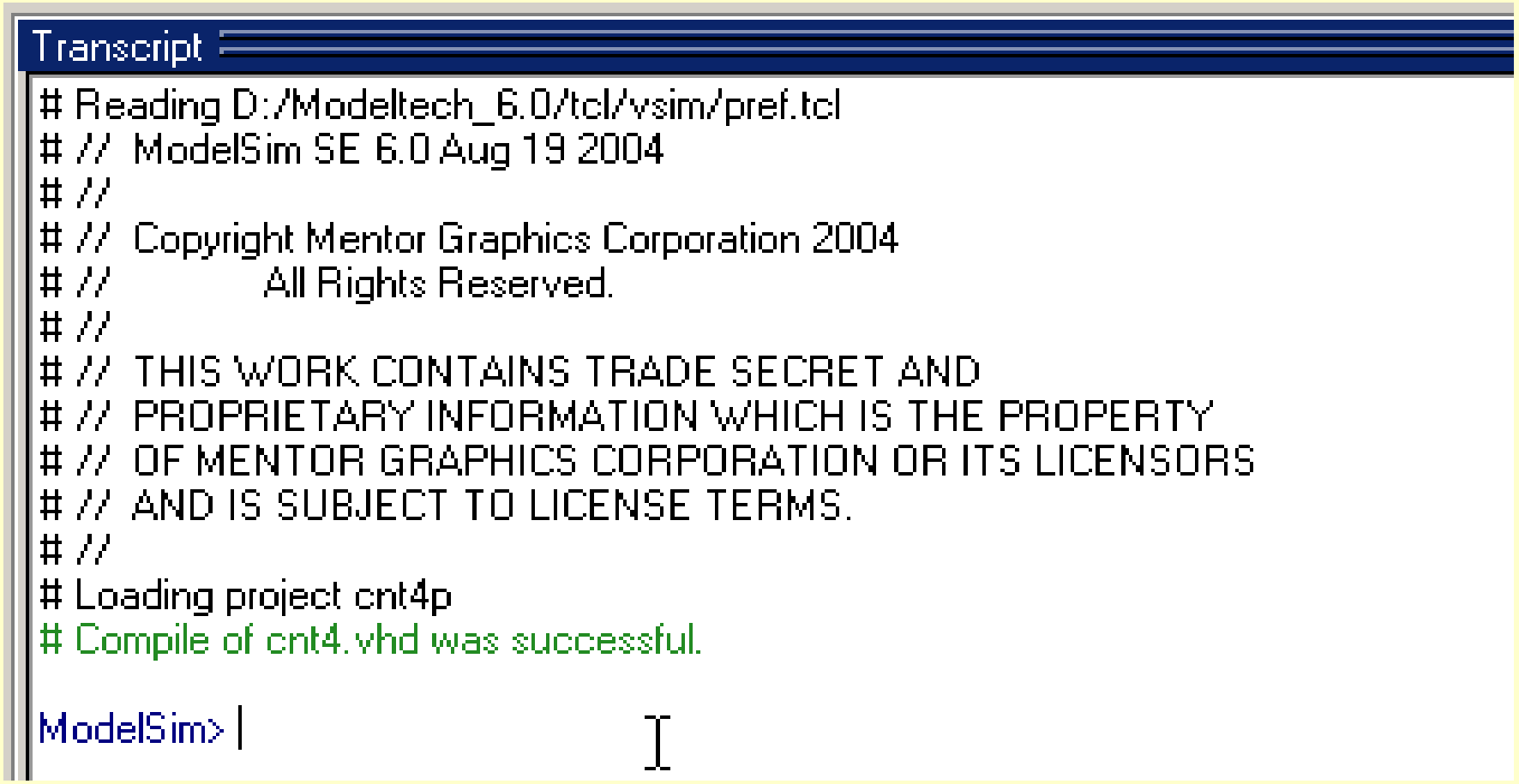


图12-9 ModelSim编译时的提示信息

12.6 使用ModelSim进行仿真

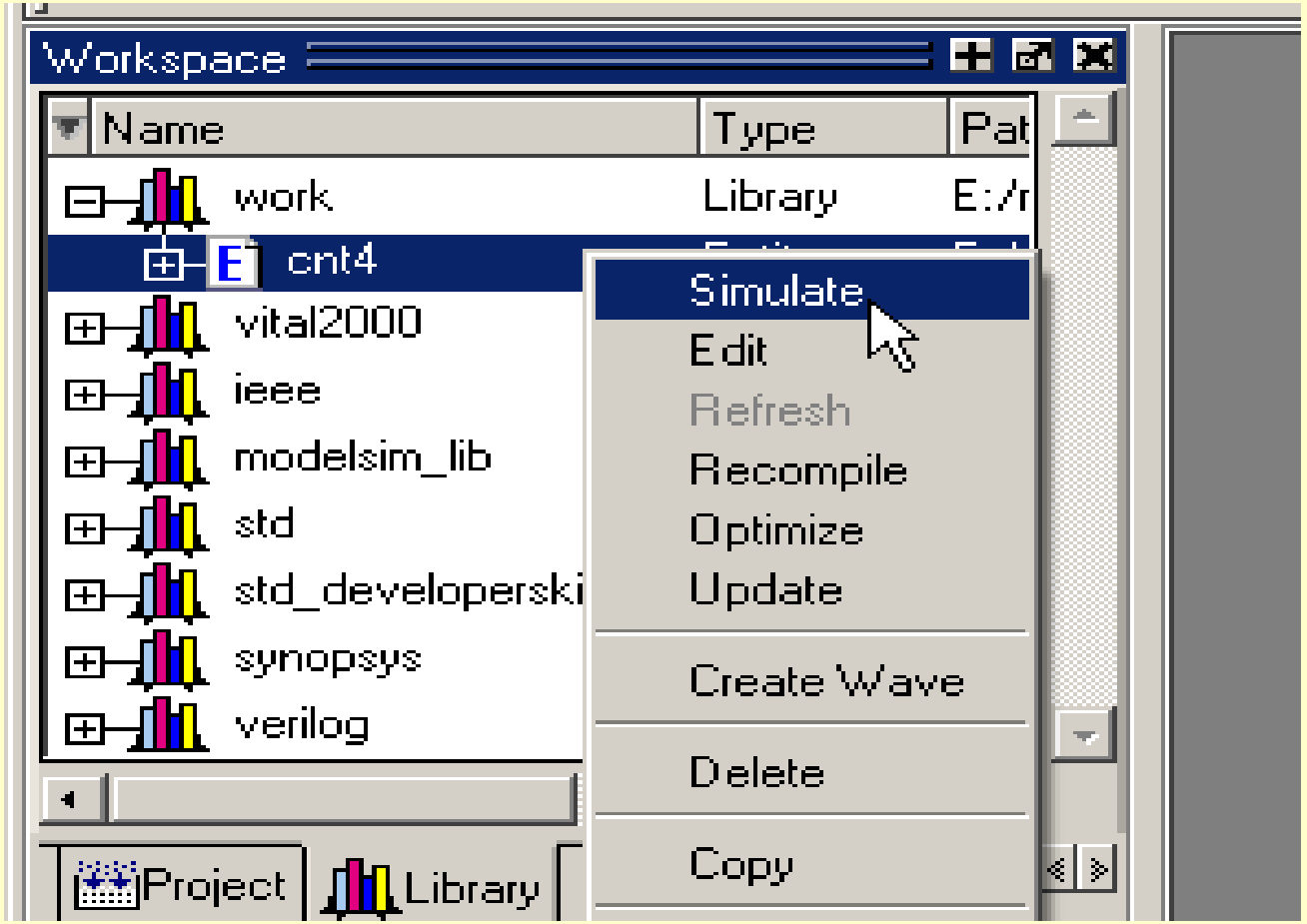


图12-10 装载设计模块

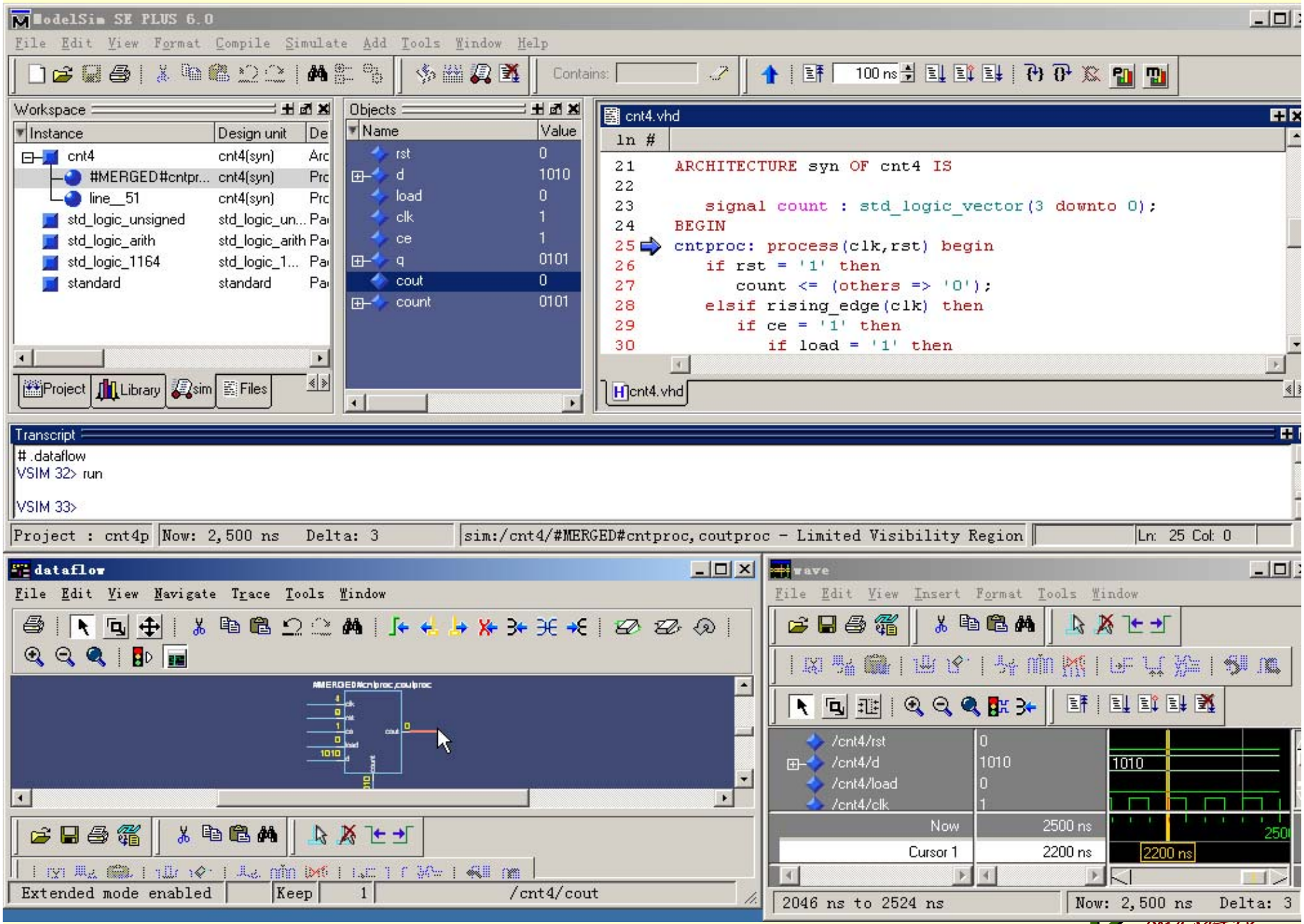


图12-11 ModelSim的仿真观察窗

【例12-9】

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY wavegen IS
     PORT
     (            clk,rst : OUT STD_LOGIC    );
end wavegen;
ARCHITECTURE sim OF wavegen is
     constant cycle:Time := 10 ns;
BEGIN
     process  begin
            clk <= '0';
            wait for cycle/2;
            clk <= '1';
            wait for cycle/2;
     end process;
     process  begin
            rst <= '1';
            wait for cycle*5;
            rst <= '0';
            wait;
     end process;
END sim;
```

12.6 使用ModelSim进行仿真

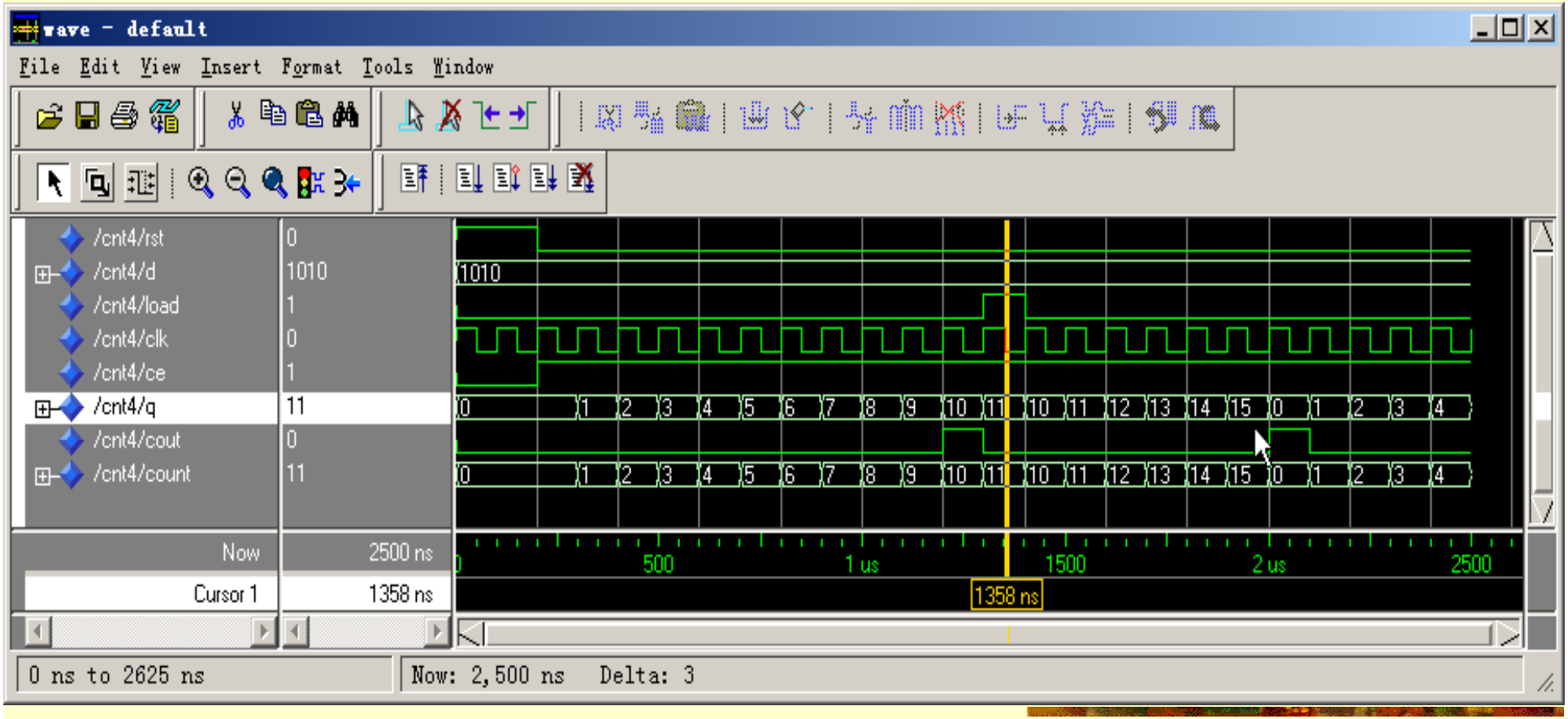


图12-12 ModelSim的波形观察窗

KX 康芯科技

12.6 使用ModelSim进行仿真

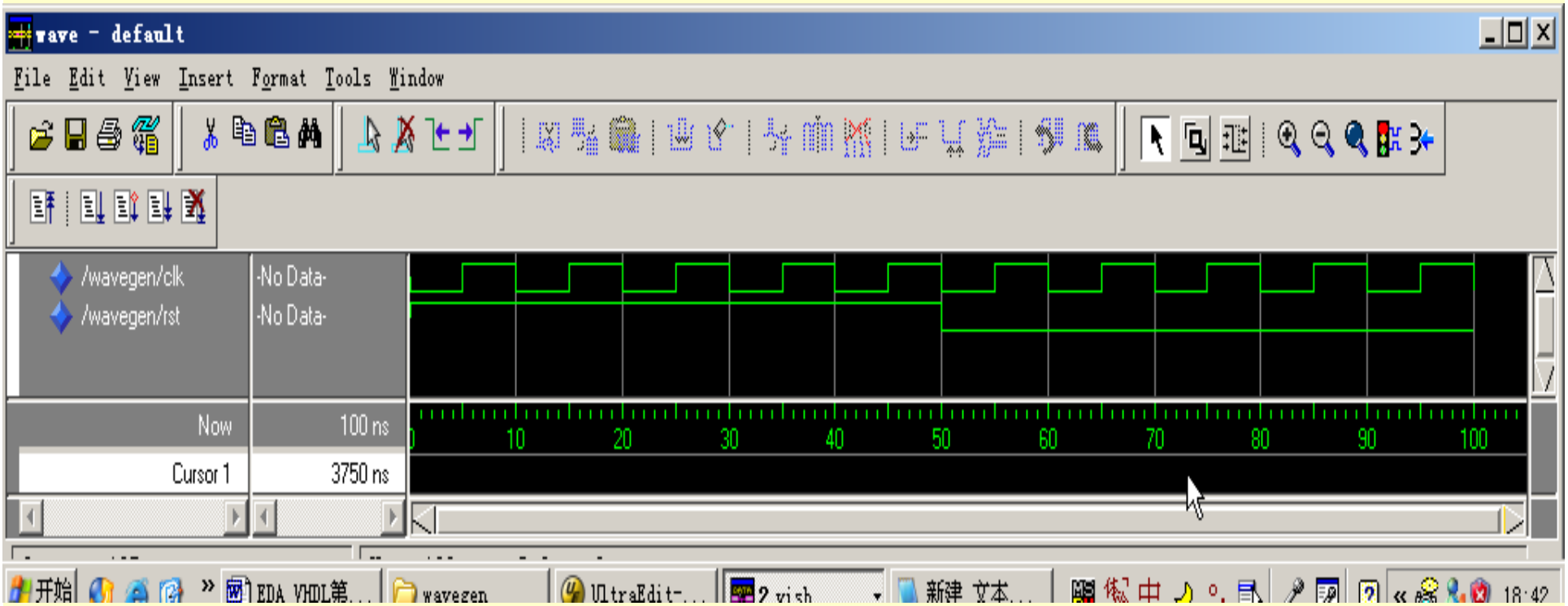


图12-12 时钟与复位信号生成

12.7 VHDL的RTL表述

12.7.1 行为描述

【例12-10】

```
LIBRARY IEEE;
    USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_UNSIGNED.ALL;
    ENTITY cunter_up IS
PORT(     reset, clock : IN  STD_LOGIC;
             counter : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)   );
    END;
    ARCHITECTURE behv of cunter_up IS
SIGNAL cnt_ff: UNSIGNED(7 DOWNTO 0);
    BEGIN
PROCESS (clock,reset,cnt_ff)
BEGIN
    IF reset='1' THEN   cnt_ff <= X"00" ;
     ELSIF (clock='1' AND clock'EVENT)
THEN   cnt_ff <= cnt_ff + 1 ;
    END IF;
END PROCESS;
    counter <= STD_LOGIC_VECTOR(cnt_ff);
    END ARCHITECTURE behv ;
```

12.7 VHDL的RTL表述

12.7.1 行为描述

【例12-11】

```
MODULE counter_up
  Clock ,reset,                    PIN ;
  Counter7..counter0               PIN  ISTYPE  'COM' ;
  Cnt_ff7..cnt_ff0                 NODE ISTYPE  'REG' ;
  Counter = [counter7..counter0];
  Cnt = [cnt_ff7..cnt_ff0];
 EQUATIONS
  Cnt.CLK = clock ;
  Cnt.AR  = reset    ;
  Cnt :=  cnt.FB + 1 ;
  Counter = cnt ;
 END counter_up
```

12.7 VHDL的RTL表述

12.7.2 数据流描述

12.7.3 结构描述

元件说明：描述局部接口。

元件例化：相对于其它元件放置元件。

元件配置：指定元件所用的设计实体。。

EDA 技术实用教程

第 11 章

优化和时序分析

11.1 资源优化

11.1.1 资源共享

【例11-1】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
ENTITY multmux IS
    PORT (A0, A1, B  : IN  std_logic_vector(3 downto 0);
                sel  : IN  std_logic;
             Result  : OUT std_logic_vector(7 downto 0));
END multmux;
ARCHITECTURE rtl OF multmux IS
BEGIN
    process(A0,A1,B,sel)
    begin
        if(sel = '0') then   Result <= A0 * B;
                      else   Result <= A1 * B;
        end if;
    end process;
END rtl;
```

KX 康芯科技

11.1 资源优化

11.1.1 资源共享

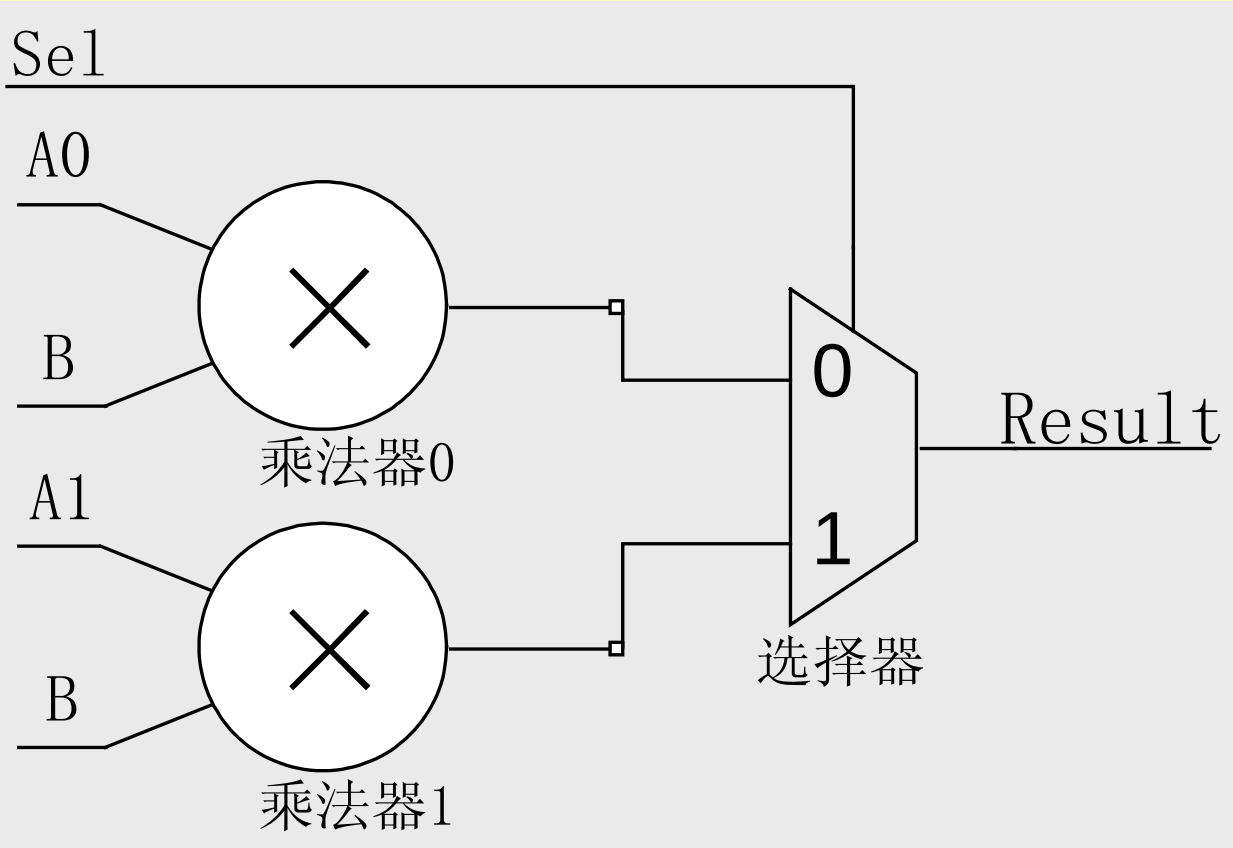


图11-1 先乘后选择的设计方法RTL结构

11.1 资源优化

11.1.1 资源共享

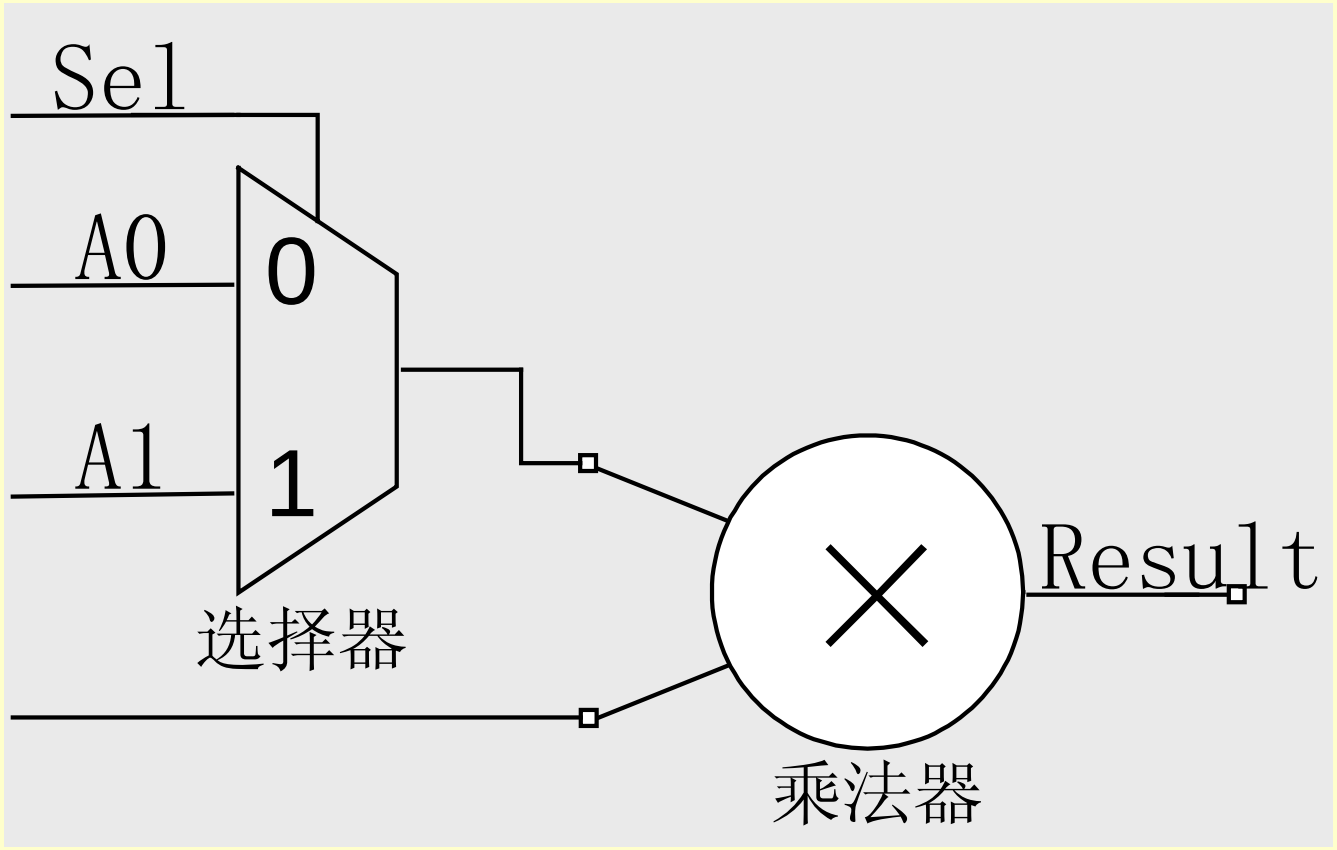


图11-2 先选择后乘设计方法RTL结构

11.1 资源优化

11.1.1 资源共享

【例11-2】

```
ARCHITECTURE rtl OF muxmult IS
    signal temp : std_logic_vector(3 downto 0);
BEGIN
    process(A0,A1,B,sel)
    begin
        if(sel = '0') then    temp <= A0; else  temp <= A1;
        end if;
        result <= temp * B;
    end process;
END rtl;
```

11.1 资源优化

11.1.1 资源共享

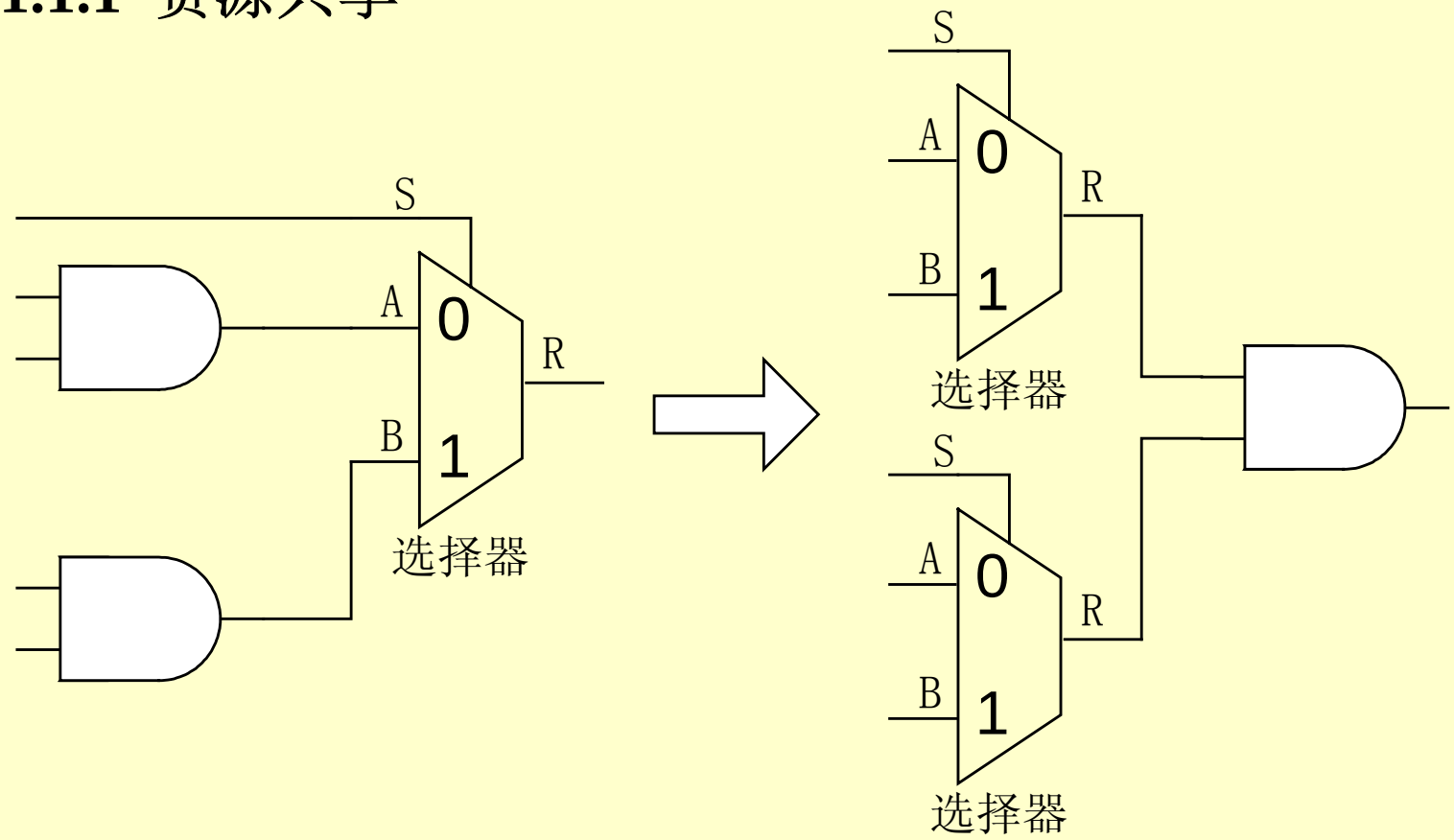


图11-3 资源共享反例

11.1 资源优化

11.1.2 逻辑优化

【例11-3】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY mult1 IS
    PORT(clk : in std_logic;
           ma : In std_logic_vector(11 downto 0);
           mc : out std_logic_vector(23 downto 0));
END mult1;
ARCHITECTURE rtl OF mult1 IS
    signal ta, tb : std_logic_vector(11 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then
        ta <= ma;  tb <= "100110111001";   mc <= ta * tb;
    end if;
end process;
END rtl;
```

11.1 资源优化

11.1.2 逻辑优化

【例11-4】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY mult2 IS
    PORT(clk : in std_logic;
         ma : In std_logic_vector(11 downto 0);
         mc : out std_logic_vector(23 downto 0));
END mult2;
ARCHITECTURE rtl OF mult2 IS
    signal ta : std_logic_vector(11 downto 0);
    constant tb : std_logic_vector(11 downto 0) := "100110111001";
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then  ta<=ma; mc<=ta * tb;
    end if;
end process;
END rtl;
```

11.1 资源优化

11.1.3 串行化

$$yout = a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$$

【例11-5】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY pmultadd IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         b0,b1,b2,b3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(15 downto 0));
END pmultadd;
ARCHITECTURE p_arch OF pmultadd IS
BEGIN
process(clk) begin
    if(clk'event and clk = '1') then
        yout <= ((a0*b0)+(a1*b1))+((a2*b2)+(a3*b3)); end if;
end process;
END p_arch;
```

11.1 资源优化

11.1.3 串行化

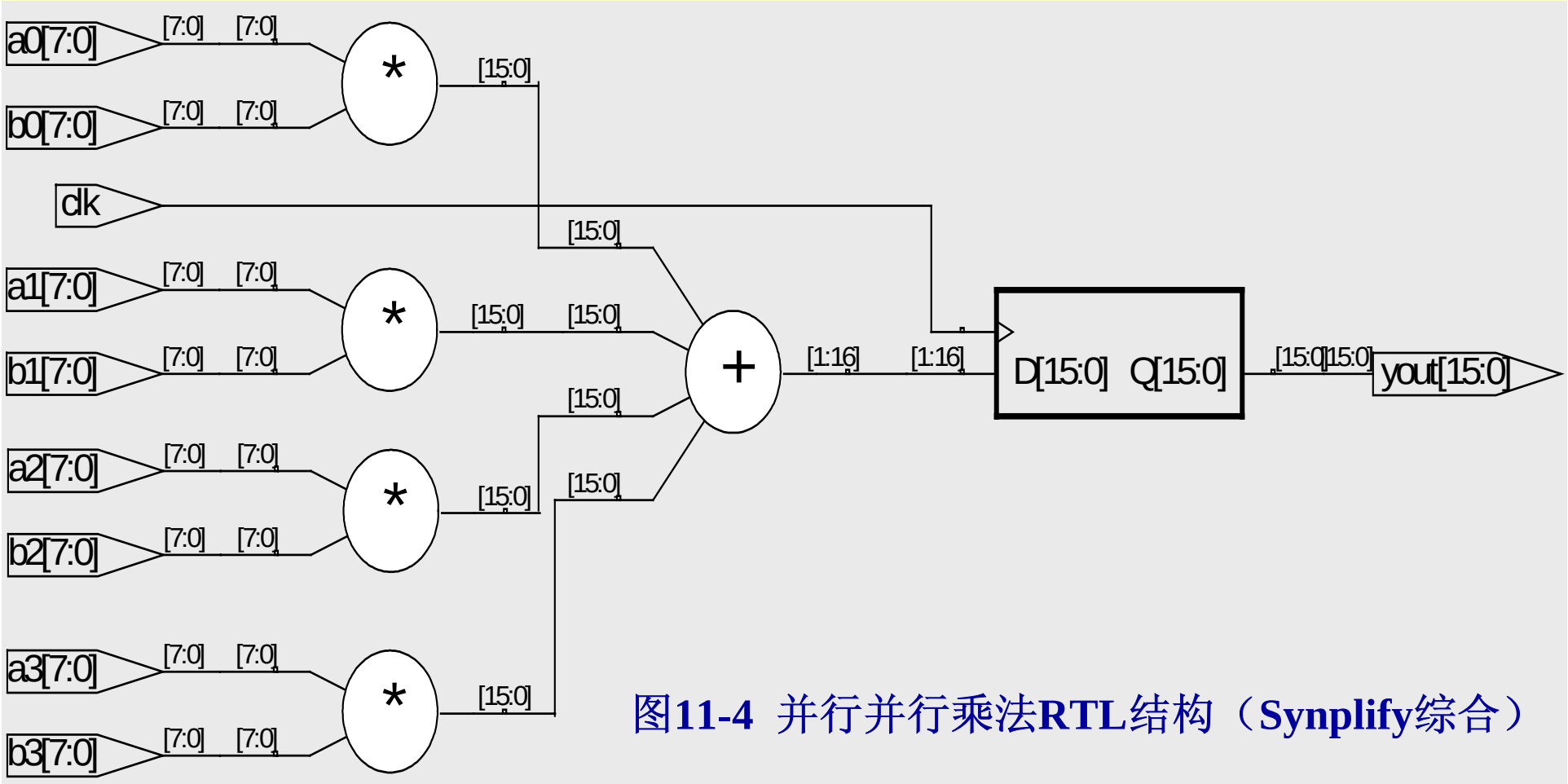


图11-4 并行并行乘法RTL结构（Synplify综合）

【例11-6】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY smultadd IS
    PORT(clk,  start : in std_logic;
         a0,a1,a2,a3 : In std_logic_vector(7 downto 0);
         b0,b1,b2,b3 : In std_logic_vector(7 downto 0);
         yout : out std_logic_vector(15 downto 0));
END smultadd;
ARCHITECTURE s_arch OF smultadd IS
    signal cnt : std_logic_vector(2 downto 0);
    signal tmpa, tmpb : std_logic_vector(7 downto 0);
    signal  tmp,  ytmp : std_logic_vector(15 downto 0);
  BEGIN
tmpa <= a0 when cnt = 0 else
         a1 when cnt = 1 else
         a2 when cnt = 2 else
         a3 when cnt = 3 else
         a0;
tmpb <= b0 when cnt = 0 else
         b1 when cnt = 1 else
         b2 when cnt = 2 else
         b3 when cnt = 3 else
         b0;
tmp <= tmpa * tmpb;
process(clk) begin
    if(clk'event and clk = '1') then
        if(start = '1') then  cnt <= "000"; ytmp <= (others=>'0');
        elsif (cnt<4) then  cnt <= cnt + 1; ytmp <= ytmp + tmp;
        elsif (cnt = 4) then  yout <= ytmp;
        end if;
    end if;
end process;
END s_arch;
```

11.2 速度优化

11.2.1 流水线设计

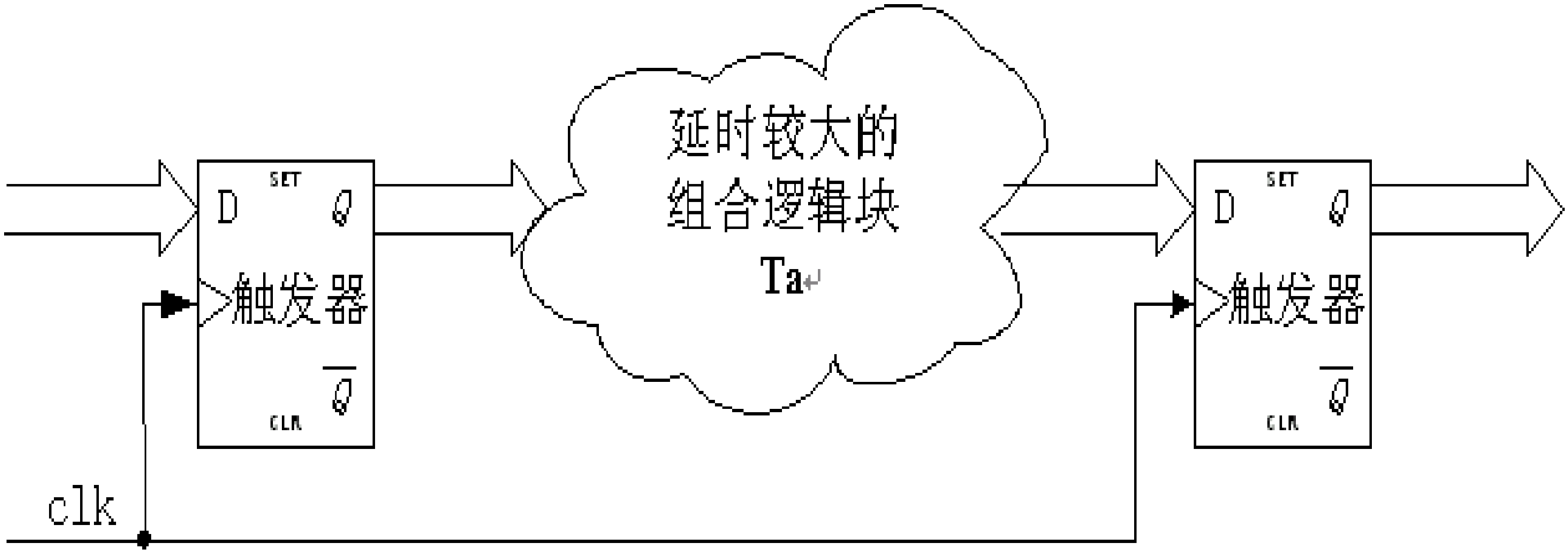


图11-5 未使用流水线

11.2 速度优化

11.2.1 流水线设计

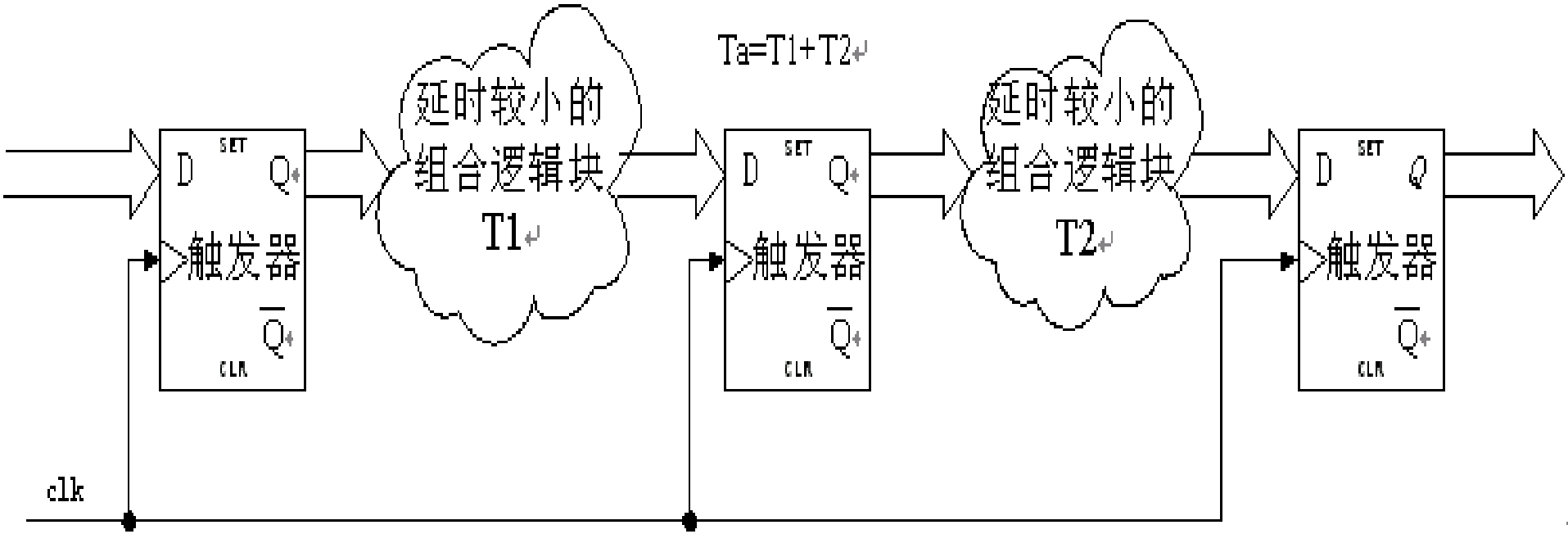


图11-6 使用流水线

11.2 速度优化

11.2.1 流水线设计

图11-7 流水线工作图示

【例11-7】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY adder4 IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(9 downto 0));
END adder4;
ARCHITECTURE normal_arch OF adder4 IS
    signal t0,t1,t2,t3 : std_logic_vector(7 downto 0);
    signal addtmp0,addtmp1 : std_logic_vector(8 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk='1') then
        t0 <= a0;  t1 <= a1;  t2 <= a2;   t3 <= a3;
    end if;
end process;
addtmp0 <= '0'&t0 + t1;
addtmp1 <= '0'&t2 + t3;
process(clk) begin
    if(clk'event and clk = '1') then yout <= '0'&addtmp0 + addtmp1;
    end if;
end process;
END normal_arch;
```

【例11-8】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
ENTITY pipeadd IS
    PORT(clk : in std_logic;
         a0,a1,a2,a3 : in std_logic_vector(7 downto 0);
         yout : out std_logic_vector(9 downto 0));
END pipeadd;
ARCHITECTURE pipelining_arch OF pipeadd IS
    signal t0,t1,t2,t3 : std_logic_vector(7 downto 0);
    signal addtmp0,addtmp1 : std_logic_vector(8 downto 0);
BEGIN
process(clk) begin
    if(clk'event and clk='1') then
        t0 <= a0; t1 <= a1;  t2 <= a2;  t3 <= a3;
    end if;
end process;
process(clk) begin
    if(clk'event and clk = '1') then
        addtmp0 <= '0'&t0 + t1;
        addtmp1 <= '0'&t2 + t3;
yout <= '0'&addtmp0 + addtmp1;
    end if;
end process;
END pipelining_arch;
```

11.2 速度优化

11.2.2 寄存器配平

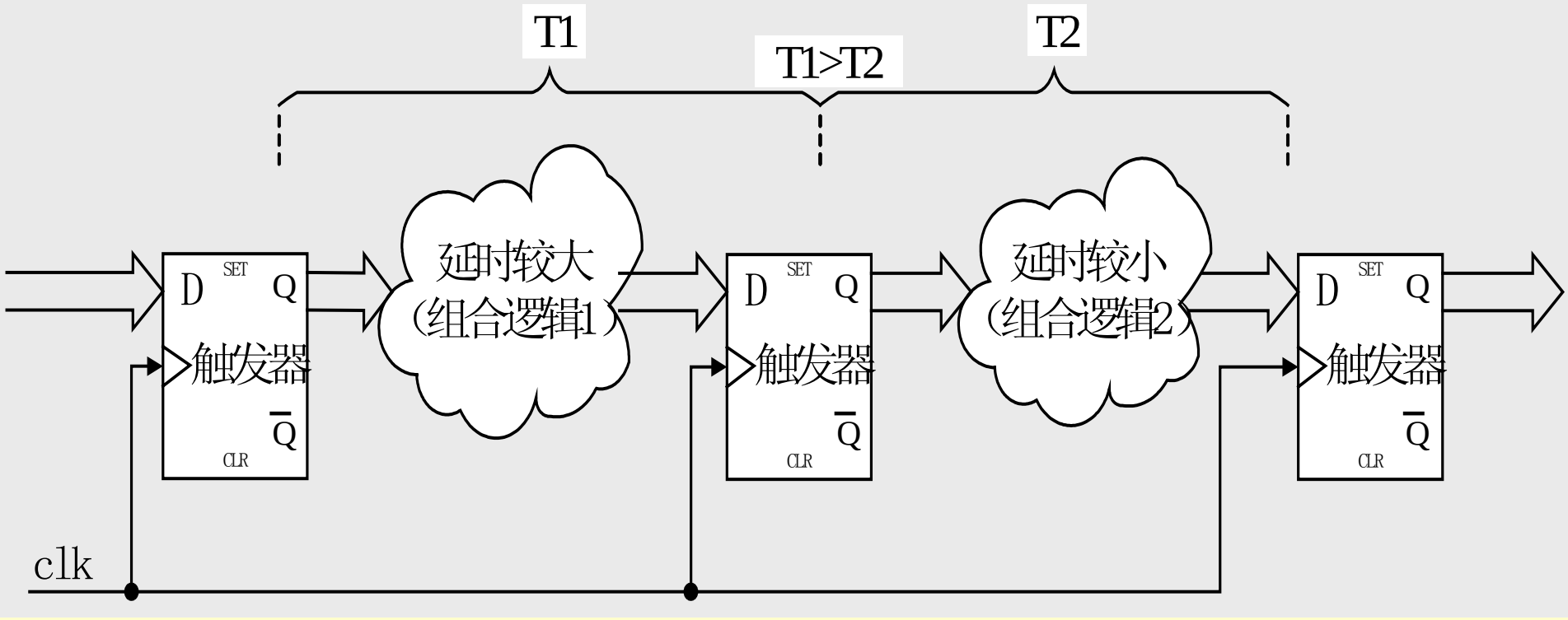


图11-8 不合理的结构

11.2 速度优化

11.2.2 寄存器配平

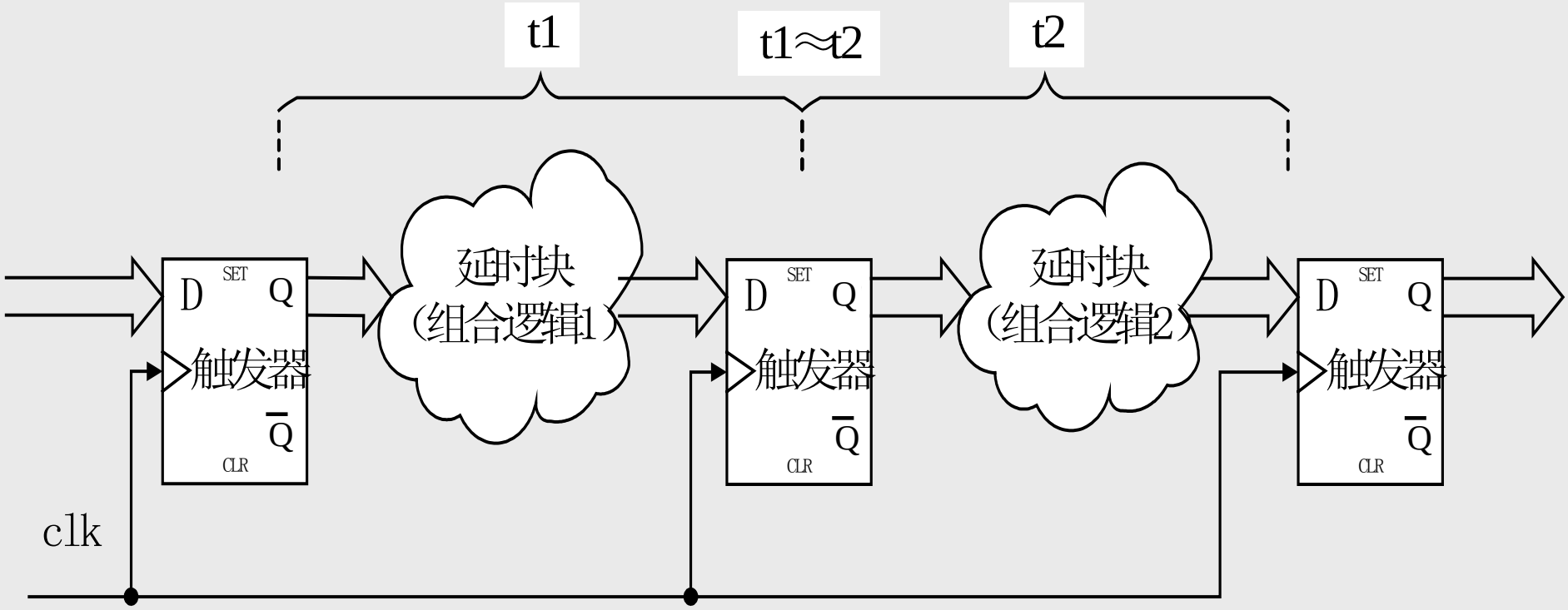


图11-9 寄存器配平的结构

11.2 速度优化

11.2.3 关键路径法

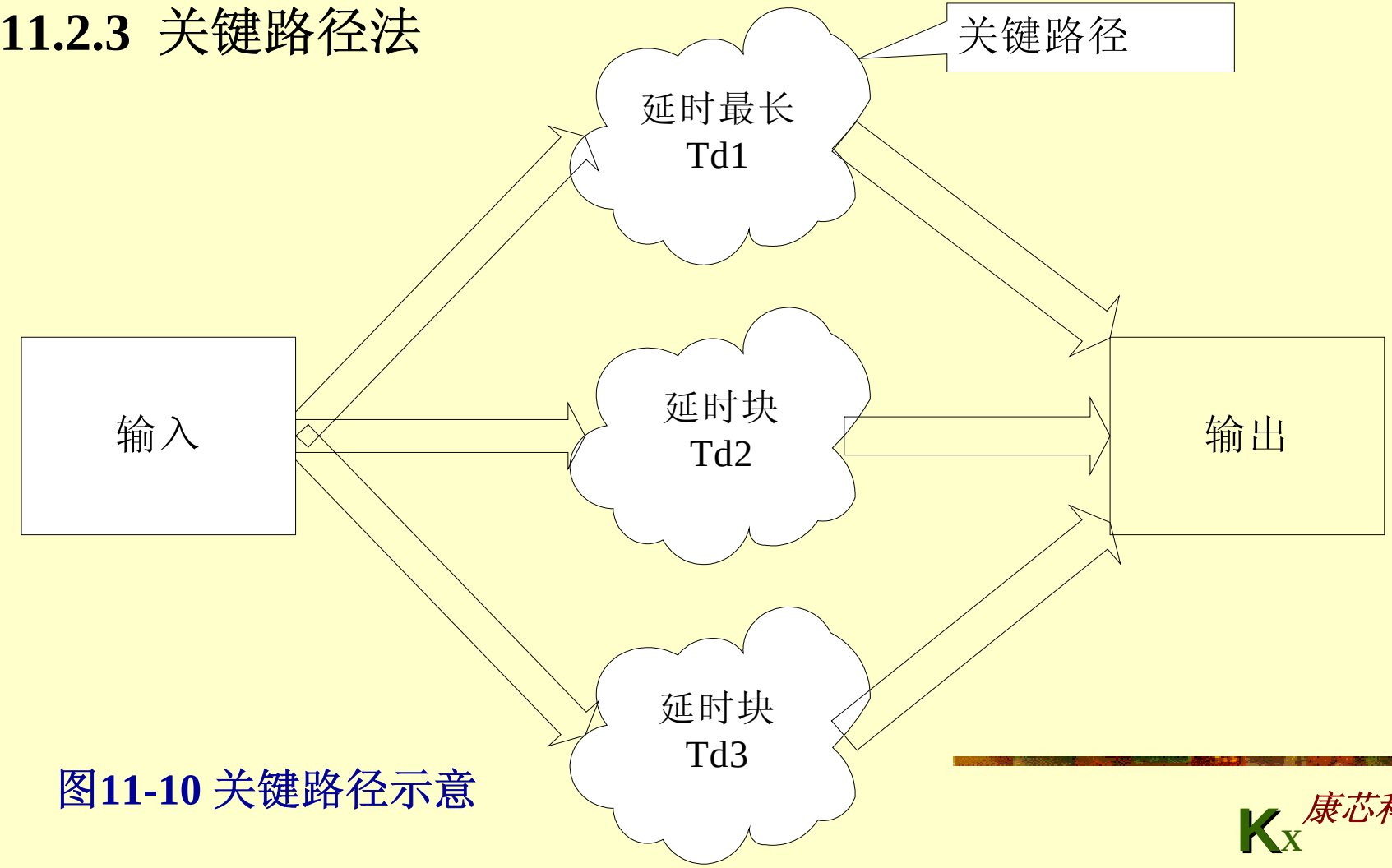


图11-10 关键路径示意

11.3 优化设置与时序分析

11.3.1 Settings设置

11.3.2 HDL版本设置及Analysis & Synthesis功能

11.3.3 Analysis & Synthesis的优化设置

11.3.4 适配器Fitter设置

11.3 优化设置与时序分析

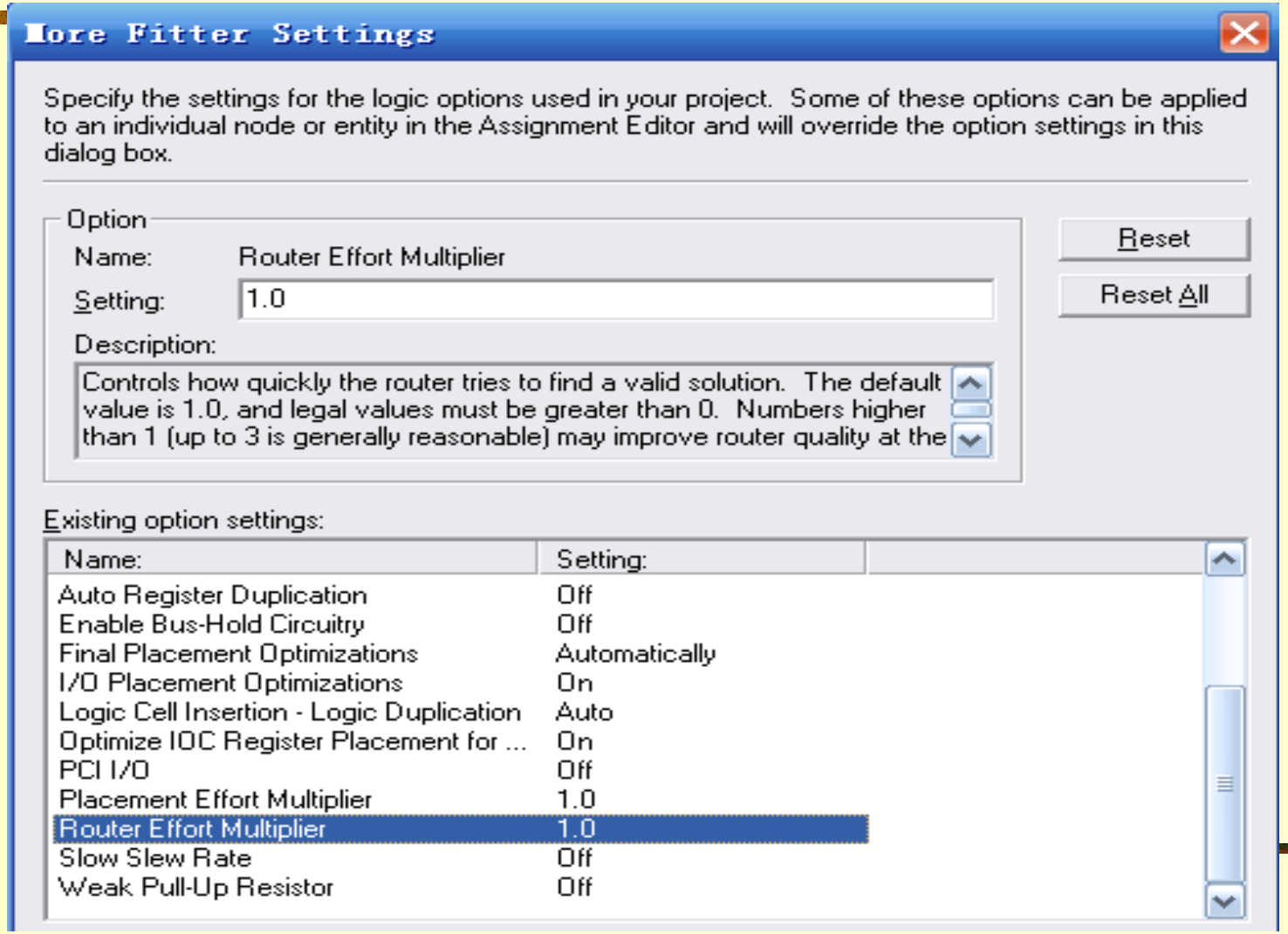


图9-11 布线倍增器优化程度指数选择

11.3 优化设置与时序分析

11.3.5 增量布局布线控制设置

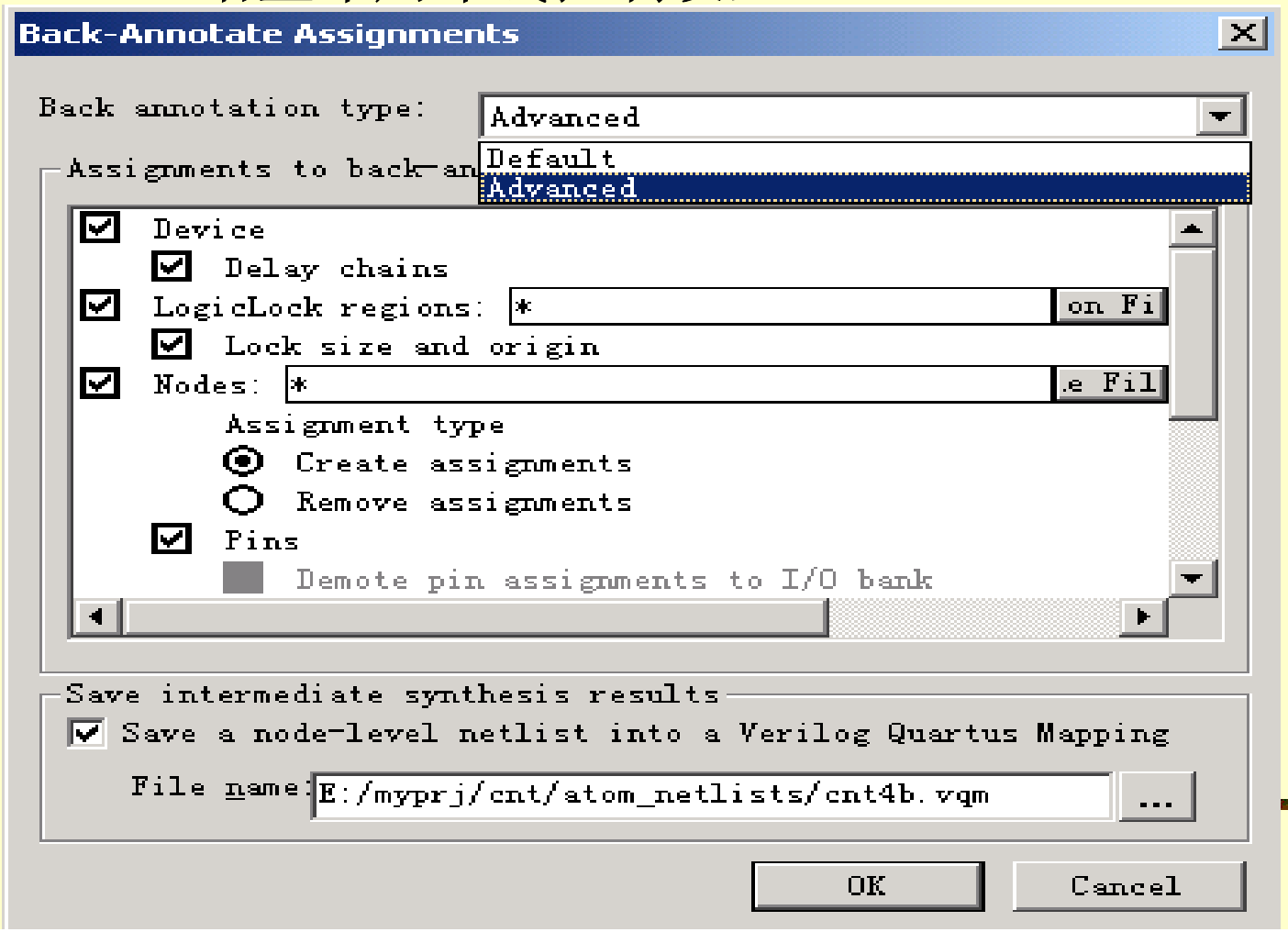


图11-12 反标设置

11.3 优化设置与时序分析

11.3.6 使用Design Assistant检查设计可靠性

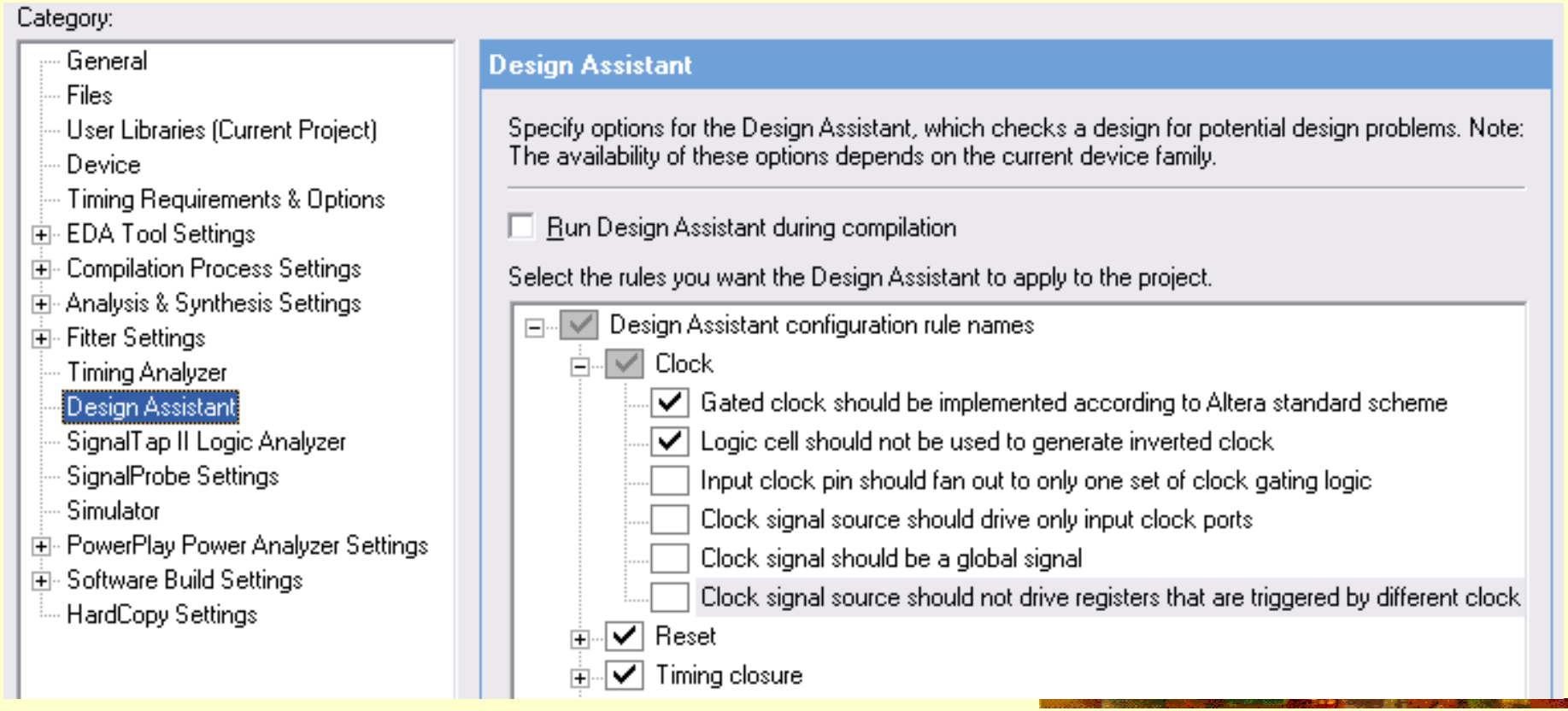


图11-13 Design Assistant设置

11.3 优化设置与时序分析

11.3.7 时序设置与分析

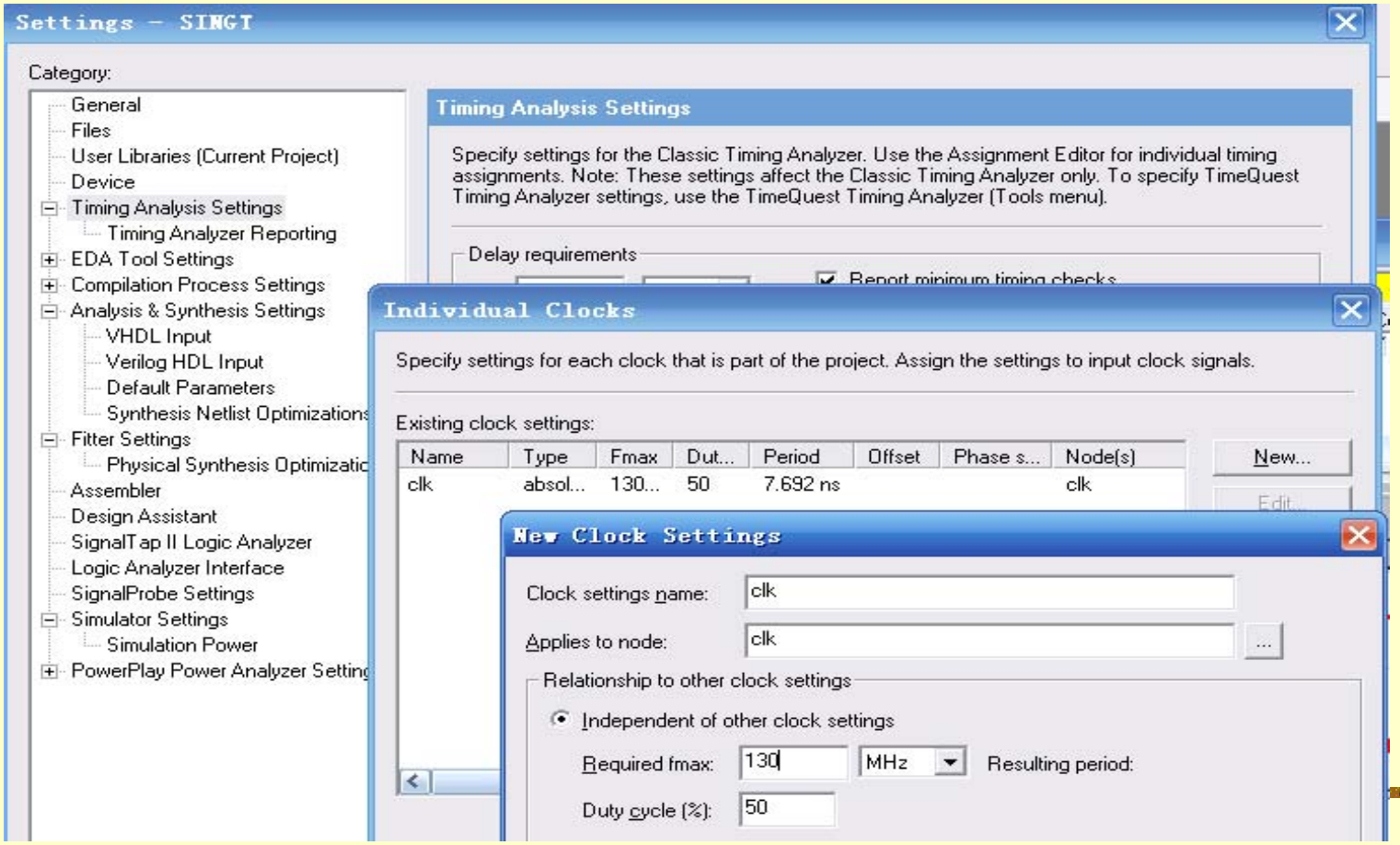


图11-14 全编译前时序条件设置（设置时钟信号CLK不低于130MHz）

11.3 优化设置与时序分析

11.3.7 时序设置与分析

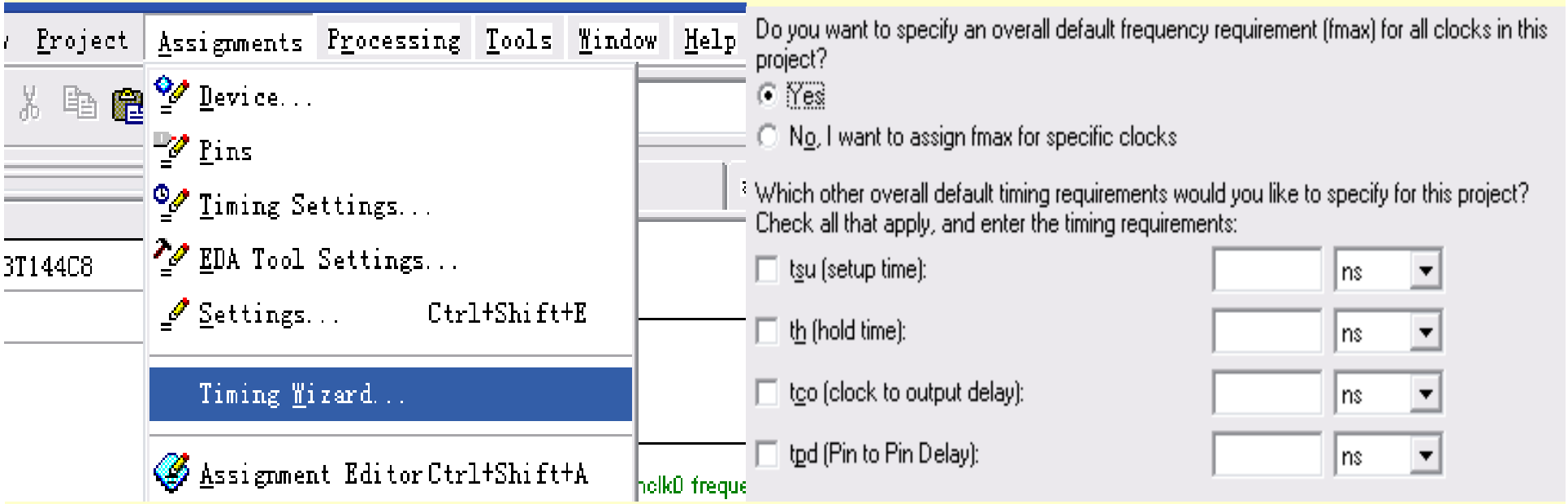
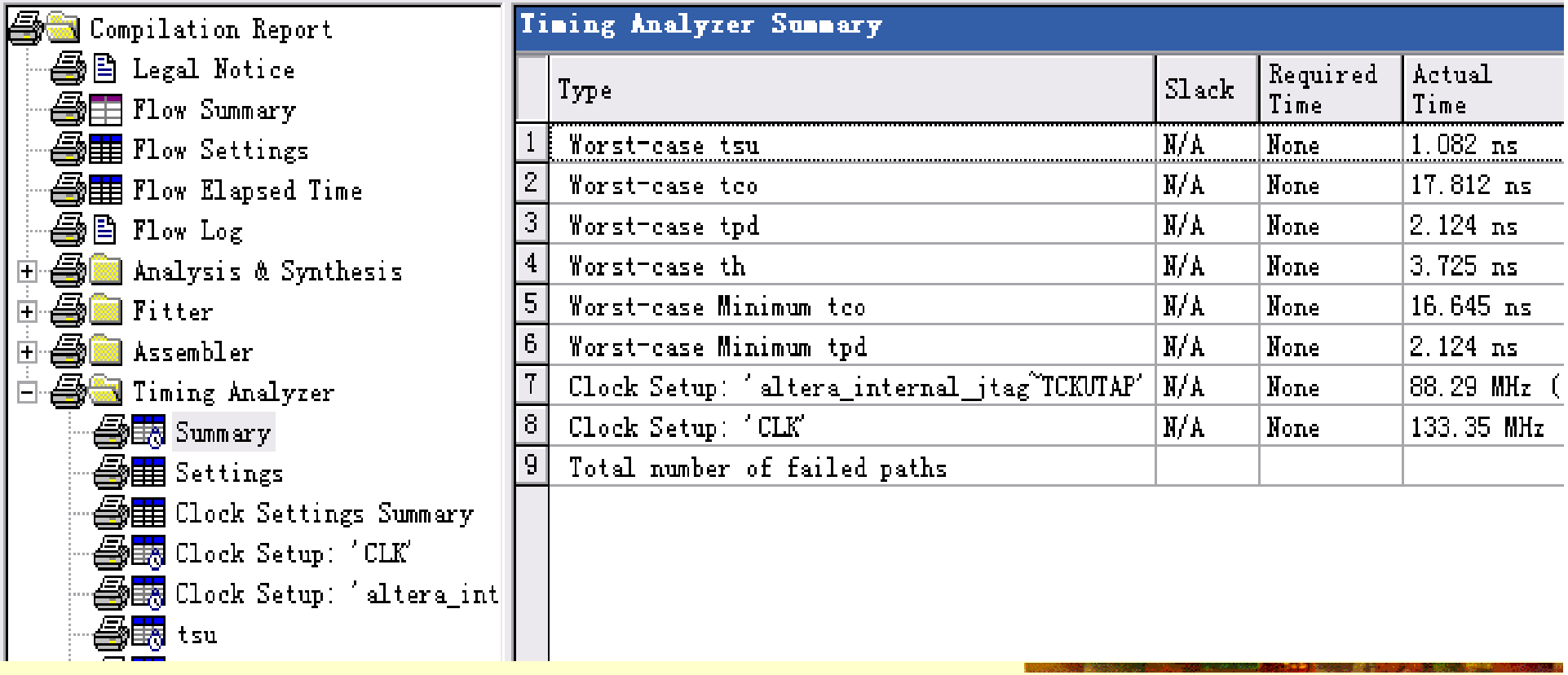


图11-15 由Timing Wizard窗口设置时序条件

11.3 优化设置与时序分析

11.3.8 查看时序分析结果



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	1.082 ns
2	Worst-case tco	N/A	None	17.812 ns
3	Worst-case tpd	N/A	None	2.124 ns
4	Worst-case th	N/A	None	3.725 ns
5	Worst-case Minimum tco	N/A	None	16.645 ns
6	Worst-case Minimum tpd	N/A	None	2.124 ns
7	Clock Setup: 'altera_internal_jtag~TCKUTAP'	N/A	None	88.29 MHz (
8	Clock Setup: 'CLK'	N/A	None	133.35 MHz
9	Total number of failed paths			

图11-16 时序分析报告窗

11.3 优化设置与时序分析

11.3.8 查看时序分析结果

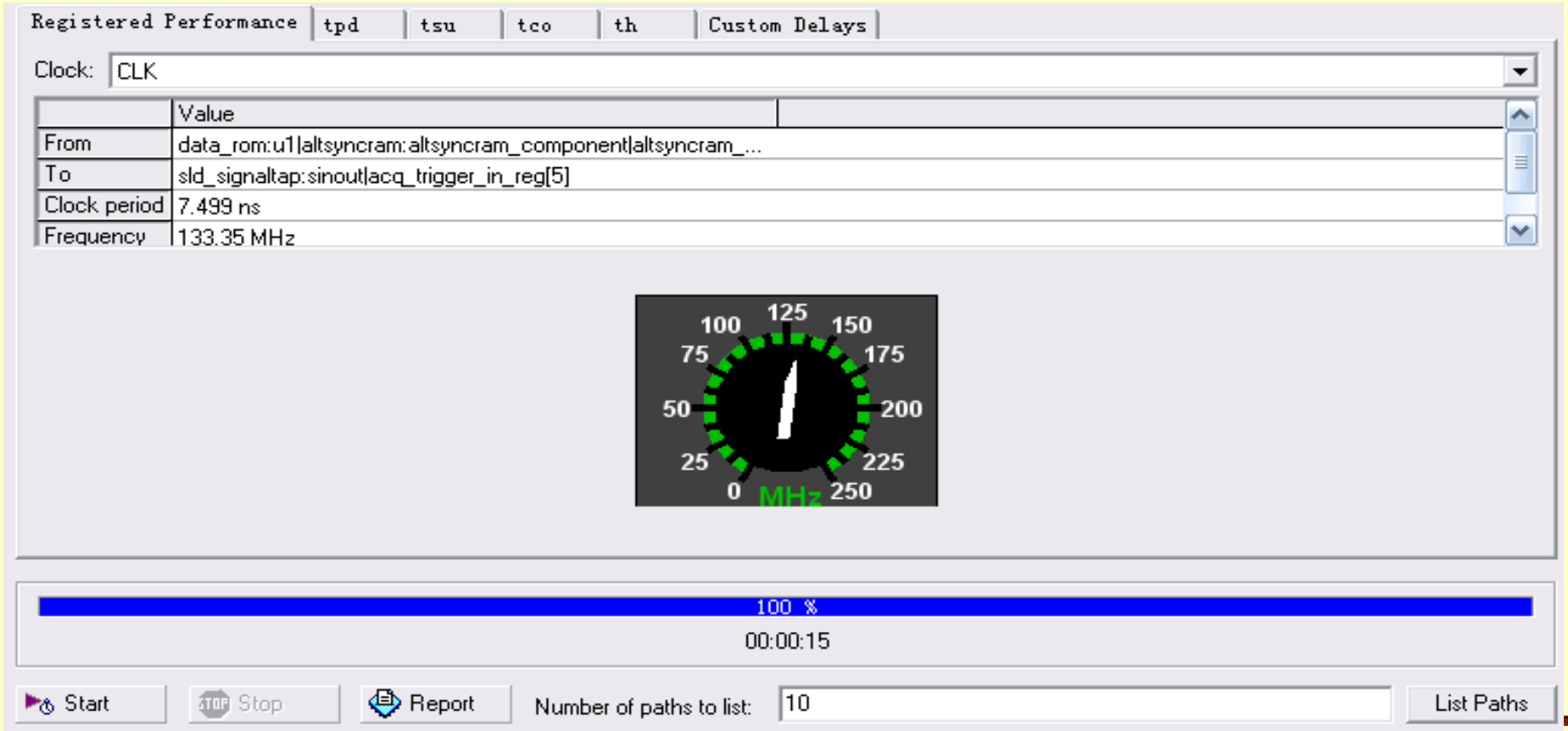


图11-17 Timing Analyzer Tool 项进入的时序分析报告窗

11.3 优化设置与时序分析

11.3.9 适配优化设置示例

Family	APEX20KE
Device	EP20K300EQC240-3
Timing Models	Final
Met timing requirements	Yes
Total logic elements	51 / 11,520 (< 1 %)
Total pins	9 / 152 (5 %)
Total virtual pins	0
Total memory bits	0 / 147,456 (0 %)
Total PLLs	0

图11-18 未用乘积项前的编译报告

【例11-9】 用CASE语句设计的正弦信号发生器

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SINGT IS
    PORT ( CLK      : IN STD_LOGIC;
           DOUT     : OUT INTEGER RANGE 255 DOWNTO 0      );
END;
ARCHITECTURE DACC OF SINGT IS
    SIGNAL Q        : INTEGER RANGE 63 DOWNTO 0 ;
    SIGNAL D        : INTEGER RANGE 255 DOWNTO 0 ;
BEGIN
PROCESS(CLK)
   BEGIN
   IF CLK'EVENT AND CLK = '1' THEN
IF Q < 63 THEN   Q <= Q + 1; ELSE   Q <= 0 ; END IF;  END IF;
END PROCESS;
PROCESS(Q)
   BEGIN
CASE Q IS
```

(接下页)

KX 康芯科技

```
WHEN 00=> D<=255; WHEN 01=> D<=254; WHEN 02=> D<=252; WHEN 03=> D<=249;
WHEN 04=> D<=245; WHEN 05=> D<=239; WHEN 06=> D<=233; WHEN 07=> D<=225;
WHEN 08=> D<=217; WHEN 09=> D<=207; WHEN 10=> D<=197; WHEN 11=> D<=186;
WHEN 12=> D<=174; WHEN 13=> D<=162; WHEN 14=> D<=150; WHEN 15=> D<=137;
WHEN 16=> D<=124; WHEN 17=> D<=112; WHEN 18=> D<= 99; WHEN 19=> D<= 87;
WHEN 20=> D<= 75; WHEN 21=> D<= 64; WHEN 22=> D<= 53; WHEN 23=> D<= 43;
WHEN 24=> D<= 34; WHEN 25=> D<= 26; WHEN 26=> D<= 19; WHEN 27=> D<= 13;
WHEN 28=> D<=  8; WHEN 29=> D<=  4; WHEN 30=> D<=  1; WHEN 31=> D<=  0;
WHEN 32=> D<=  0; WHEN 33=> D<=  1; WHEN 34=> D<=  4; WHEN 35=> D<=  8;
WHEN 36=> D<= 13; WHEN 37=> D<= 19; WHEN 38=> D<= 26; WHEN 39=> D<= 34;
WHEN 40=> D<= 43; WHEN 41=> D<= 53; WHEN 42=> D<= 64; WHEN 43=> D<= 75;
WHEN 44=> D<= 87; WHEN 45=> D<= 99; WHEN 46=> D<=112; WHEN 47=> D<=124;
WHEN 48=> D<=137; WHEN 49=> D<=150; WHEN 50=> D<=162; WHEN 51=> D<=174;
WHEN 52=> D<=186; WHEN 53=> D<=197; WHEN 54=> D<=207; WHEN 55=> D<=217;
WHEN 56=> D<=225; WHEN 57=> D<=233; WHEN 58=> D<=239; WHEN 59=> D<=245;
WHEN 60=> D<=249; WHEN 61=> D<=252; WHEN 62=> D<=254; WHEN 63=> D<=255;
WHEN OTHERS => NULL ;
END CASE;  END PROCESS;
  DOUT <= D ;
END;
```

11.3 优化设置与时序分析

11.3.9 适配优化设置示例

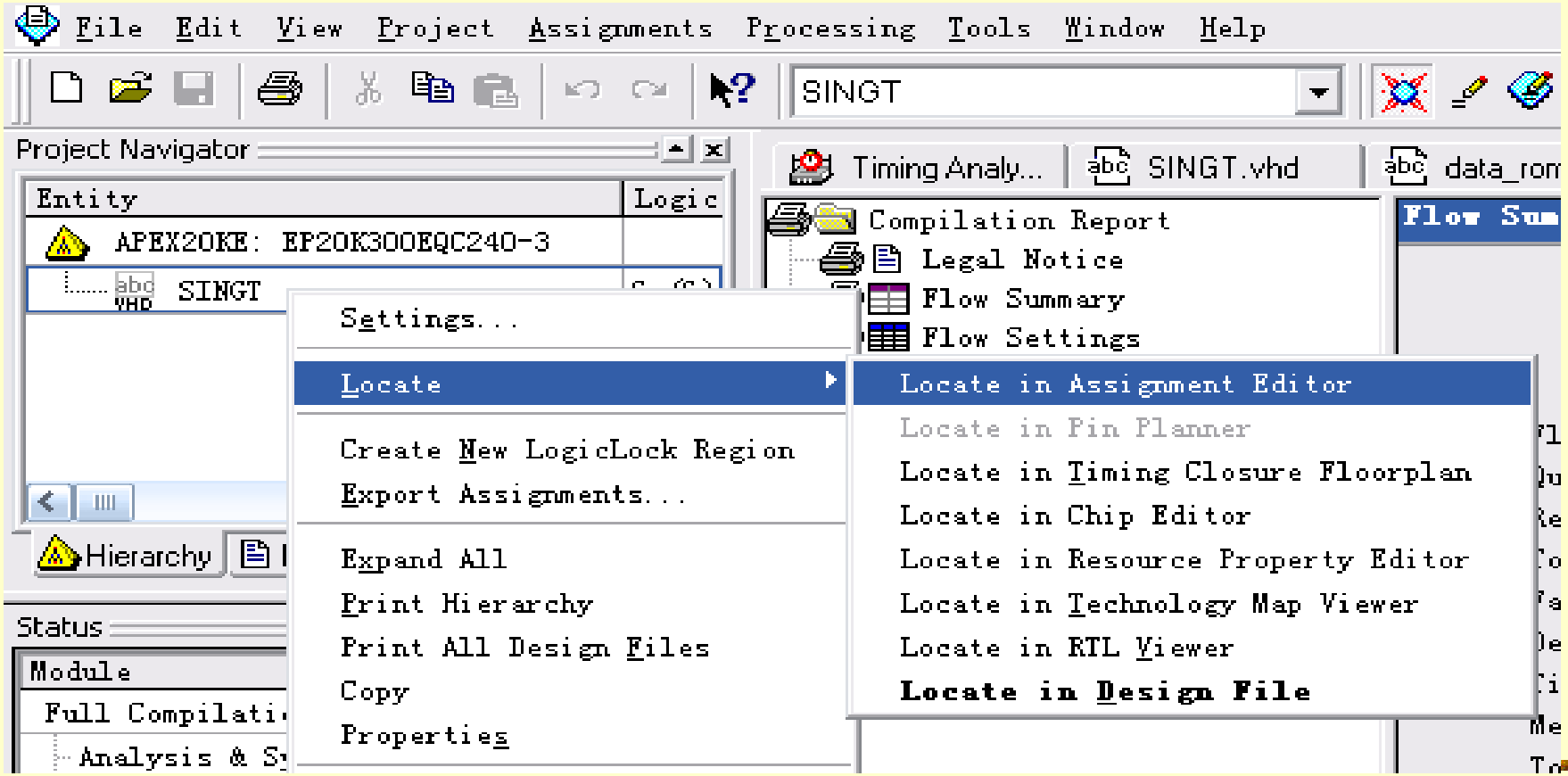


图11-19 针对工程选择Locate in Assignment Editor

康芯科技

11.3 优化设置与时序分析

11.3.9 适配优化设置示例

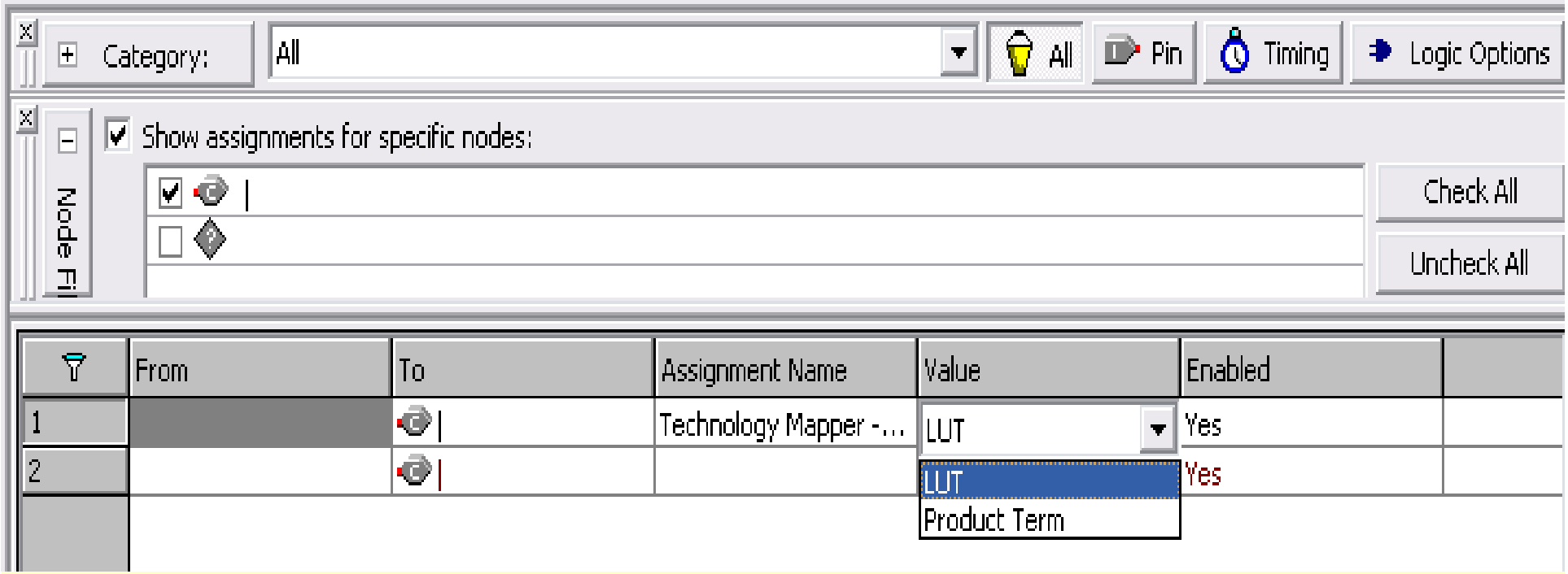


图11-20 选用乘积项逻辑优化

11.3 优化设置与时序分析

11.3.9 适配优化设置示例

图11-21在floorplan中可以看到使用了32个ESB

11.3 优化设置与时序分析

11.3.9 适配优化设置示例

Family	APEX20KE
Device	EP20K300EQC240-3
Timing Models	Final
Met timing requirements	Yes
Total logic elements	6 / 11,520 (< 1 %)
Total pins	9 / 152 (5 %)
Total virtual pins	0
Number of product terms	48
Total PLLs	0
Total memory bits	6,144 / 147,456 (4 %)

图11-22使用了乘积项的编译报告

11.3 优化设置与时序分析

11.3.10 Slow Slew Rate设置

Existing option settings:

Name:	Setting:
Auto Register Duplication	Off
Enable Bus-Hold Circuitry	Off
Final Placement Optimizations	Automatically
I/O Placement Optimizations	On
Logic Cell Insertion - Logic Duplication	Auto
Optimize IOC Register Placement for ...	On
PCI I/O	Off
Placement Effort Multiplier	1.0
Router Effort Multiplier	1.0
Slow Slew Rate	On
Weak Pull-Up Resistor	Off

图11-23 Slow Slew Rate选择

11.3 优化设置与时序分析

11.3.11 LogicLock优化技术

大规模系统开发中，应用逻辑锁定技术可以优化设计，合理分配硬件资料，提高系统的工作速度和可靠性。**QuartusII**支持逻辑锁定技术的**FPGA**器件系列有**APEX20K**、**APEXII**、**Excalibur**、**Cyclone/II**和**Stratix/II**等。

11.4 Chip Editor应用

11.4.1 Chip Editor应用实例

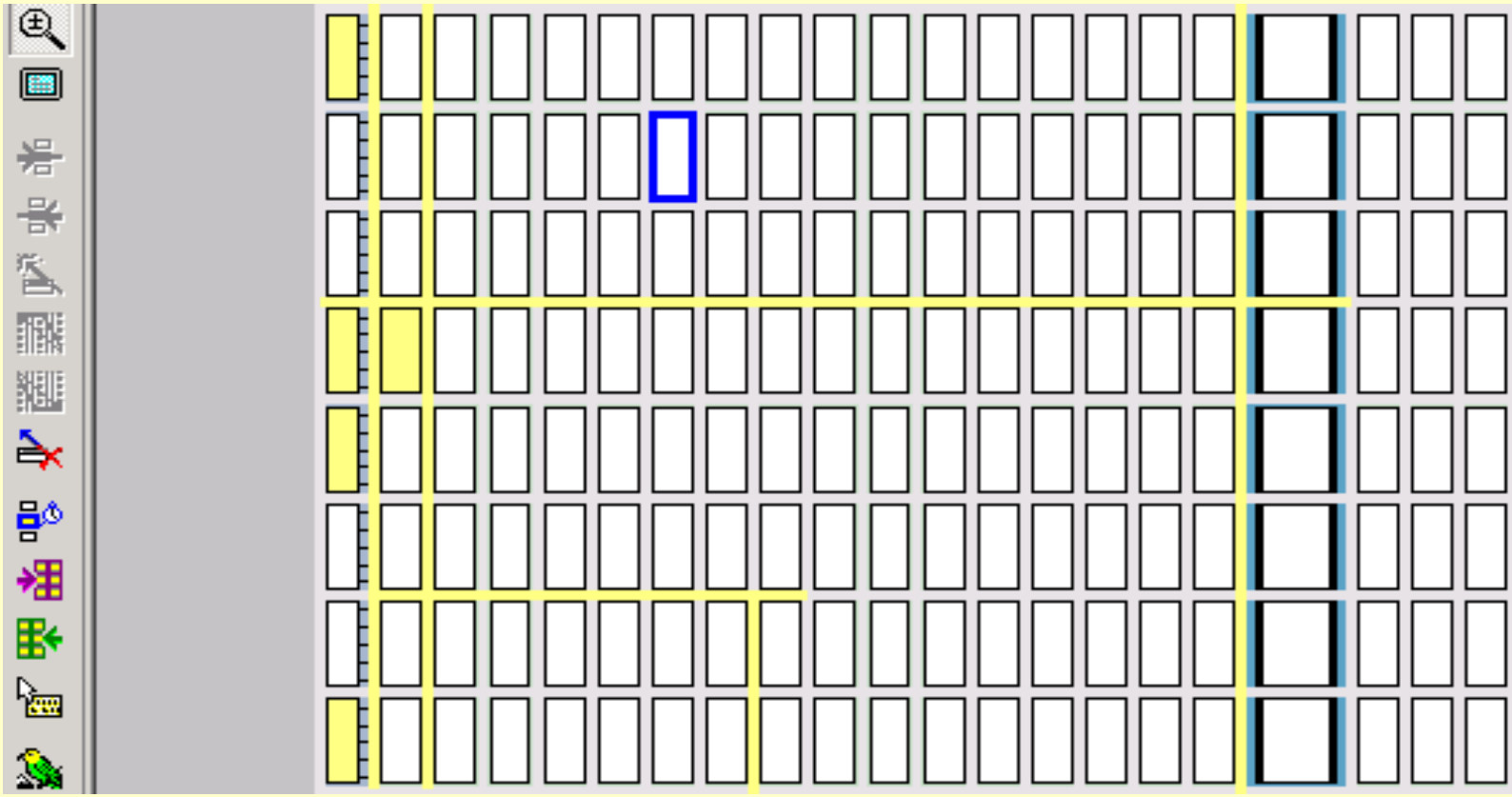
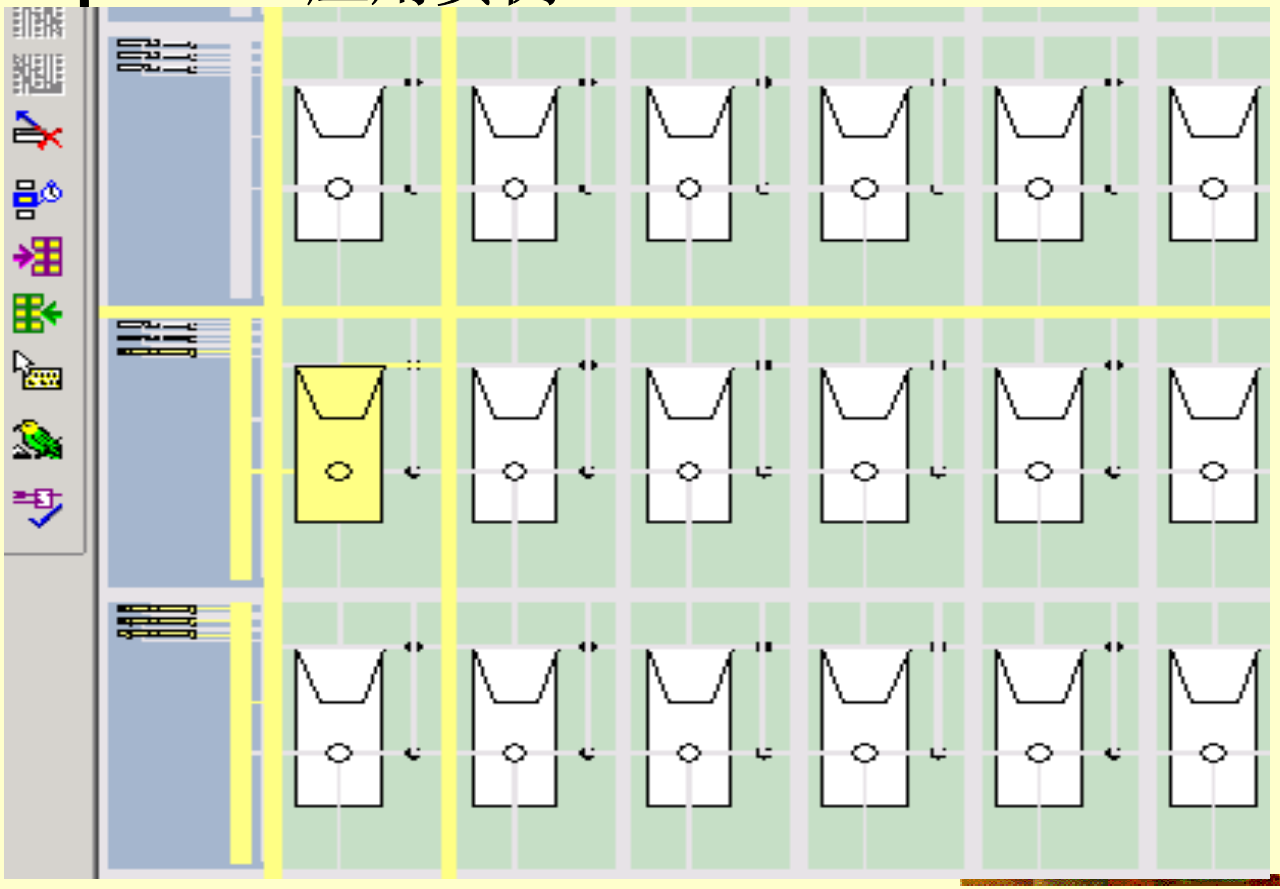


图9-24 最左侧是CNT4B占用的LAB

11.4 Chip Editor应用

11.4.1 Chip Editor应用实例



9-25 放大后的LAB分布

11.4 Chip Editor应用

11.4.1 Chip Editor应用实例

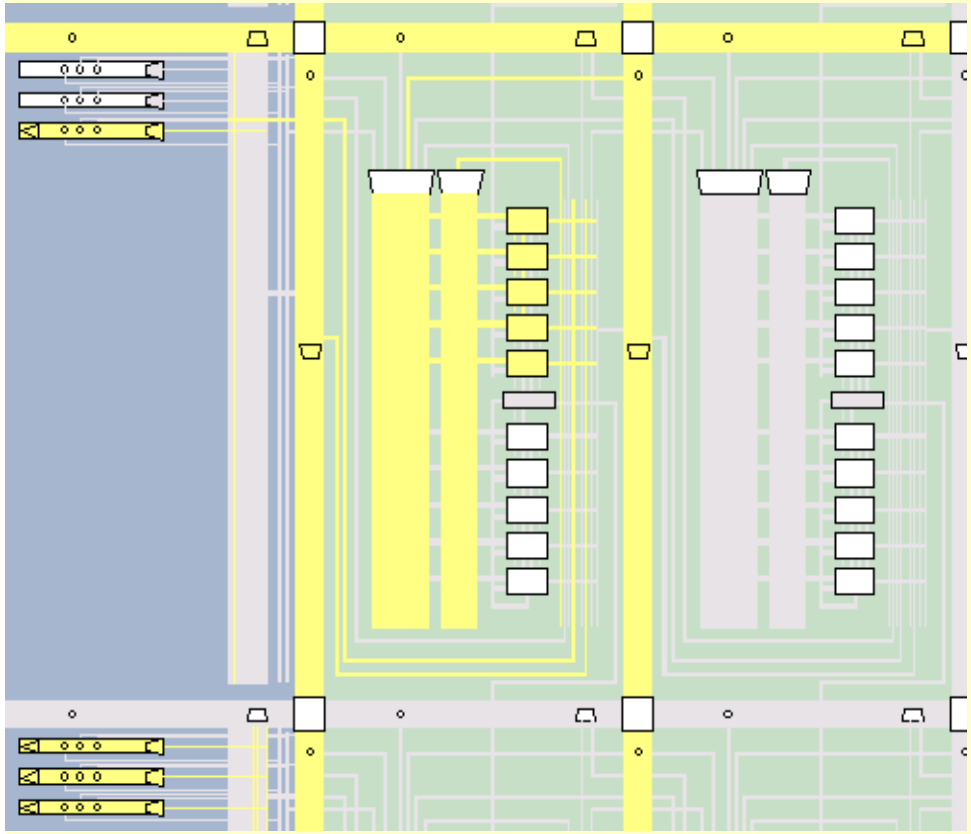


图11-26 被占用的LAB

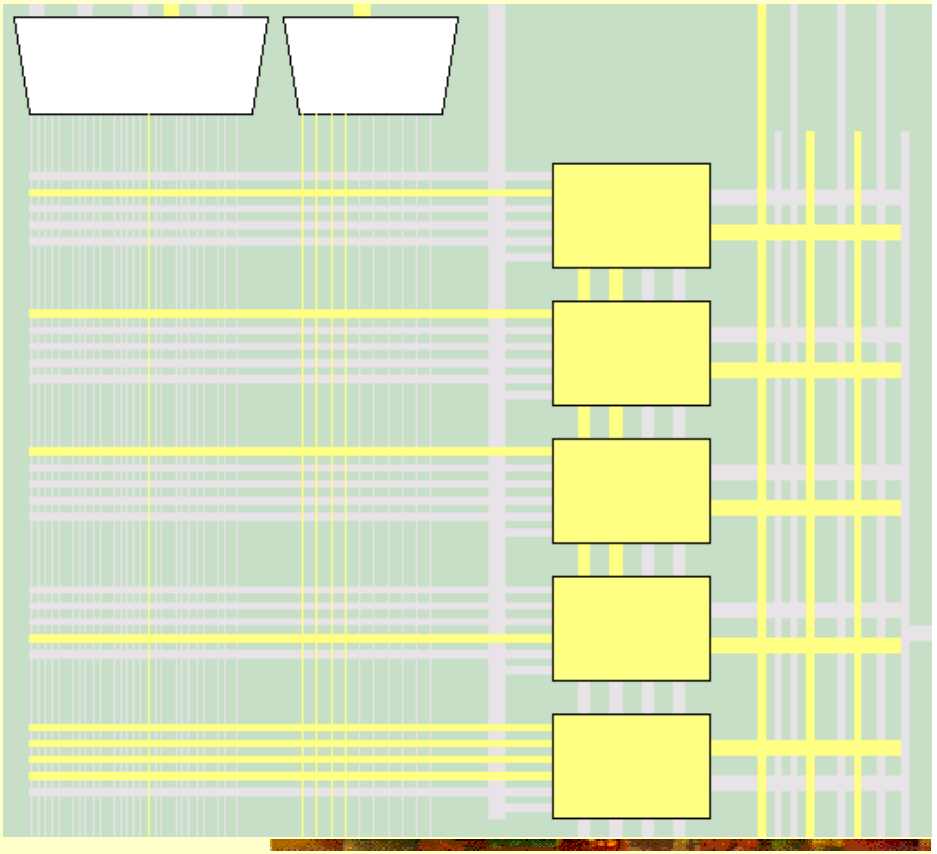


图11-27 LAB中被占用的5个LCs

11.4 Chip Editor应用

11.4.1 Chip Editor应用实例

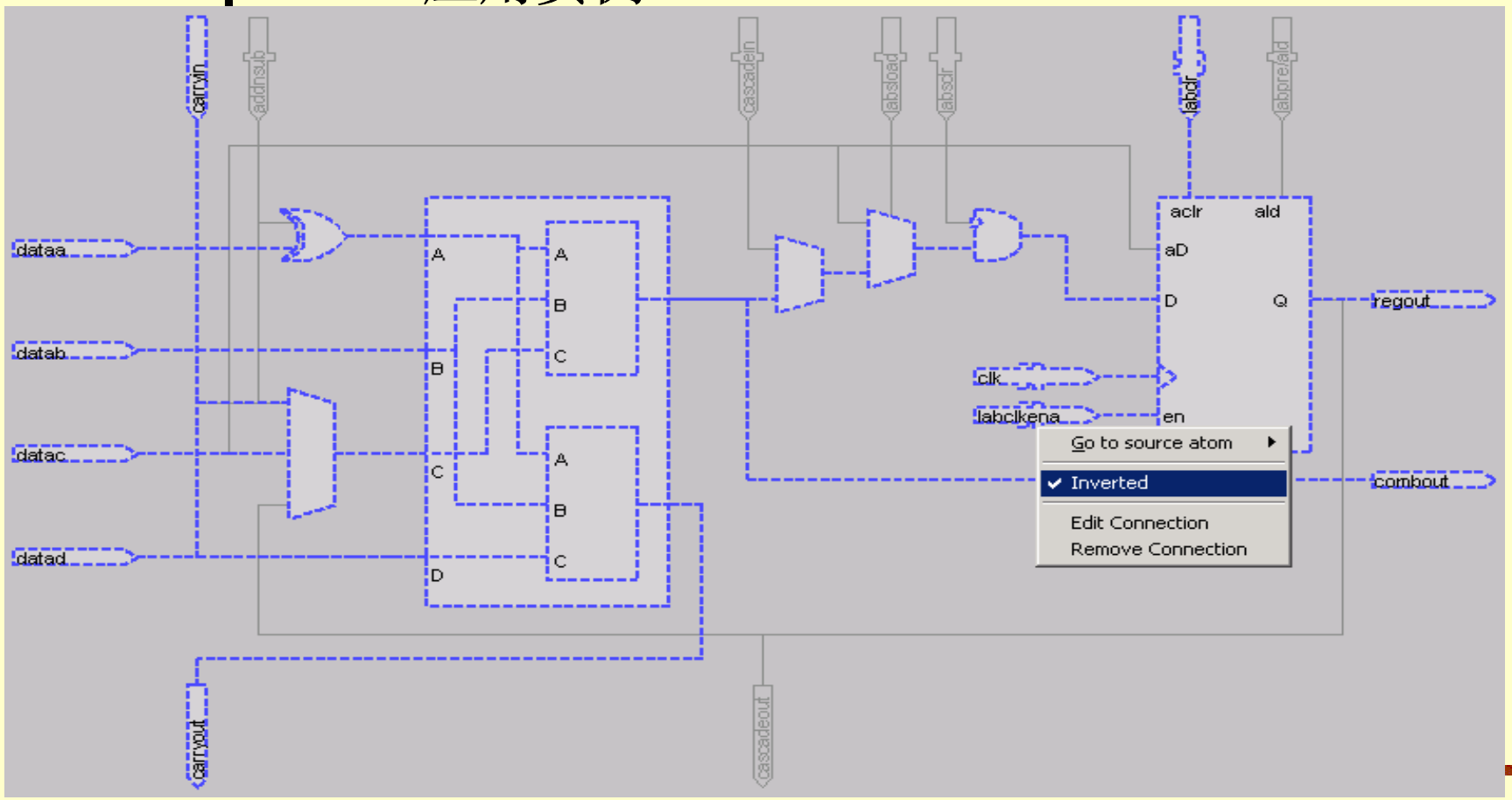


图11-28 Resource Property Editor的门级原理图编辑窗

康芯科技

11.4 Chip Editor应用

11.4.1 Chip Editor应用实例

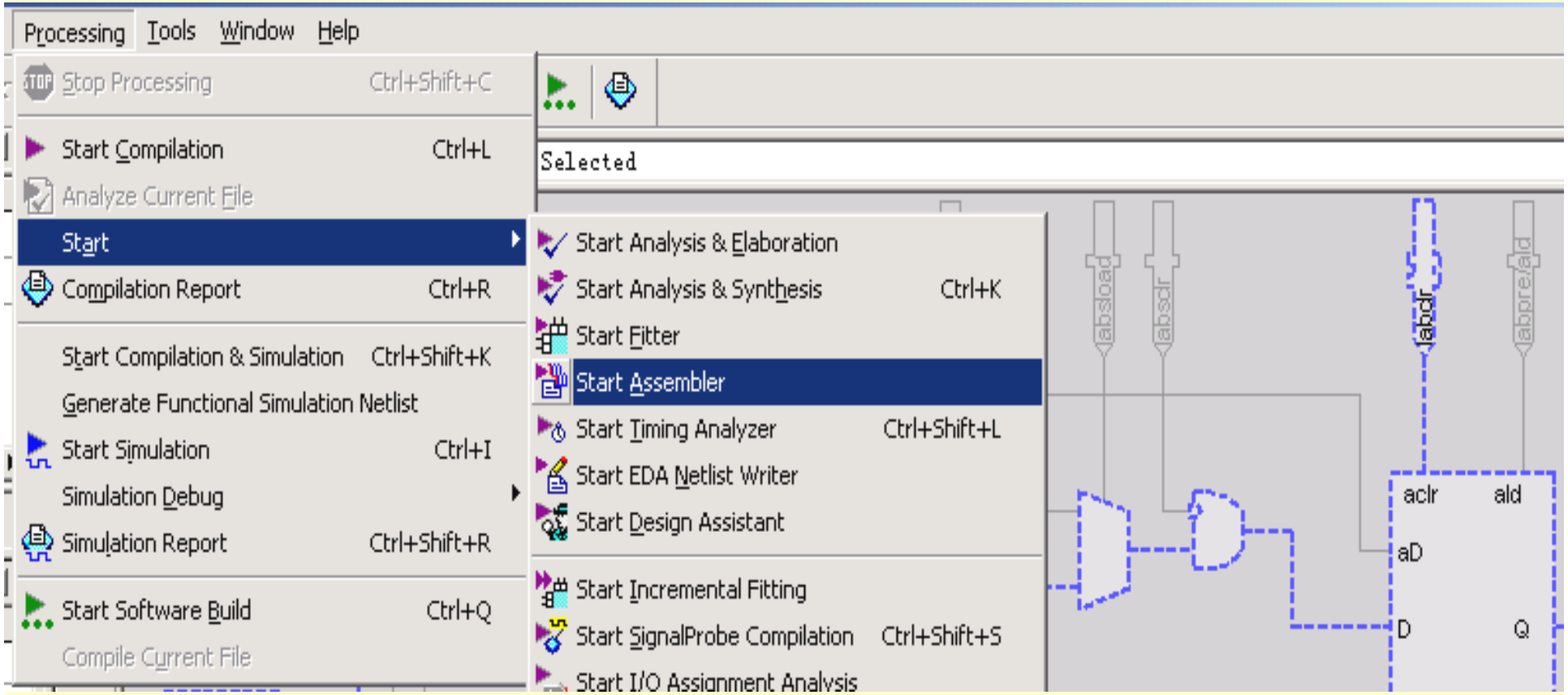


图11-29 的时序分析报告窗图

康芯科技

11.4 Chip Editor应用

11.4.2 Chip Editor功能说明

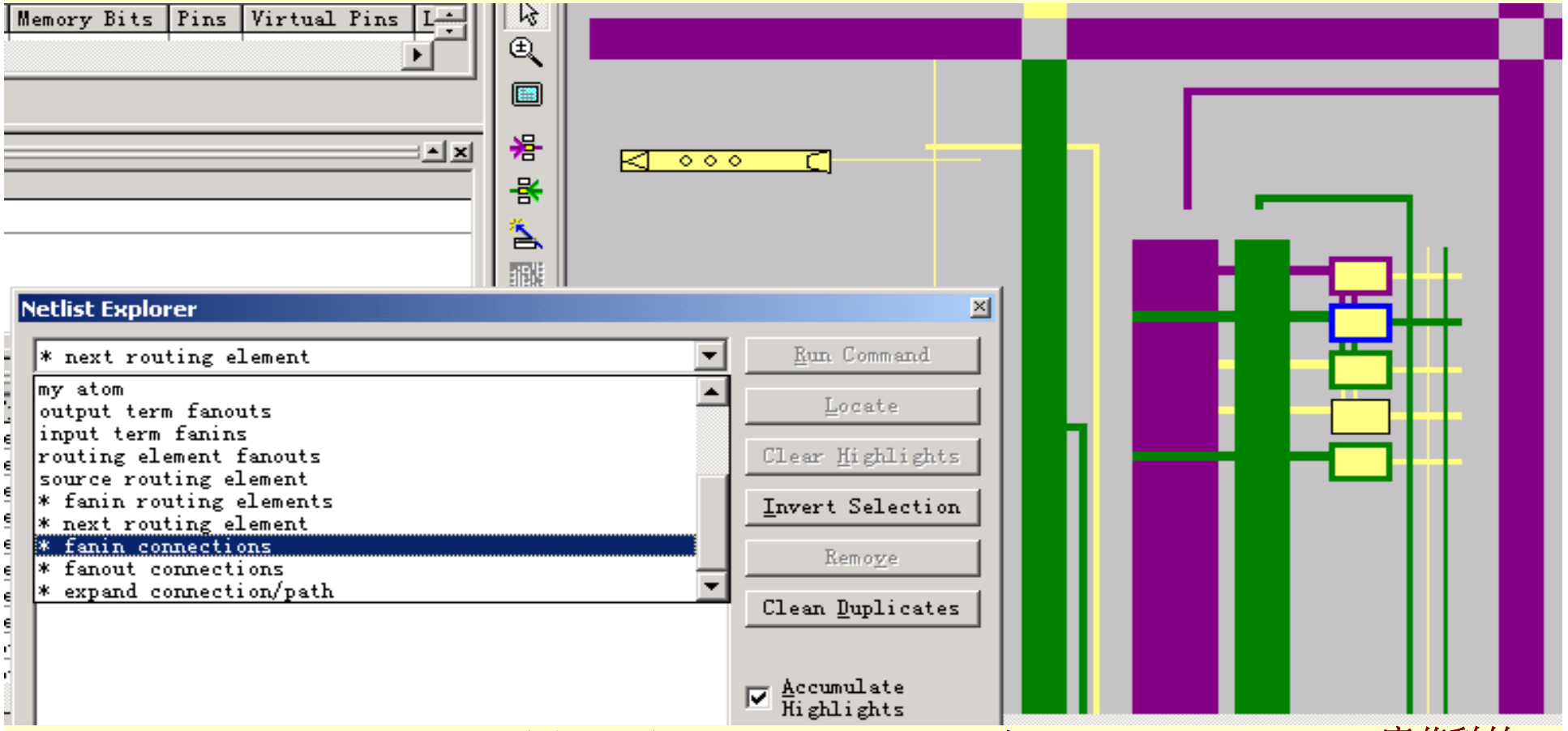


图9-30 打开Netlist Explorer窗

11.4 Chip Editor应用

11.4.2 Chip Editor功能说明

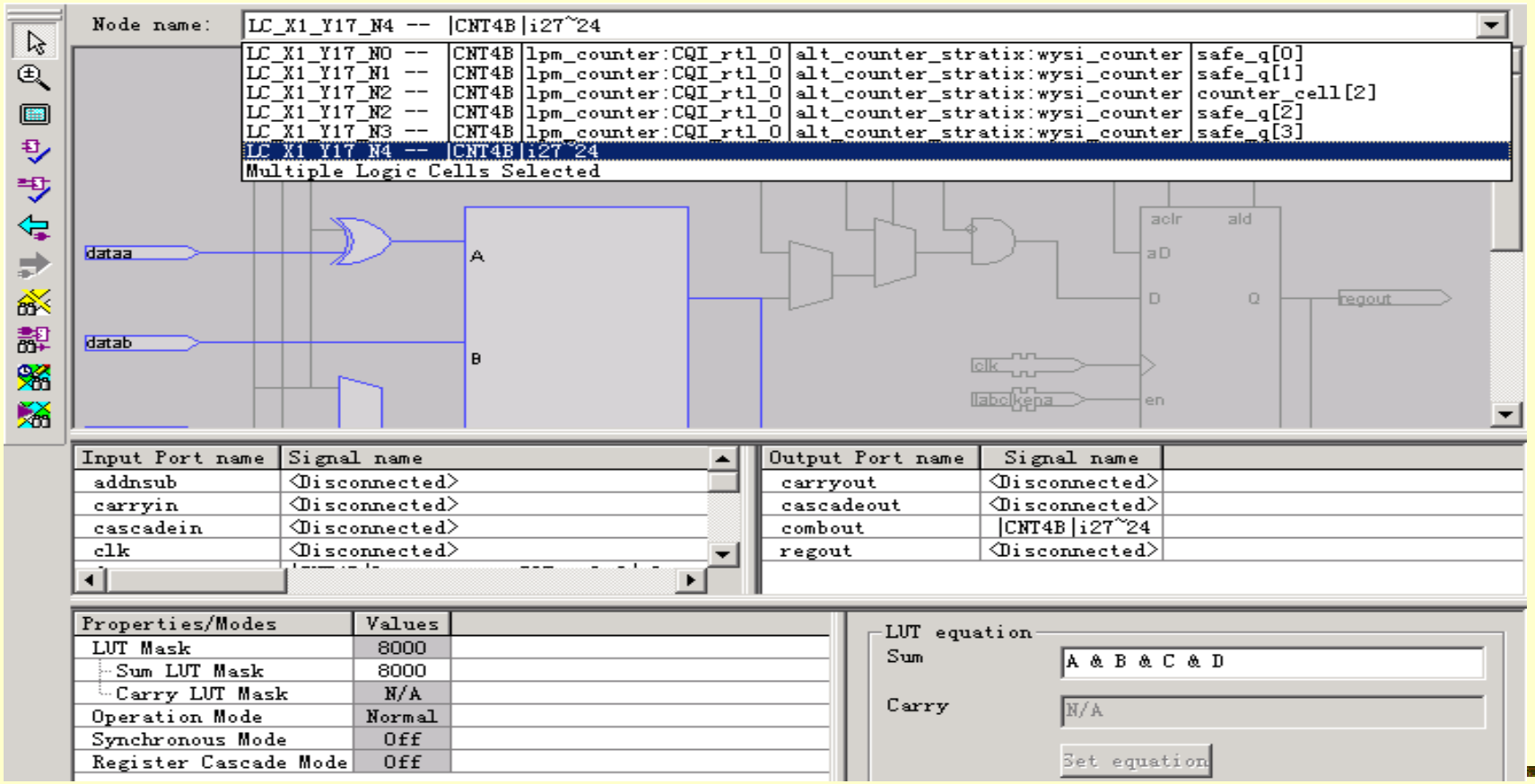


图11-31 打开属性和端口连接窗

11.4 Chip Editor应用

11.4.3 利用Change Manager检测底层逻辑

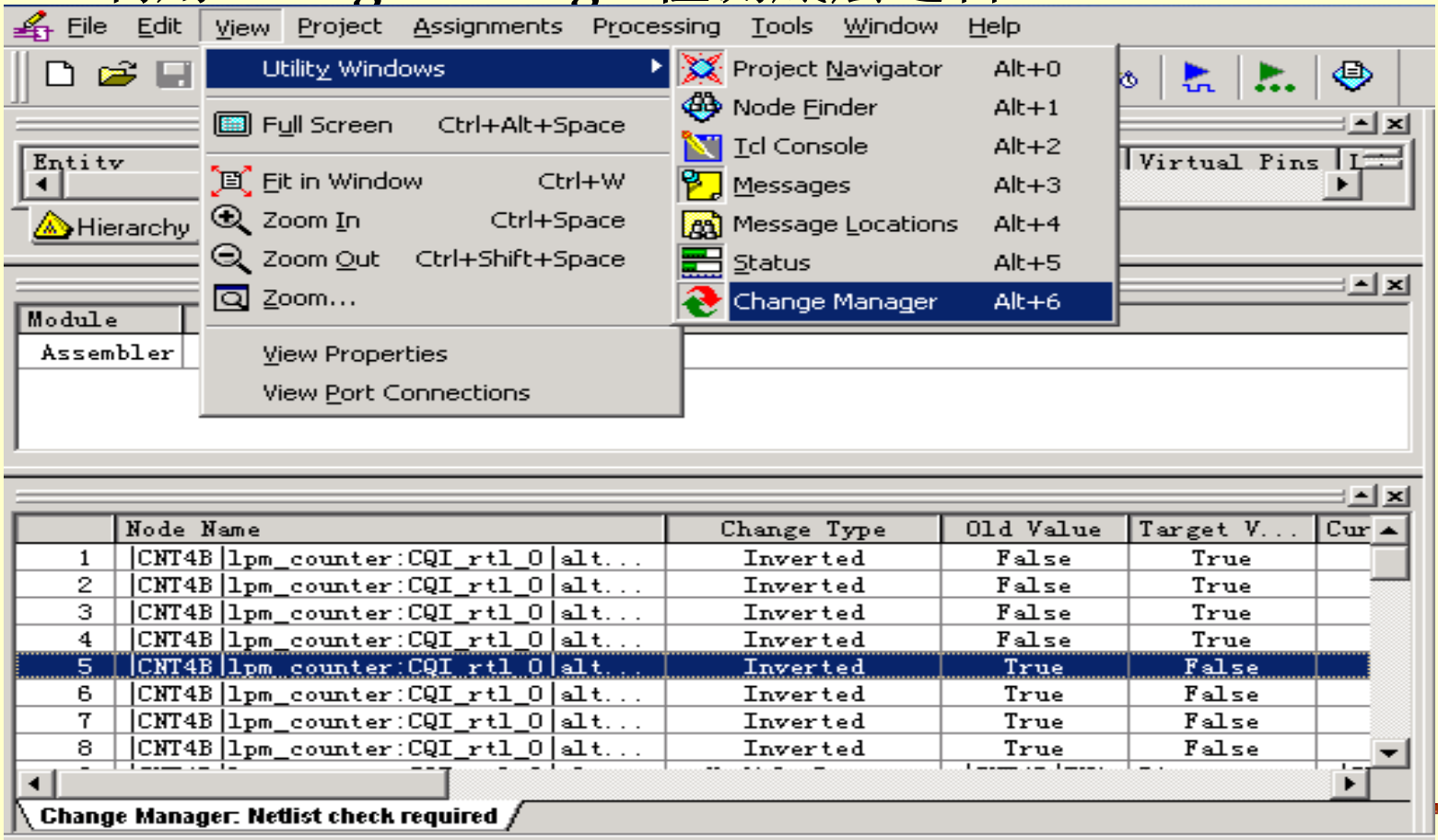


图11-32 打开Change Manager窗

康芯科技

习 题

11-1. 利用资源共享的面积优化方法对下面程序进行优化(仅要求在面积上优化)。习题程序如下：

【例11-10】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;
ENTITY addmux IS
    PORT (A,B,C,D   : IN  std_logic_vector(7 downto 0);
          sel       : IN  std_logic;
          Result    : OUT std_logic_vector(7 downto 0));
END addmux;
ARCHITECTURE rtl OF addmux IS
BEGIN
    process(A,B,C,D,sel)
    begin
        if(sel = '0') then  Result <= A + B;
                      else   Result <= C + D;
        end if;
    end process;
END rtl;
```

习 题

11-2. 试通过优化逻辑的方式对图**11-33**中所示的结构进行改进，给出**VHDL**代码和结构图。

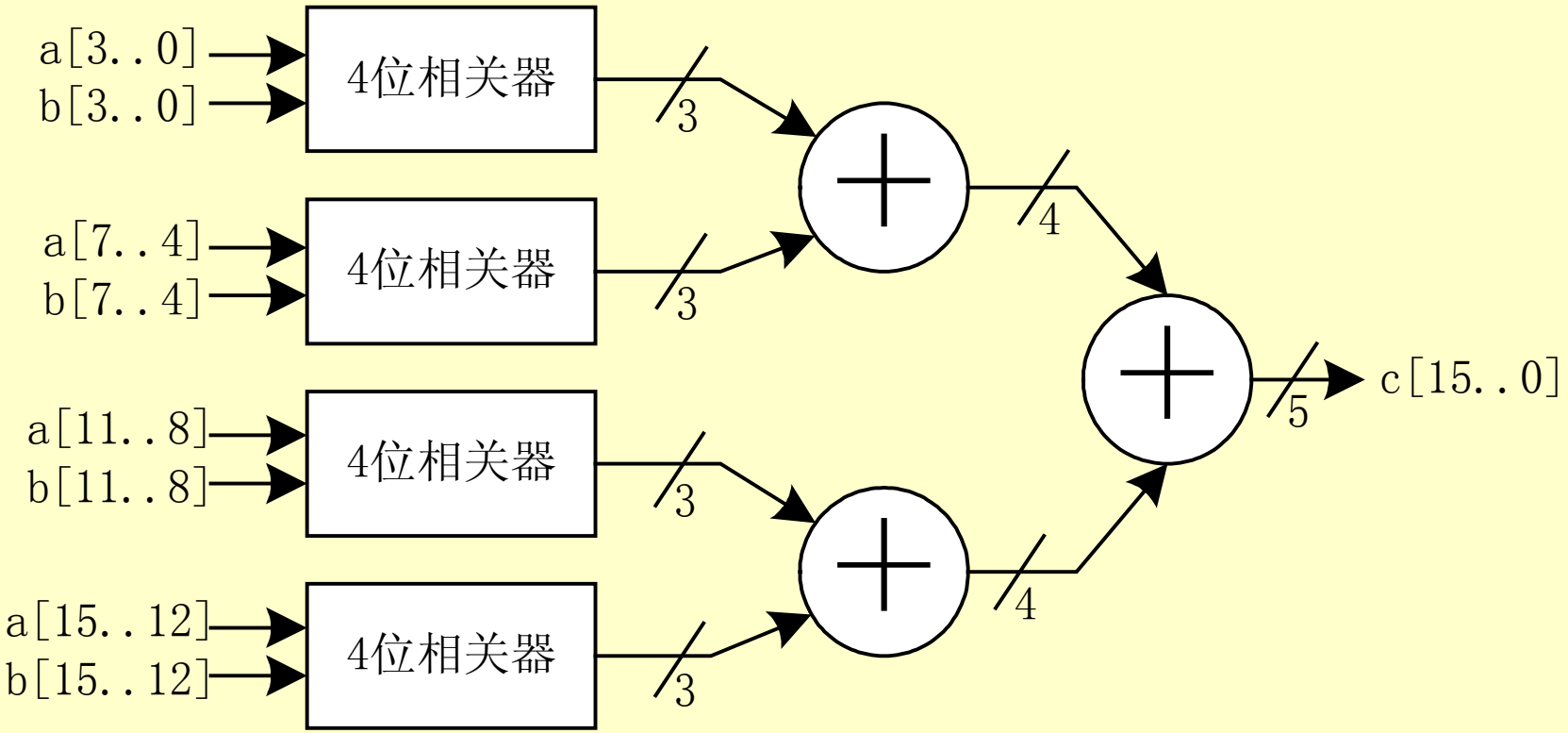


图11-33 习题11-2图

习 题

11-3. 已知4阶直接型FIR滤波器节的数学表达式如下：

$$y(n)=x(n)h(0)+x(n-1)+x(n-2)h(2)+x(n-3)h(3)$$

$x(n)$与$x(n-m)$，m=0，1，2，3是延迟关系，m表示延迟的clk数。$x(n-m)$与$h(m)$的位宽均为8位，$y(n)$为10位，其中$h(m)$在模块例化后为常数。该模块的输入为$x(n)$、clk，输出为$y(n)$，试实现该逻辑。

11-4. 对习题11-3中的FIR滤波器节在速度上进行优化(在$h(m)$固定的情况下)，试采用流水线技术。

11-5. 利用FPGA的LUT结构，构建资源占用较小的常数乘法器，改进习题11-3、11-4的设计，减少模块的资源使用。

11-6. 若对速度要求不高，但目标芯片的容量较小，试把习题11-3中的FIR滤波器用串行化的方式实现。

习 题

11-7. 设计一个连续乘法器，输入为**a0**，**a1**，**a2**，**a3**，位宽各为**8**位，输出**rout**为**32**位，完成**rout=a0*a1*a2*a3**，试实现之。

11-8. 对**11-7**进行优化，判断以下实现方法，那种方法更好？

rout=((a0*a1)*a2)*a3

rout=(a0*a1)*(a2*a3)

11-9. 为提高速度，对习题**11-8**中的前一种方法加上流水线技术进行实现。

11-10. 试对以上的习题解答通过设置**QuartusII**相关选项的方式，提高速度，减小面积。

实 验 与 设 计

11-1 采用流水线技术设计高速数字相关器

(1) 实验目的：设计一个在数字通信系统中常见的数字相关器，并利用流水线技术提高其工作速度，对其进行仿真和硬件测试。

(2) 实验原理：数字相关器用于检测等长度的两个数字序列间相等的位数，实现序列间的相关运算。

一位相关器，即异或门，异或的结果可以表示两个1位数据的相关程度。异或为0表示数据位相同；异或为1表示数据位不同。多位数字相关器可以由多个一位相关器构成，如N位的数字相关器由N个异或门和N个1位相关结果统计电路构成。

(3) 实验内容1：根据上述原理设计一个并行4位数字相关器(例9-17是示例程序)。

提示：利用CASE语句完成4个1位相关结果的统计。

实 验 与 设 计

11-1 采用流水线技术设计高速数字相关器

【例11-11】

```
stemp <= a XOR b;
PROCESS(stemp) BEGIN
        CASE stemp IS
         WHEN "0000" => c <= "100";
        --4
         WHEN "0001"|"0010"|"0100"|"1000" => c <= "011";
        --3
         WHEN "0011"|"0101"|"1001"|"0110"|"1010"|"1100" => c <= "010";      --2
                  WHEN "0111"|"1011"|"1101"|"1110" => c <= "001";            --1
                  WHEN "1111" => c <= "000";          -- 0;
                  WHEN OTHERS => c <= "000";
        END CASE;
END PROCESS;
```

实 验 与 设 计

11-1 采用流水线技术设计高速数字相关器

(4) 实验内容2：利用实验内容1中的4位数字相关器设计并行16位数字相关器。使用QuartusII估计最大延时，并计算可能运行的最高频率。

(5) 实验内容3：在以上实验的基础上，利用设计完成的4位数字相关器设计并行16位数字相关器，其结构框图可参考图11-33，并利用QuartusII计算运行速度。

(6) 实验内容4：上面的16位数字相关器是用3级组合逻辑实现的，在实际使用时，对其有高速的要求，试使用流水线技术改善其运行速度。在输入、输出及每一级组合逻辑的结果处加入流水线寄存器，提高速度，可参照本章内容进行设计。

(7) 实验思考题：考虑采用流水线后的运行速度与时钟clock的关系，测定输出与输入的总延迟。若输入序列是串行化的，数字相关器的结构如何设计？如何利用流水线技术提高其运行速度？

(8) 实验报告：根据以上的实验内容写出实验报告，包括设计原理、程序设计、程序分析、仿真分析、硬件测试和详细实验过程。

实 验 与 设 计

11-2 线性反馈移位寄存器设计

(1) 实验目的：学习用VHDL设计LFSR，掌握利用FPGA的特殊结构中高效实现LFSR的方法。

(2) 实验原理：LFSR即Linear Feedback Shift Register 线性反馈移位寄存器，是一种十分有用的时序逻辑结构，广泛用于伪随机序列发生、可编程分频器、CRC校验码生成、PN码等等。图11-34是典型的LFSR结构。由图中可以看出LFSR由移位寄存器加上xor构成，不同的xor决定了不同的生成多项式。图11-34中的生成多项式为$X^3+X^2+X^0$。

(3) 实验内容：依据图11-34设计一个LFSR，其生成多项式为$X^4+X^3+X^0$。试在FPGA上加以实现，并利用本章中提及的QuartusII优化选项，使之达到最高运行速度，并在GW48 EDA开发系统上，对其产生的码序列进行观察。

实 验 与 设 计

11-2 线性反馈移位寄存器设计

(4) 实验思考题1：另有一种LFSR的结构，见图11-35，试分析与图11-34中LFSR结构的异同点。

(5) 实验思考题2：对图11-35结构的LFSR电路进行改进，设计成串行CRC校验码发生器(提示：反馈线上加入xor，xor的一个输入端接待编码串行有效信息输入)。

(6) 实验报告：作出本项实验设计的完整电路图，详细说明其工作原理，完成测试实验内容，对实验中的码序列进行记录，写出电路可达到的最高运行速度及设置的QuartusII选项。

实 验 与 设 计

11-2 线性反馈移位寄存器设计

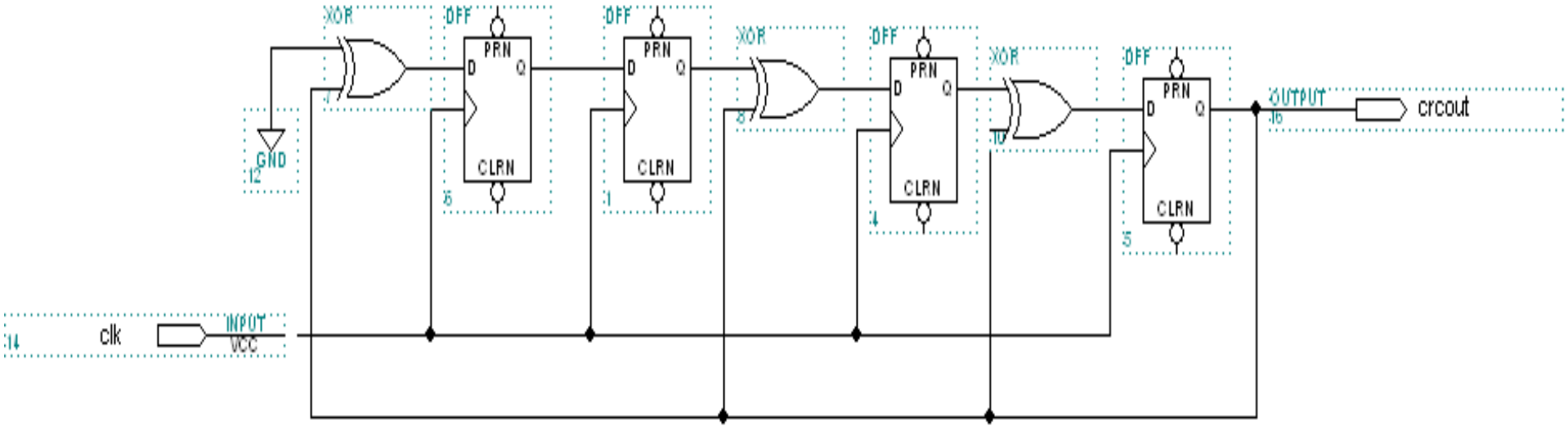


图11-34 LFSR举例

实 验 与 设 计

11-2 线性反馈移位寄存器设计

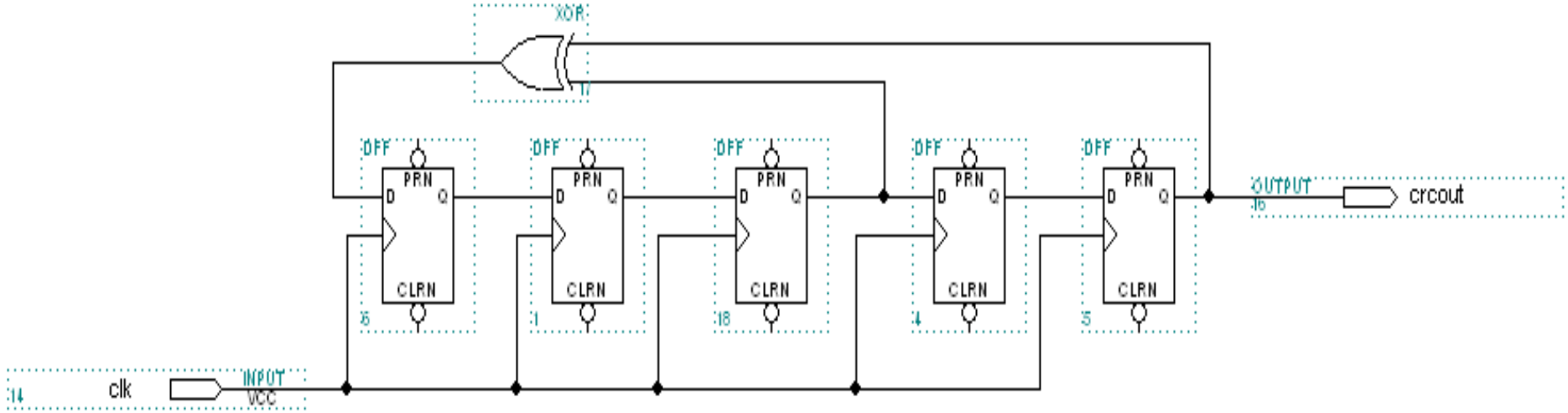


图11-35 另一种LFSR结构

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

(1) 实验目的：学习利用EDA技术和FPGA实现直接数字频率综合器DDS的设计。

(2) 实验原理：直接数字频率综合技术，即DDS技术，是一种新型的频率合成技术和信号产生方法。其电路系统具有较高的频率分辨率，可以实现快速的频率切换，并且在改变时能够保持相位的连续，很容易实现频率、相位和幅度的数控调制。

$$f_{SIN} = M（f_{clk}/2^n）$$ 11-1

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

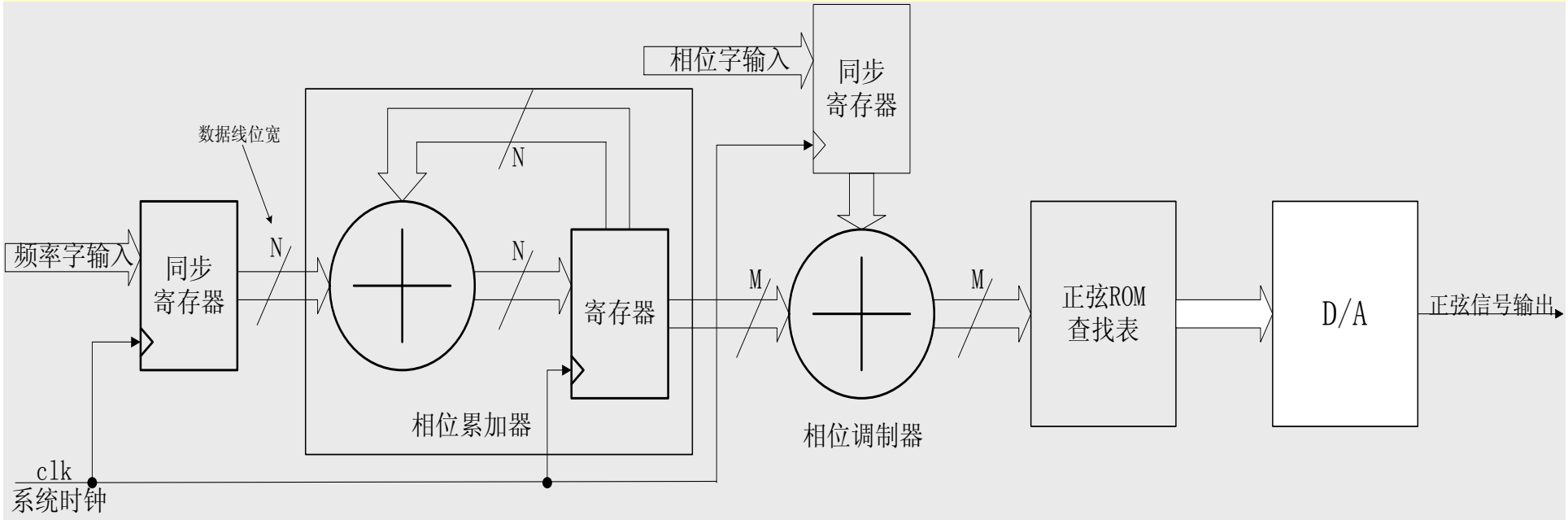
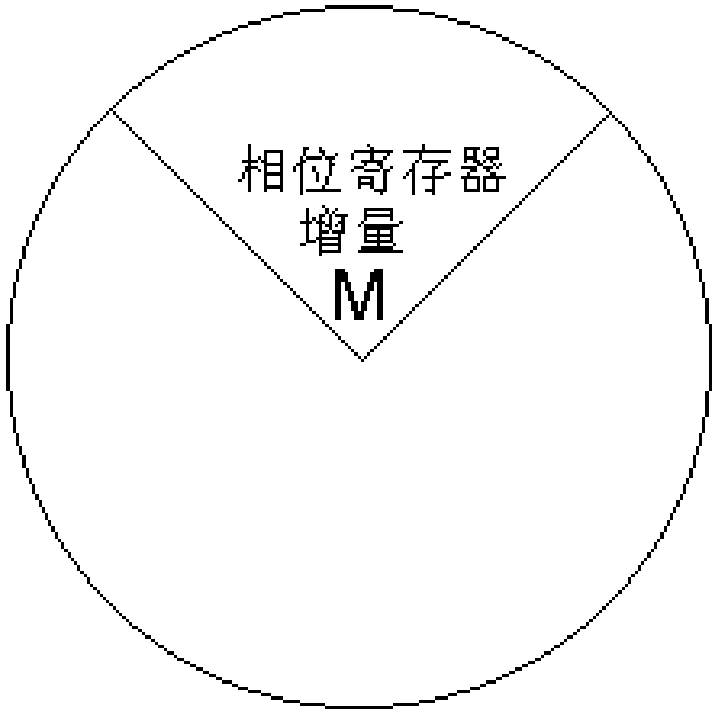


图11-36 DDS基本结构

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验



相位累加器宽	对应采样点数
8	256
12	4096
16	65536
20	1048576
24	16777216
28	268435456
32	4294967296

图11-37 相位累加器位宽和采样点关系

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

【例11-12】

```
LIBRARY ieee; --波形数据ROM
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
ENTITY data_rom IS
        PORT
        (       address         : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
                inclock         : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)     );
END data_rom;
 . . .
                init_file => "./data/LUT10X10.mif", --波形数据初始化文件路径
                lpm_hint => "ENABLE_RUNTIME_MOD=YES, INSTANCE_NAME=rom2",
. . .
END;
```

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

【例11-13】

```
LIBRARY IEEE;  --32位加法器模块
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ADDER32B IS
    PORT (  A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            S : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)     );
END ADDER32B;
ARCHITECTURE behav OF ADDER32B IS
    BEGIN
     S <= A + B;
END behav;
```

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

【例11-14】--32位寄存器模块

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY REG32B IS
    PORT (  Load : IN STD_LOGIC;
             DIN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END REG32B;
ARCHITECTURE behav OF REG32B IS
BEGIN
    PROCESS(Load, DIN)
   BEGIN
   IF Load'EVENT AND Load = '1' THEN     -- 时钟到来时，锁存输入数据
            DOUT <= DIN;
        END IF;
    END PROCESS;
END behav;
```

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

【例11-15】rom_data.mif 10位正弦波数据文件，读者可用MATLAB/DSP Builder生成

```
WIDTH=10;
DEPTH=1024;
ADDRESS_RADIX=DEC;
DATA_RADIX=DEC;
CONTENT BEGIN
  0 : 513; 1 : 515; 2 : 518; 3 : 521; 4 : 524; 5 : 527;  6 : 530; 7 : 533;
 8 : 537; 9 : 540; 10 : 543; 11 : 546; 13 : 549; 13 : 552; 14 : 555;
  . . . . . （略去部分数据）
  1018 : 493; 1019 : 496; 1020 : 499; 1021 : 502; 1022 : 505; 1023 : 508;
END;
```

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

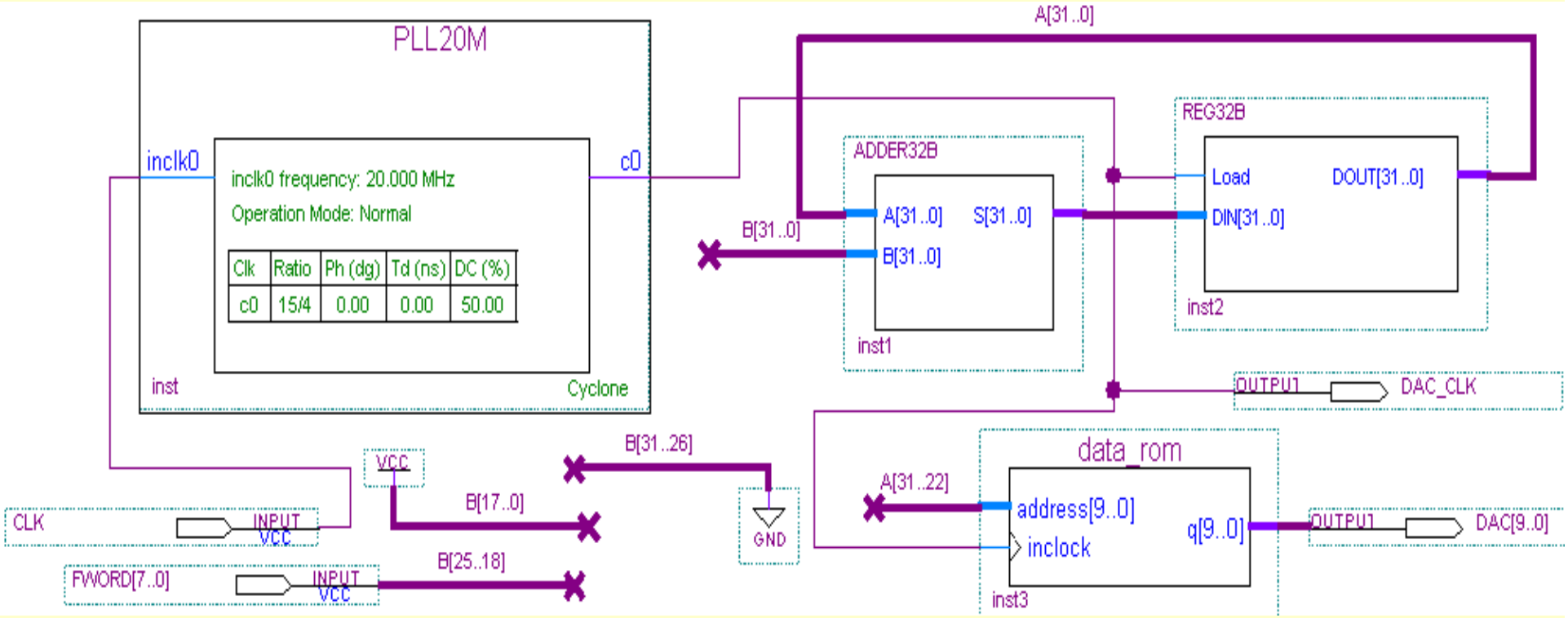


图11-38 DDS.vhd顶层原理图

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

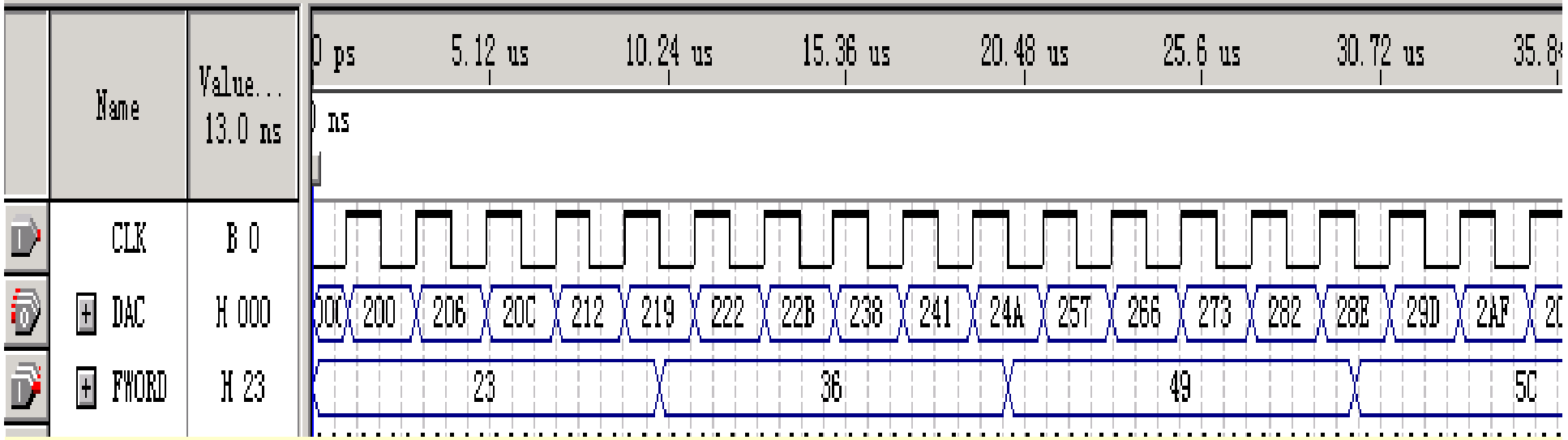


图11-39 DDS.vhd仿真波形

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

(3) 实验内容1： 详细述叙DDS的工作原理，依据例11-12至例11-15完成仿真，并由仿真结果进一步说明DDS的原理。完成编译和下载。选择模式1；键2、键1输入8位频率字FWORD；利用GW48系统ADDA板上的10位D/A5651输出波形，用示波器观察输出波形。

(4) 实验内容2： 根据图11-36，在原设计（图11-38）中加入相位控制电路，用键4、键3输入8位相位字PWORD；重复实验1的内容。

(5) 实验内容3： 将图11-38的顶层原理图表述为VHDL，重复实验1的内容。

(6) 实验内容4： 在图11-38的设计中增加一些元件，设计成扫频信号源，扫频速率、扫频频域、扫频步幅可设置。

实 验 与 设 计

11-3 直接数字式频率合成器(DDS)设计实验

(7) 实验内容5： 例11-12后的程序将32位频率字和10位相位字作了截断，都是8位。如果不作截断，修改其中的程序，并设法在GW48实验系统上完成实验（提示，增加2个锁存器与单片机通信）。

(8) 实验内容6： 将上例改成频率可数控的正交信号发生器，即使电路输出两路信号，且相互正交，一路为正弦(sin)信号，一路为余弦(cos)信号（此电路可用于正交方式的信号调制解调）。

(9) 实验内容7： 利用上例设计一个FSK信号发生器，并硬件实现之。

(10) 思考题： 如果不作截断，此例的频率精度和相位精度分别是多少？

实 验 与 设 计

11-4 基于DDS的数字移相信号发生器设计实验

(1) 实验原理：移相信号发生器是2003年大学生电子设计竞赛题中的一个设计项目。图11-40是基于DDS模型的数字移相信号发生器的电路模型图，示例程序如例11-16所示。

（2）实验内容1：完成10位输出数据宽度的移相信号发生器的设计，其中包括设计正弦波形数据MIF文件（数据深度1024、数据类型是十进制数）；给出仿真波形。最后进行硬件测试，对于GW48系统，推荐选择模式1：CLK接clock0，接13MHz；用键4、3控制相位字PWORD输入，键2、1控制频率字FWORD输入。观察它们的图形和李萨如图形。

（3）实验内容2：修改设计，增加幅度控制电路（如可以用一乘法器控制输出幅度）。

（4）实验内容3：将此信号发生器改成具有扫频功能的波形发生器，扫速可数控，点频扫频可控。

（5）实验思考题：如果频率控制字宽度直接用32位，相位控制字宽度直接用10位，输出仍为10位，时钟为20MHz，计算频率、相位和幅度三者分别的步进精度是多少，给出输出频率的上下限。

（6）实验报告：根据以上的实验要求、实验内容和实验思考题写出实验报告。

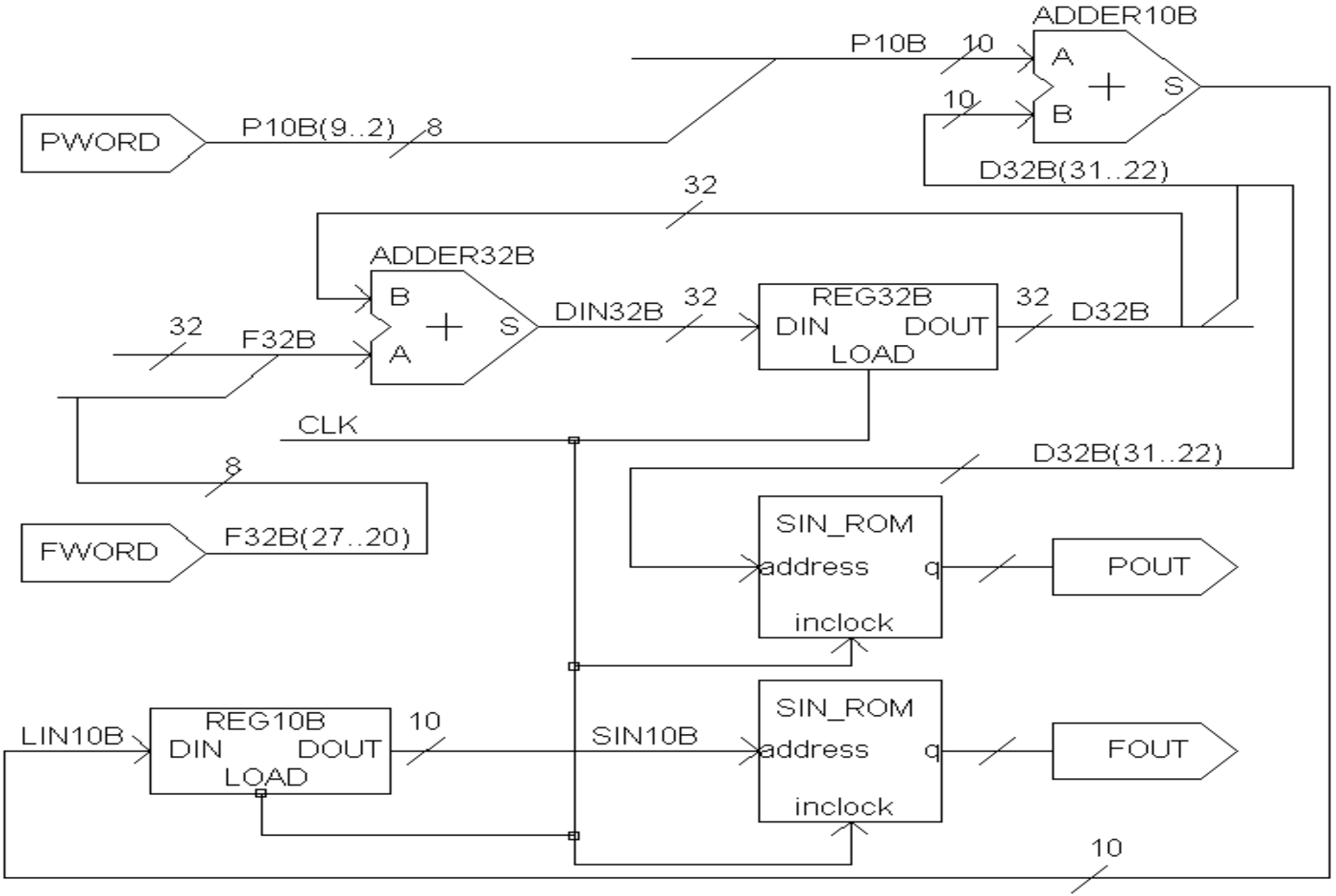


图11-40 数字移相信号发生器电路模型图

KX 康芯科技

【例11-16】数字移相信号发生器顶层设计文件，元件连接结构参考图11-40。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DDS_VHDL IS                                    -- 顶层设计
   PORT (  CLK : IN  STD_LOGIC;  --系统时钟
         FWORD : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);    --频率控制字
         PWORD : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);    --相位控制字
         FOUT  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);    --可移相正弦信号输出
         POUT  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)   ); --参考信号输出
 END;
ARCHITECTURE one OF DDS_VHDL IS
    COMPONENT REG32B                --32位锁存器
      PORT (  LOAD : IN STD_LOGIC;
              DIN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
             DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
   END COMPONENT;
    COMPONENT REG10B                --10位锁存器
      PORT (  LOAD :  IN STD_LOGIC;
              DIN :  IN STD_LOGIC_VECTOR(9 DOWNTO 0);
             DOUT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0) );
   END COMPONENT;
   COMPONENT ADDER32B               --32位加法器
     PORT (  A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
```

(接下页）

```
B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            S : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)     );
   END COMPONENT;
   COMPONENT ADDER10B                --10位加法器
     PORT (  A : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
             B : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
             S : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)     );
   END COMPONENT;
   COMPONENT SIN_ROM                 --10位地址10位数据正弦信号数据ROM
    PORT ( address   : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                       inclock              : IN STD_LOGIC ;
                    q             : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)        );
   END COMPONENT;
    SIGNAL F32B，D32B，DIN32B    : STD_LOGIC_VECTOR(31 DOWNTO 0);
    SIGNAL P10B，LIN10B，SIN10B  : STD_LOGIC_VECTOR( 9 DOWNTO 0);
 BEGIN
 F32B(27 DOWNTO 20)<=FWORD ;      F32B(31 DOWNTO 28)<="0000";
 F32B(19 DOWNTO 0)<="00000000000000000000" ;
 P10B( 9 DOWNTO 2)<=PWORD ;      P10B( 1 DOWNTO 0)<="00"  ;
 u1 : ADDER32B  PORT MAP( A=>F32B,B=>D32B, S=>DIN32B );
 u2 :   REG32B  PORT MAP( DOUT=>D32B,DIN=> DIN32B, LOAD=>CLK );
 u3 :  SIN_ROM  PORT MAP( address=>SIN10B, q=>FOUT, inclock=>CLK );
 u4 : ADDER10B  PORT MAP( A=>P10B,B=>D32B(31 DOWNTO 22),S=>LIN10B );
 u5 :   REG10B  PORT MAP( DOUT=>SIN10B,DIN=>LIN10B, LOAD=>CLK );
u6 :  SIN_ROM  PORT MAP( address=>D32B(31 DOWNTO 22), q=>POUT, inclock=>CLK );
END;
```

实 验 与 设 计

1-5 基于DDS的幅度调制AM信号发生器设计

（1）实验原理：幅度调制信号发生器是2005年大学生电子设计竞赛题中的一个设计项目。

（2）实验内容1：利用MATLAB和DSP Builder，根据图11-41，完成电路模型设计，使产生图11-42的波形，最后在FPGA上硬件实现，并于示波器上验证图11-42的波形。

（3）实验内容2：将图11-41模型完全用VHDL表达出来（顶层设计可以用原理图），直接用QuartusII实现硬件设计，仿真和FPGA实现，比较两种方法的异同点（如输出频率的高低，资源利用等）。

实 验 与 设 计

1-5 基于DDS的幅度调制AM信号发生器设计

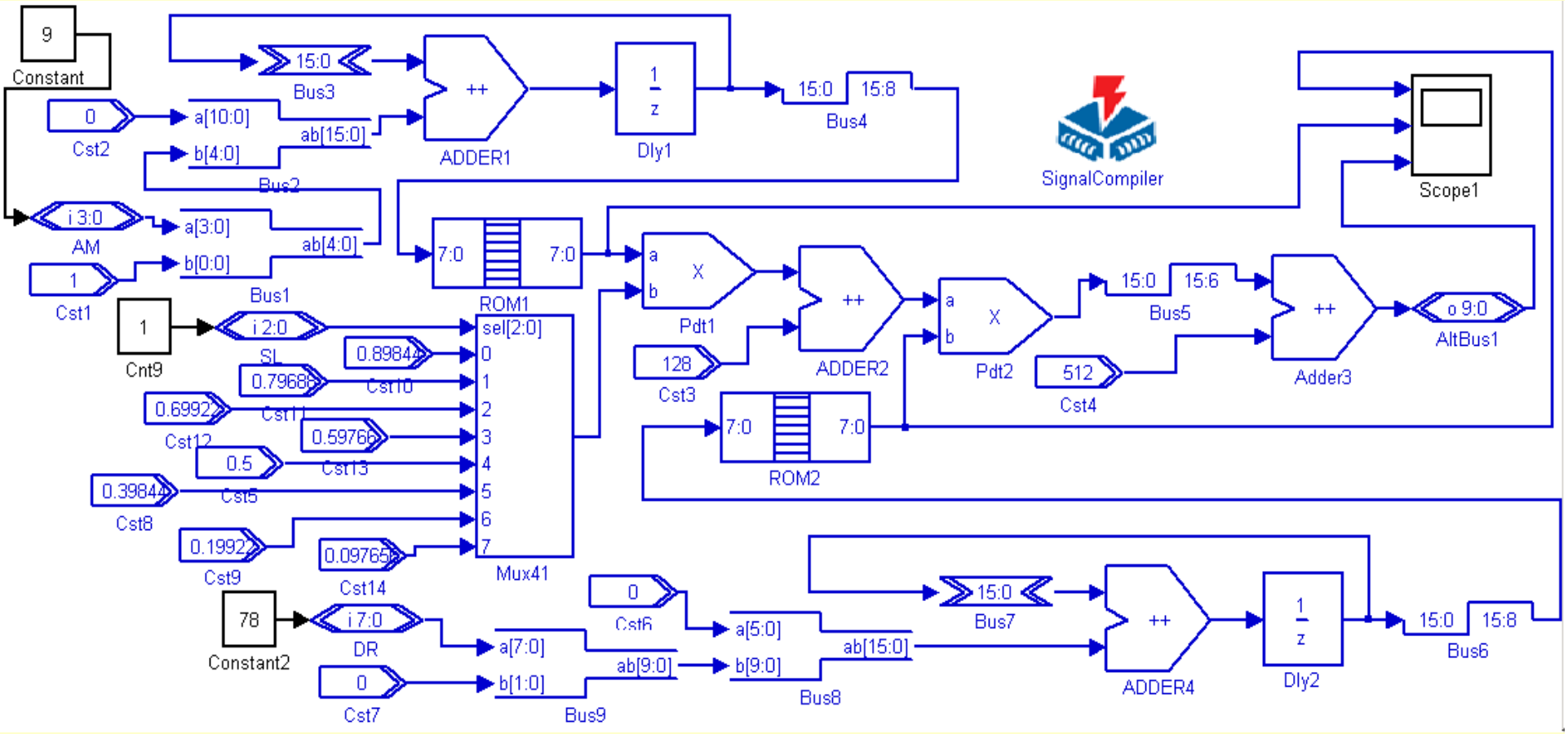


图11-41 AM发生器模型

康芯科技

实 验 与 设 计

1-5 基于DDS的幅度调制AM信号发生器设计

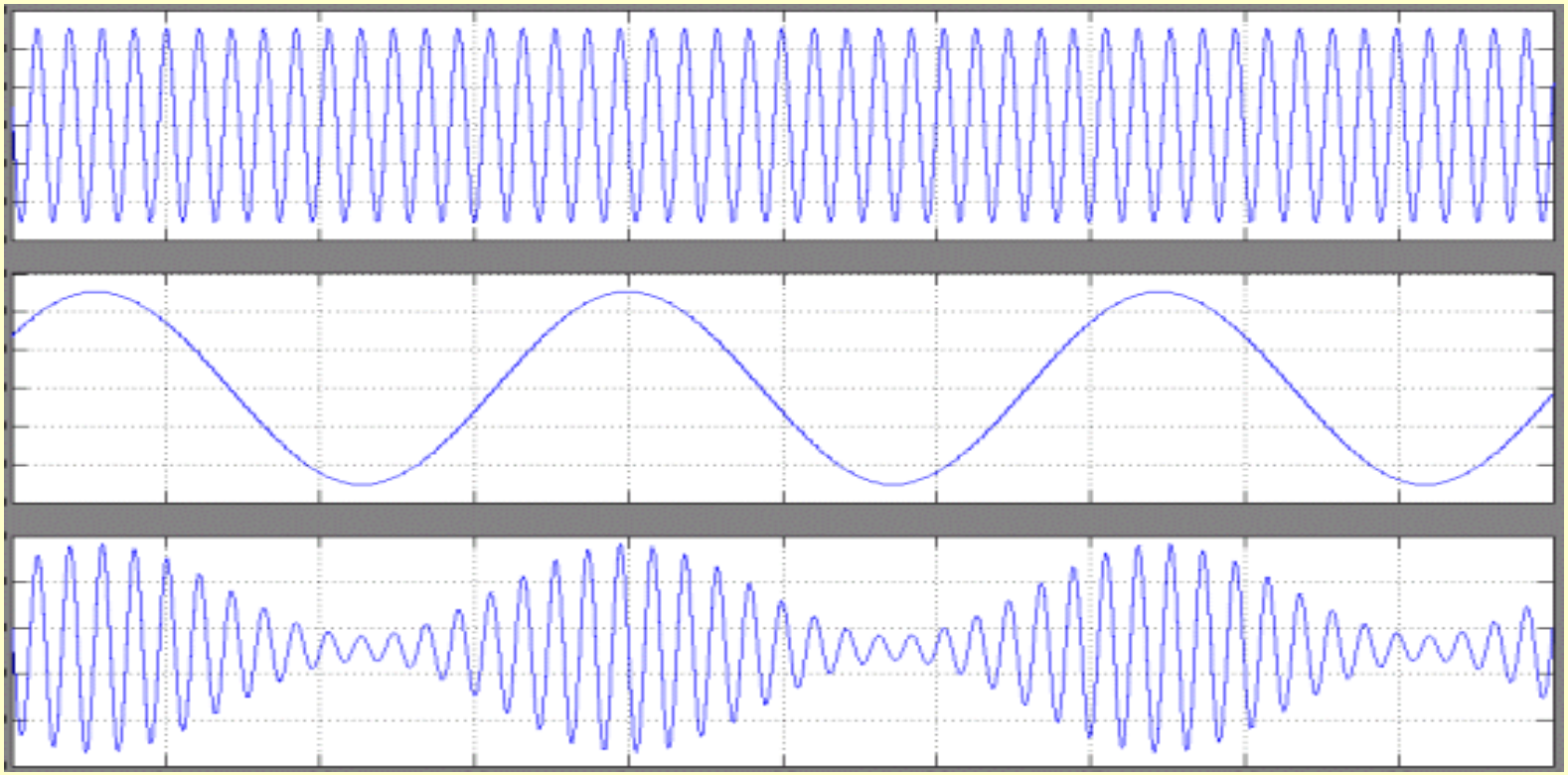


图11-42 AM模型仿真波形

11-6. 频率调制FM信号发生器设计

实验内容：根据实验11-15和图11-43，用VHDL设计频率调制信号发生器。要求载波频率、调制波频率和调制度可数控。

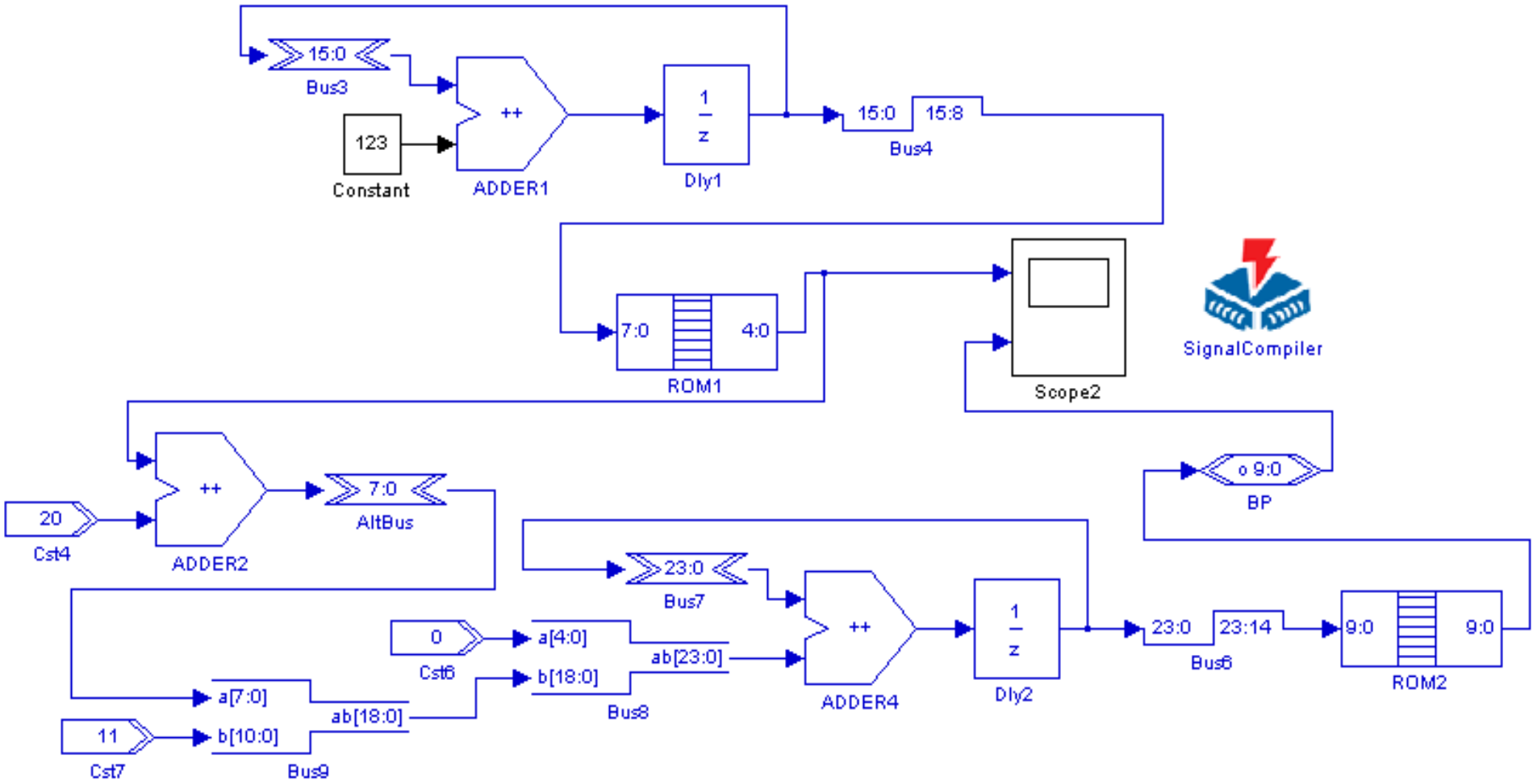


图11-43 频率调制信号发生器MATLAB模型

康芯科技

EDA 技术实用教程

第 10 章

VHDL基本语句

10.1 顺序语句

10.1.1 赋值语句

信号赋值语句　　变量赋值语句

10.1.2 IF语句

10.1.3 CASE语句

单个普通数值，如6。

数值选择范围，如(2 TO 4)。

并列数值，如3|5。

混合方式，以上三种方式的混合。

【例10-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux41 IS
PORT (s4, s3, s2, s1 : IN STD_LOGIC;
       z4, z3, z2, z1 : OUT STD_LOGIC);
END mux41;
ARCHITECTURE activ OF mux41 IS
SIGNAL sel : INTEGER RANGE 0 TO 15;
BEGIN
PROCESS (sel , s4, s3, s2, s1 )
  BEGIN
        sel<= 0 ;                                  -- 输入初始值
       IF (s1 ='1')  THEN sel <=  sel+1 ;
    ELSIF (s2 ='1')  THEN sel <= sel+2 ;
    ELSIF (s3 ='1')  THEN sel <= sel+4 ;
    ELSIF (s4 ='1')  THEN sel <= sel+8 ;
    ELSE NULL;                                  -- 注意，这里使用了空操作语句
  END IF ;
       z1<='0' ; z2<='0'; z3<='0'; z4<='0';   --输入初始值
CASE sel IS
  WHEN  0   => z1<='1' ;                        -- 当sel=0时选中
  WHEN 1|3 => z2<='1' ;                         -- 当sel为1或3时选中
  WHEN 4 To 7|2  => z3<='1';       -- 当sel为2、4、5、6或7时选中
  WHEN OTHERS => z4<='1' ;         -- 当sel为8～15中任一值时选中
 END CASE ;
END PROCESS ;
END activ ;
```

10.1 顺序语句

10.1.3 CASE语句

【例10-2】

```
SIGNAL value : INTEGER RANGE 0 TO 15;
SIGNAL  out1 : STD_LOGIC ;
   ...
  CASE value IS                                  -- 缺少以WHEN引导的条件句
  END CASE;
    ...
 CASE value IS
    WHEN 0 => out1<= '1' ;                       -- value2～15的值未包括进去
    WHEN 1 => out1<= '0' ;
      END CASE
    ...
      CASE value IS
    WHEN 0 TO 10 => out1<= '1';                  -- 选择值中5～10的值有重叠
    WHEN 5 TO 15 => out1<= '0';
END CASE;
```

【例10-3】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY alu IS
    PORT(   a, b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            opcode: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            result: OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
END alu;
ARCHITECTURE behave OF alu IS
    CONSTANT plus     : STD_LOGIC_VECTOR (1 DOWNTO 0) := b"00";
    CONSTANT minus    : STD_LOGIC_VECTOR (1 DOWNTO 0) := b"01";
    CONSTANT equal    : STD_LOGIC_VECTOR (1 DOWNTO 0) := b"10";
    CONSTANT not_equal: STD_LOGIC_VECTOR (1 DOWNTO 0) := b"11";
BEGIN
PROCESS (opcode,a,b)
BEGIN
    CASE opcode IS
        WHEN plus =>  result <= a + b;  -- a、b相加
        WHEN minus => result <= a - b;  -- a、b相减
        WHEN equal =>                       -- a、b相等
      IF (a = b) THEN  result <= x"01";
                  ELSE result <= x"00";
      END IF;
        WHEN not_equal =>                 -- a、b不相等
            IF (a /= b) THEN result <= x"01";
                        ELSE    result <= x"00";
            END IF;
    END CASE;
END PROCESS;
END behave;
```

10.1 顺序语句

10.1.4 LOOP语句

(1) 单个LOOP语句，其语法格式如下：

[LOOP标号：] LOOP

顺序语句

END LOOP [LOOP标号];

```
...
L2 : LOOP
      a := a+1;
     EXIT L2 WHEN a >10 ;        -- 当a大于10时跳出循环
      END LOOP L2;
...
```

10.1 顺序语句

10.1.4 LOOP语句

(2) FOR_LOOP语句，语法格式如下：

```
[LOOP标号：] FOR 循环变量，IN  循环次数范围  LOOP
        顺序语句
END LOOP [LOOP标号];
```

10.1 顺序语句

【例10-4】

```
 LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
ENTITY p_check IS
 PORT (  a : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
          y : OUT STD_LOGIC );
END p_check;
ARCHITECTURE opt OF p_check IS
  SIGNAL tmp : STD_LOGIC ;
BEGIN
  PROCESS(a)
    BEGIN
     tmp <='0';
     FOR n IN 0 TO 7 LOOP
      tmp <= tmp XOR a(n);
     END LOOP ;
      y <= tmp;
     END PROCESS;
  END opt;
```

10.1 顺序语句

10.1.4 LOOP语句

【例10-5】

```
    SIGNAL a, b, c : STD_LOGIC_VECTOR (1 TO 3);
...
FOR n IN 1 To 3 LOOP
a(n) <= b(n) AND c(n);
END LOOP;
```

此段程序等效于顺序执行以下三个信号赋值操作：

```
a(1)<=b(1) AND c(1);
a(2)<=b(2) AND c(2);
a(3)<=b(3) AND c(3);
```

10.1 顺序语句

10.1.5 NEXT语句

```
NEXT;                                    -- 第一种语句格式
NEXT LOOP标号;                           -- 第二种语句格式
NEXT LOOP标号 WHEN 条件表达式;           -- 第三种语句格式
```

【例10-6】

```
     ...
L1 : FOR cnt_value IN 1 TO 8 LOOP
s1 : a(cnt_value) := '0';
       NEXT WHEN (b=c);
s2 : a(cnt_value + 8 ):= '0';
END LOOP L1;
```

10.1 顺序语句

10.1.5 NEXT语句

【例10-7】

```
        ...
L_x : FOR cnt_value IN 1 TO 8 LOOP
 s1 :  a(cnt_value):= '0';
       k := 0;
L_y : LOOP
  s2 : b(k) := '0';
       NEXT L_x WHEN (e>f);
  s3 : b(k+8) := '0';
       k := k+1;
       NEXT LOOP L_y ;
       NEXT LOOP L_x ;
   …
```

10.1 顺序语句

10.1.6 EXIT语句

```
EXIT;                                    -- 第一种语句格式
EXIT LOOP标号;                           -- 第二种语句格式
EXIT LOOP标号 WHEN 条件表达式;           -- 第三种语句格式
```

10.1 顺序语句

10.1.6 EXIT语句

【例10-8】

```
SIGNAL a, b : STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL a_less_then_b : Boolean;
...
 a_less_then_b <= FALSE ;                    -- 设初始值
 FOR i IN 1 DOWNTO 0 LOOP
 IF (a(i)='1' AND b(i)='0') THEN
 a_less_then_b <= FALSE ;                    -- a > b
 EXIT ;
 ELSIF (a(i)='0' AND b(i)='1') THEN
 a_less_then_b <= TRUE ;                     -- a < b
    EXIT;
    ELSE    NULL;
    END IF;
 END LOOP;                  -- 当 i=1时返回LOOP语句继续比较
```

10.1 顺序语句

10.1.7 WAIT语句

```
WAIT;                                    -- 第一种语句格式
WAIT ON 信号表;                          -- 第二种语句格式
WAIT UNTIL 条件表达式;                   -- 第三种语句格式
WAIT FOR 时间表达式;     -- 第四种语句格式，超时等待语句
```

10.1 顺序语句

10.1.7 WAIT语句

【例10-9】

```
SIGNAL s1,s2 : STD_LOGIC;
...
PROCESS
BEGIN
...
WAIT ON s1,s2 ;
END PROCESS ;
```

【例10-10】

(a) WAIT_UNTIL结构

```
...
Wait until enable ='1';
...
```

(b) WAIT_ON结构

```
LOOP
  Wait on enable;
EXIT WHEN enable ='1';
END LOOP;
```

10.1 顺序语句

10.1.7 WAIT语句

```
WAIT UNTIL  信号=Value ;                              -- (1)
WAIT UNTIL  信号'EVENT AND 信号=Value;                -- (2)
WAIT UNTIL  NOT 信号'STABLE AND 信号=Value;           -- (3)
```

```
WAIT UNTIL clock ='1';
WAIT UNTIL rising_edge(clock)  ;
WAIT UNTIL NOT clock'STABLE AND clock ='1';
WAIT UNTIL clock ='1' AND clock'EVENT;
```

10.1 顺序语句

10.1.7 WAIT语句

【例10-11】

```
PROCESS
BEGIN
WAIT UNTIL clk ='1';
ave <= a;
WAIT UNTIL clk ='1';
ave <= ave + a;
WAIT UNTIL clk ='1';
ave <= ave + a;
WAIT UNTIL clk ='1';
ave <= (ave + a)/4 ;
END PROCESS ;
```

10.1 顺序语句

10.1.7 WAIT语句

【例10-12】

```
    PROCESS
    BEGIN
     rst_loop :  LOOP
      WAIT UNTIL clock ='1' AND clock'EVENT;    -- 等待时钟信号
      NEXT rst_loop WHEN (rst='1');             -- 检测复位信号rst
      x <= a ;                                  -- 无复位信号，执行赋值操作
      WAIT UNTIL clock ='1' AND clock'EVENT;    -- 等待时钟信号
      NEXT rst_loop When (rst='1');             -- 检测复位信号rst
      y <= b ;                                  -- 无复位信号，执行赋值操作
      END LOOP rst_loop ;
END PROCESS;
```

【例10-13】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY shifter IS
    PORT ( data : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
           shift_left: IN STD_LOGIC;
           shift_right: IN STD_LOGIC;
           clk: IN STD_LOGIC;
           reset : IN STD_LOGIC;
           mode : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
           qout : BUFFER STD_LOGIC_VECTOR (7 DOWNTO 0) );
END shifter;
ARCHITECTURE behave OF shifter IS
  SIGNAL enable: STD_LOGIC;
  BEGIN
  PROCESS
  BEGIN
   WAIT UNTIL (RISING_EDGE(clk) );    --等待时钟上升沿
    IF (reset = '1') THEN    qout <= "00000000";
     ELSE   CASE mode IS
       WHEN "01" => qout<=shift_right & qout(7 DOWNTO 1);--右移
       WHEN "10" => qout<=qout(6 DOWNTO 0) & shift_left; --左移
         WHEN "11" => qout <= data;                    -- 并行加载
         WHEN OTHERS => NULL;
        END CASE;
      END IF;
  END PROCESS;
END behave;
```

10.1 顺序语句

10.1.8 子程序调用语句

1. 过程调用

过程名[([形参名=>]实参表达式
{，[形参名=>]实参表达式})];

【例10-14】

```
PACKAGE data_types IS                                    -- 定义程序包
SUBTYPE data_element IS INTEGER RANGE 0 TO 3 ; -- 定义数据类型
TYPE data_array IS ARRAY (1 TO 3) OF data_element;
END data_types;
USE WORK.data_types.ALL;   --打开以上建立在当前工作库的程序包data_types
ENTITY sort IS
    PORT ( in_array : IN  data_array ;
           out_array : OUT data_array);
END sort;
  ARCHITECTURE exmp OF sort IS
  BEGIN
```

(接下页)

```
PROCESS (in_array)                          -- 进程开始，设data_types为敏感信号
   PROCEDURE swap(data : INOUT data_array;
                                            -- swap的形参名为data、low、high
                low, high :    IN INTEGER ) IS
    VARIABLE       temp :  data_element ;
    BEGIN                                          -- 开始描述本过程的逻辑功能
       IF (data(low) > data(high)) THEN  -- 检测数据
             temp := data(low) ;
        data(low) := data(high);
       data(high) := temp ;
       END IF ;
      END swap ;                                    -- 过程swap定义结束
   VARIABLE my_array : data_array ;          -- 在本进程中定义变量my_array
   BEGIN                                          -- 进程开始
    my_array := in_array ;                    -- 将输入值读入变量
    swap(my_array, 1, 2);
    -- my_array、1、2是对应于data、low、high的实参
    swap(my_array, 2, 3);                -- 位置关联法调用，  第2、第3元素交换
    swap(my_array, 1, 2);               -- 位置关联法调用，  第1、第2元素再次交换
     out_array <= my_array ;
 END Process ;
END exmp ;
```

【例10-15】

```
ENTITY sort4 is
GENERIC (top : INTEGER :=3);
    PORT (a, b, c, d : IN BIT_VECTOR (0 TO top);
      ra, rb, rc, rd : OUT BIT_VECTOR (0 TO top));
END sort4;
ARCHITECTURE muxes OF sort4 IS
PROCEDURE sort2(x, y : INOUT BIT_VECTOR (0 TO top)) is
    VARIABLE tmp : BIT_VECTOR (0 TO top);
BEGIN
    IF x > y THEN  tmp := x;  x := y;     y := tmp;
    END IF;
END sort2;
BEGIN
 PROCESS (a, b, c, d)
    VARIABLE va, vb, vc, vd : BIT_VECTOR(0 TO top);
BEGIN
    va := a;  vb := b; vc := c;  vd := d;
    sort2(va, vc);
    sort2(vb, vd);
    sort2(va, vb);
    sort2(vc, vd);
    sort2(vb, vc);
    ra <= va;      rb <= vb;  rc <= vc;  rd <= vd;
    END PROCESS;
END muxes;
```

2、函数调用

KX 康心科技

10.1 顺序语句

10.1.9 RETURN语句

```
RETURN;                -- 第一种语句格式
RETURN 表达式;         -- 第二种语句格式
```

【例10-16】

```
PROCEDURE rs (SIGNAL s ,  r :  IN   STD_LOGIC ;
              SIGNAL q , nq : INOUT STD_LOGIC) IS
 BEGIN
  IF ( s ='1' AND r ='1') THEN
   REPORT "Forbidden state : s and r are quual to '1'";
   RETURN ;
   ELSE
   q <= s AND nq AFTER 5 ns ;
  nq <= s AND  q AFTER 5 ns ;
  END IF ;
END PROCEDURE rs ;
```

10.1 顺序语句

【例10-17】

```
    FUNCTION opt (a, b, opr :STD_LOGIC)  RETURN  STD_LOGIC IS
  BEGIN
  IF (opr ='1') THEN   RETURN (a AND b);
                    ELSE  RETURN (a OR b) ;
  END IF ;
END FUNCTION opt ;
```

10.1.10 空操作语句

```
CASE Opcode IS
WHEN  "001" =>  tmp := rega AND regb ;
WHEN  "101" =>  tmp := rega OR regb ;
WHEN  "110" =>  tmp := NOT rega ;
WHEN OTHERS  =>  NULL ;
END CASE ;
```

10.2 并行语句

- 并行信号赋值语句(Concurrent Signal Assignments)。
- 进程语句(Process Statements)。
- 块语句(Block Statements)。
- 条件信号赋值语句(Selected Signal Assignments)。
- 元件例化语句(Component Instantiations)，其中包括类属配置语句。
- 生成语句(Generate Statements)。
- 并行过程调用语句(Concurrent Procedure Calls)。

```
ARCHITECTURE   结构体名   OF   实体名   IS
      说明语句
       BEGIN
       并行语句
END ARCHITECTURE 结构体名
```

10.2 并行语句

10.2.1 并行信号赋值语句

1. 简单信号赋值语句

赋值目标 <= 表达式

```
ARCHITECTURE curt OF bc1 IS
SIGNAL s1, e, f, g, h : STD_LOGIC ;
BEGIN
    output1 <= a AND b ;
    output2 <= c + d ;
    g <= e OR f ;
    h <= e XOR f ;
    s1 <= g ;
END ARCHITECTURE curt;
```

10.2 并行语句

10.2.1 并行信号赋值语句

2. 条件信号赋值语句

```
赋值目标 <= 表达式 WHEN 赋值条件 ELSE
表达式 WHEN 赋值条件 ELSE
                    ...
        表达式；
```

【例10-18】

```
ENTITY  mux IS
  PORT ( a,b,c : IN BIT ;
         p1,p2 : IN BIT ;
          z    : OUT BIT );
END;
ARCHITECTURE behv OF mux IS
 BEGIN
   z  <= a WHEN p1 = '1' ELSE
         b WHEN p2 = '1' ELSE
         c  ;
END;
```

10.2 并行语句

10.2.1 并行信号赋值语句

2. 条件信号赋值语句

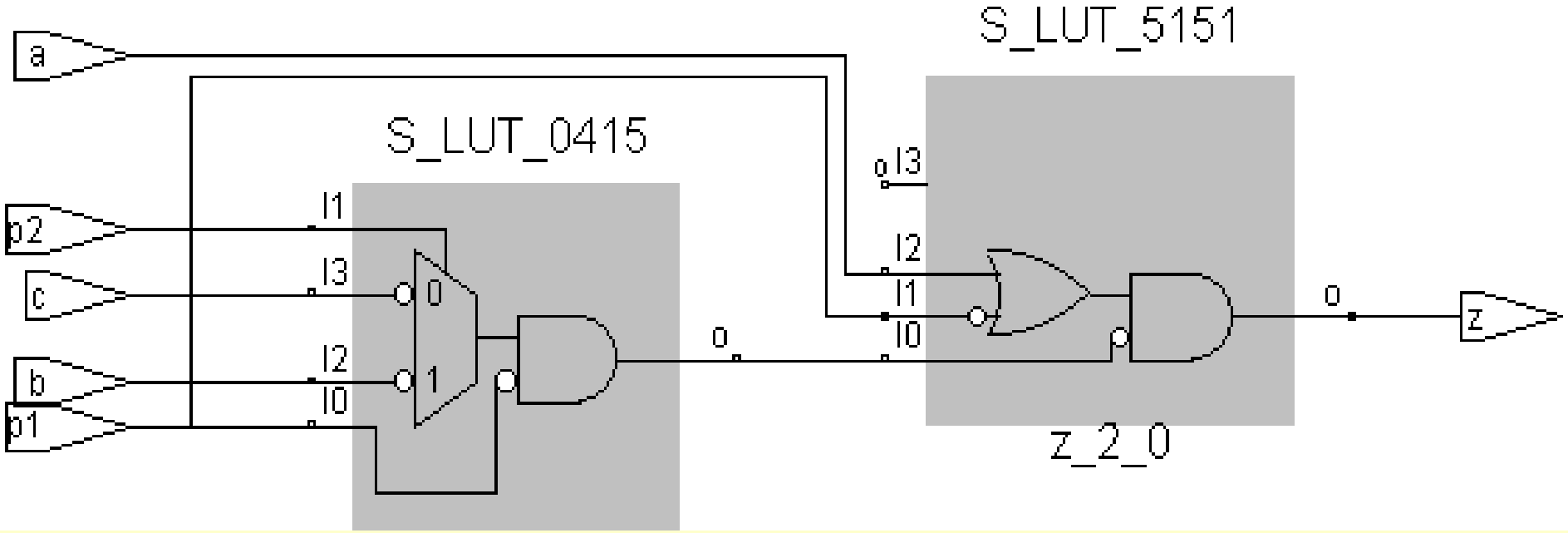


图10-1 例10-18的RTL电路图（Synplify综合）

10.2 并行语句

10.2.1 并行信号赋值语句

3. 选择信号赋值语句

```
WITH 选择表达式 SELECT
赋值目标信号 <= 表达式 WHEN 选择值
表达式 WHEN 选择值
...
表达式 WHEN 选择值;
```

【例10-19】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY decoder IS
   PORT (  a,  b,  c  :   IN STD_LOGIC;
         data1, data2 :   IN STD_LOGIC;
             dataout   : OUT STD_LOGIC  );
END decoder;
ARCHITECTURE concunt OF decoder IS
    SIGNAL instruction : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
    BEGIN
 instruction <= c & b & a ;
 WITH instruction SELECT
 dataout <= data1  AND data2  WHEN "000" ,
            data1  OR  data2  WHEN "001" ,
            data1 NAND data2  WHEN "010" ,
            data1 NOR  data2  WHEN "011" ,
            data1 XOR  data2  WHEN "100" ,
            data1 XNOR data2  WHEN "101" ,
                        'Z'   WHEN OTHERS ;
END concunt;
```

10.2 并行语句

10.2.1 并行信号赋值语句

3. 选择信号赋值语句

```
...
WITH selt SELECT
muxout <= a WHEN   0|1  ,      -- 0或1
          b WHEN 2 TO 5 ,      -- 2或3，或4或5
          c WHEN    6   ,
          d WHEN    7   ,
         'Z' WHEN OTHERS ;
   …
```

10.2 并行语句

10.2.2 块语句结构

BLOCK 语句的表达格式如下：

```
块标号   : BLOCK [(块保护表达式)]
          接口说明
              类属说明
              BEGIN
          并行语句
END BLOCK 块标号 ;
```

【例10-20】

```
    ...
   ENTITY gat IS
  GENERIC(l_time : TIME ;  s_time : TIME ) ; -- （参数传递）类属说明
    PORT (b1, b2, b3 : INOUT BIT) ;          -- 结构体全局端口定义
    END ENTITY gat  ;
    ARCHITECTURE func OF gat IS
    SIGNAL  a1 : BIT ;                       -- 结构体全局信号 a1定义
    BEGIN
Blk1 : BLOCK                                 -- 块定义，块标号名是blk1
    GENERIC (gb1, gb2 : Time) ;             -- 定义块中的局部类属参量
    GENERIC MAP (gb1 => l_time,gb2 => s_time); -- 局部端口参量设定
    PORT (pb : IN BIT; pb2 : INOUT BIT );  -- 块结构中局部端口定义
    PORT MAP (pb1 => b1, pb2 => a1 ) ;       -- 块结构端口连接说明
    CONSTANT delay : Time := 1 ms ;             -- 局部常数定义
    SIGNAL s1 : BIT ;                           -- 局部信号定义
    BEGIN
    s1 <= pb1 AFTER delay ;
    pb2 <= s1 AFTER gb1, b1 AFTER gb2 ;
    END BLOCK blk1 ;
 END ARCHITECTURE func ;
```

KX 康芯科技

10.2 并行语句

10.2.2 块语句结构

【例10-21】

```
   ...
b1 :  BLOCK
      SIGNAL s1: BIT ;
      BEGIN
 S1 <= a AND b ;
b2 :  BLOCK
 SIGNAL s2: BIT ;
 BEGIN
   s2 <= c AND d ;
   b3 : BLOCK
    BEGIN
       Z <= s2 ;
    END BLOCK b3 ;
 END BLOCK b2 ;
   y <= s1 ;
   END BLOCK b1 ;
```

…

【例10-22】

```
LIBRARY IEEE;
 USE IEEE. std_logic_1164.ALL;
  ENTITY f_adder IS
    PORT ( ain, bin , cin : IN std_logic;
          sum, cout : OUT std_logic );
  END  f_adder;
  ARCHITECTURE e_ad OF f_adder IS
   SIGNAL so1, co1, co2 : std_logic;
  BEGIN
    h_adder1 : BLOCK          --半加器u1
      BEGIN
       PROCESS( ain,bin )
        BEGIN
        so1<=NOT(ain XOR (NOT bin)); co1<= ain AND bin;
       END PROCESS;
  END BLOCK   h_adder1;
h_adder2: BLOCK               --半加器u2
      SIGNAL so2 : std_logic;
      BEGIN
   so2 <= NOT(so1 XOR (NOT cin)) ; co2<=so1 and cin ; sum<=so2;
END BLOCK h_adder2;
  or2 : BLOCK                  --或门u3
         BEGIN
      PROCESS (co2, co1)
          BEGIN
        cout<= co2 OR co1;
END PROCESS;
END BLOCK or2;
     END e_ad;
```

10.2 并行语句

10.2.2 块语句结构

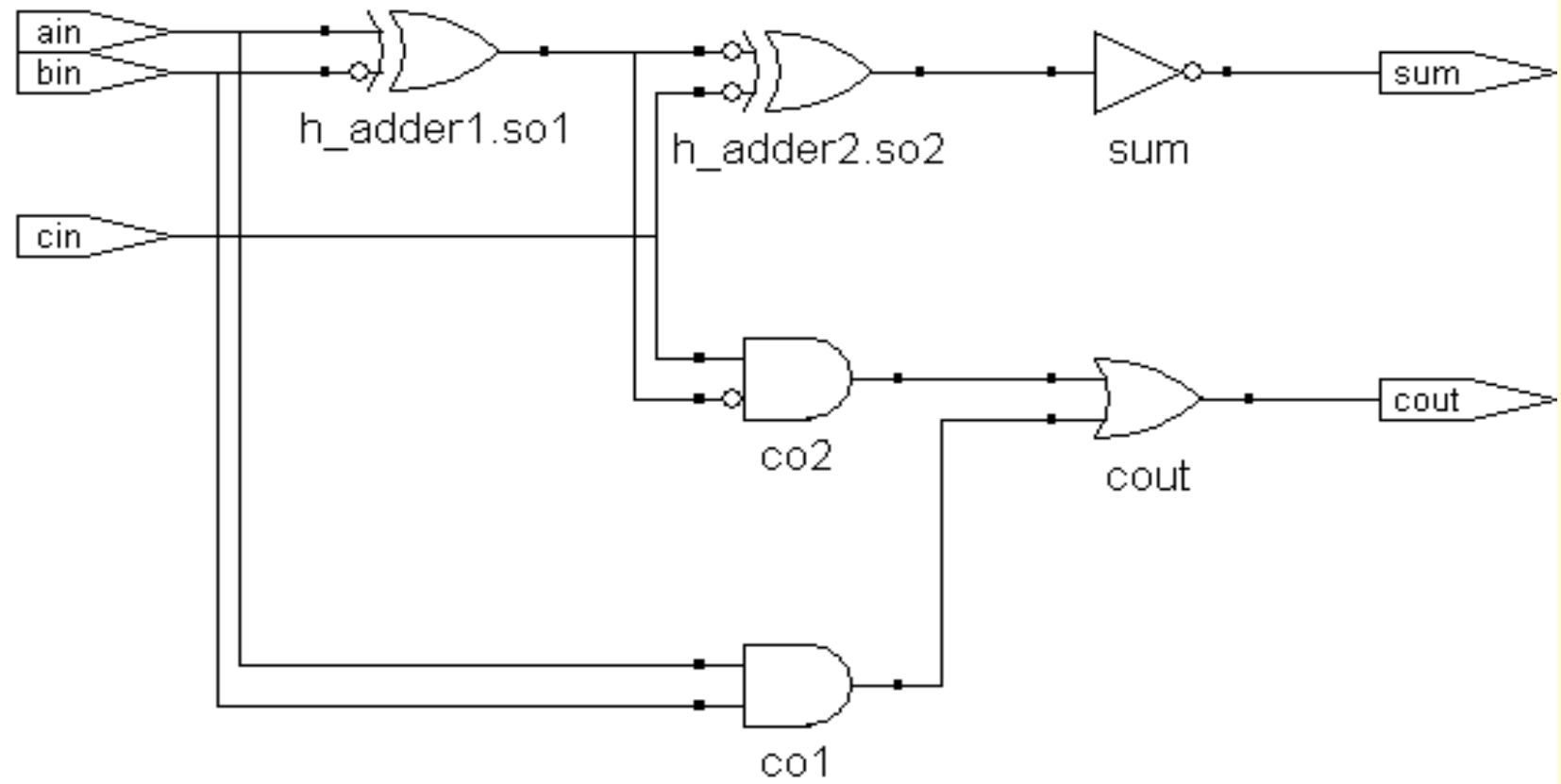


图10-2 例10-22的RTL电路图（Synplify综合）

10.2 并行语句

10.2.3 并行过程调用语句

过程名(关联参量名);

【例10-23】

```
...
   PROCEDURE adder(SIGNAL a, b :IN STD_LOGIC ;  --过程名为adder
                   SIGNAL sum : OUT STD_LOGIC );
   ...
   adder(a1，b1，sum1) ;                          -- 并行过程调用
   ...                    -- 在此，a1、b1、sum1即为分别对应于a、b、sum
的关联参量名
   PROCESS( c1，c2) ;                             -- 进程语句执行
   BEGIN
   Adder(c1，c2，s1) ;      -- 顺序过程调用，在此c1、c2、s1即为分别对
END PROCESS ;          -- 应于a、b、sum的关联参量名
```

10.2 并行语句

10.2.3 并行过程调用语句

【例10-24】

```
  PROCEDURE check(SIGNAL a : IN  STD_LOGIC_VECTOR;        -- 在调用时
                  SIGNAL error : OUT BOOLEAN ) IS         -- 再定位宽
  VARIABLE found_one : BOOLEAN := FALSE ;                 -- 设初始值
  BEGIN
  FOR i IN a'RANGE LOOP       -- 对位矢量a的所有的位元素进行循环检测
  IF a(i) = '1' THEN           -- 发现a中有 '1'
  IF found_one THEN      -- 若found_one为TRUE，则表明发现了一个以上的'1'
    ERROR <= TRUE;       -- 发现了一个以上的'1'，令found_one为TRUE
       RETURN;            -- 结束过程
  END IF;
  Found_one := TRUE;    -- 在a中已发现了一个'1'
  End IF;
  End LOOP;                -- 再测a中的其他位
  error <= NOT found_one; -- 如果没有任何'1' 被发现，error 将被置TRUE
END PROCEDURE check;
```

10.2 并行语句

10.2.3 并行过程调用语句

```
...
    CHBLK：BLOCK
  SIGNAL s1: STD_LOGIC_VECTOR (0 TO 0);    -- 过程调用前设定位矢尺寸
  SIGNAL s2: STD_LOGIC_VECTOR (0 TO 1);
  SIGNAL s3: STD_LOGIC_VECTOR (0 TO 2);
  SIGNAL s4: STD_LOGIC_VECTOR (0 TO 3);
  SIGNAL e1, e2, e3, e4: Boolean;
  BEGIN
   Check (s1, e1);                  -- 并行过程调用，关联参数名为s1、e1
   Check (s2, e2);                  -- 并行过程调用，关联参数名为s2、e2
   Check (s3, e3);                  -- 并行过程调用，关联参数名为s3、e3
   Check (s4, e4);                  -- 并行过程调用，关联参数名为s4、e4
    END  BLOCK;
   …
```

10.2 并行语句

10.2.4 元件例化语句

```
COMPONENT 元件名 IS
    GENERIC (类属表);                          -- 元件定义语句
    PORT  (端口名表);
END COMPONENT 文件名;
例化名 : 元件名 PORT MAP(                      -- 元件例化语句
      [端口名 =>] 连接端口名，...);
```

10.2 并行语句

10.2.5 生成语句

```
[标号：] FOR 循环变量 IN 取值范围 GENERATE
            说明
          BEGIN
          并行语句
              END GENERATE [标号] ；
[标号：] IF 条件GENERATE
            说明
          Begin
          并行语句
              END GENERATE [标号] ；
```

10.2 并行语句

10.2.5 生成语句

表达式 **TO** 表达式 ; **--** 递增方式，如**1 TO 5**

表达式 **DOWNTO** 表达式 ; **--** 递减方式，如**5 DOWNTO 1**

【例10-25】

```
  ...
COMPONENT comp
PORT (x : IN STD_LOGIC ;
       y : OUT STD_LOGIC );
END COMPONENT ;
SIGNAL a :STD_LOGIC_VECTOR(0 TO 7);
SIGNAL b :STD_LOGIC_VECTOR(0 TO 7);
...
gen : FOR i IN  a'RANGE  GENERATE
 u1: comp PORT MA (x=>a(i), y=>b(i));
END GENERATE gen,
 …
```

10.2 并行语句

10.2.5 生成语句

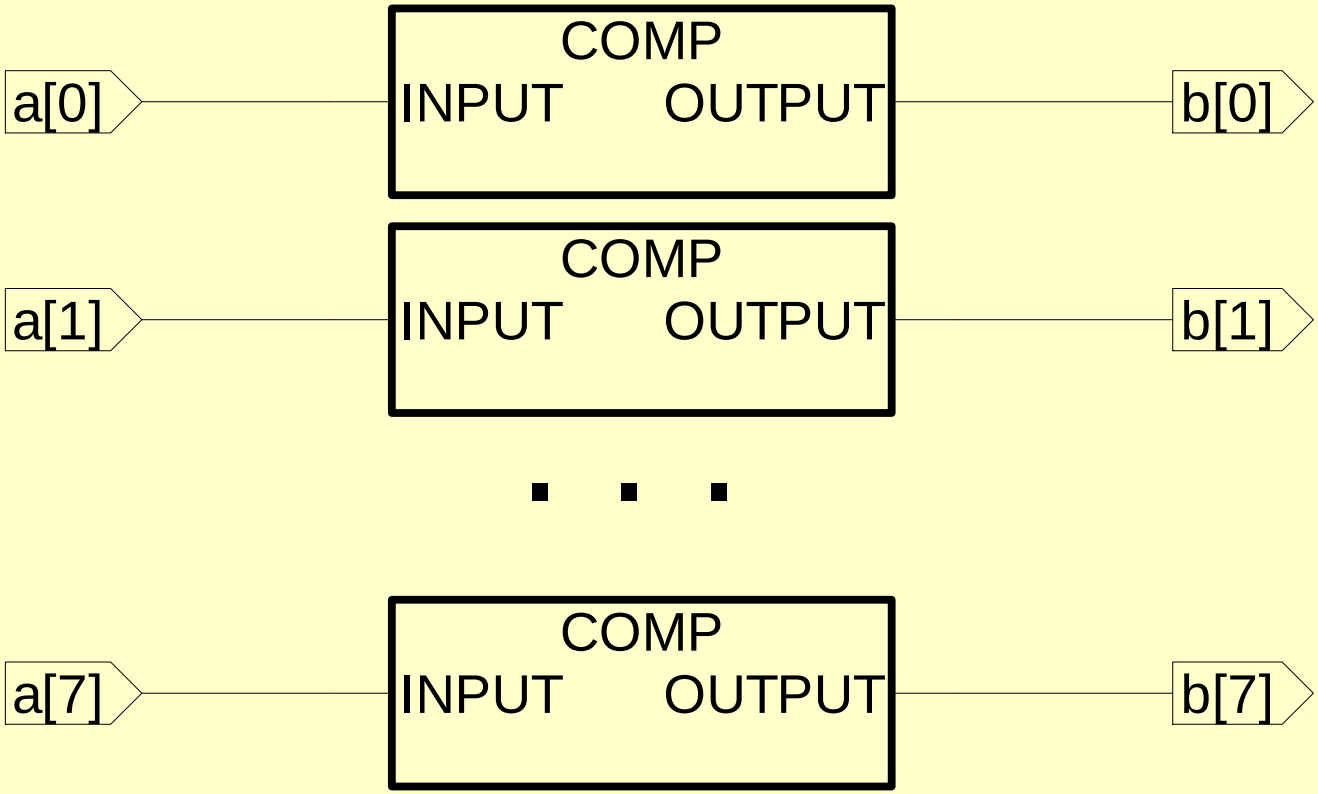


图10-3 生成语句产生的8个相同的电路模块

10.2 并行语句

10.2.5 生成语句

【例10-26】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Latch IS
  PORT( D, ENA : IN STD_LOGIC;
        Q : OUT STD_LOGIC   );
END ENTITY Latch ;
  ARCHITECTURE one OF Latch IS
  SIGNAL sig_save : STD_LOGIC;
  BEGIN
   PROCESS (D, ENA)
     BEGIN
      IF  ENA = '1'  THEN  sig_save <= D ;  END IF ;
        Q <= sig_save ;
   END PROCESS ;
END ARCHITECTURE one;
```

【例10-27】

```
LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   ENTITY SN74373 IS
   PORT (D : IN STD_LOGIC_VECTOR( 8 DOWNTO 1 );
     OEN ,G : IN  STD_LOGIC;
           Q : OUT STD_LOGIC_VECTOR(8 DOWNTO 1));
   END ENTITY SN74373;
ARCHITECTURE two OF SN74373 IS
      SIGNAL sigvec_save : STD_LOGIC_VECTOR(8 DOWNTO 1);
     BEGIN
     PROCESS(D, OEN, G , sigvec_save)
      BEGIN
       IF OEN = '0' THEN Q <= sigvec_save; ELSE Q <= "ZZZZZZZZ"; END IF;
       IF G = '1' THEN  Sigvec_save <= D;  END IF;
    END PROCESS;
  END ARCHITECTURE two;
 ARCHITECTURE one OF SN74373 IS
  COMPONENT Latch
  PORT (   D, ENA : IN STD_LOGIC;
                 Q : OUT STD_LOGIC );
  END COMPONENT;
  SIGNAL sig_mid : STD_LOGIC_VECTOR( 8 DOWNTO 1 );
  BEGIN
   GeLatch : FOR iNum IN 1 TO 8 GENERATE
    Latchx : Latch PORT MAP(D(iNum),G,sig_mid(iNum));
  END GENERATE;
      Q <= sig_mid  WHEN OEN = '0' ELSE
           "ZZZZZZZZ";   --当OEN=1时，Q(8)~Q(1)输出状态呈高阻态
END ARCHITECTURE one;
```

【例10-28】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY d_ff IS
PORT (  d, clk_s : IN STD_LOGIC ;
                q : OUT STD_LOGIC ;
               nq : OUT STD_LOGIC  );
END ENTITY d_ff;
ARCHITECTURE a_rs_ff OF d_ff IS
BEGIN
bin_p_rs_ff : PROCESS(CLK_S)
    BEGIN
      IF clk_s = '1' AND clk_s'EVENT  THEN q <= d; nq <= NOT d;
      END IF;
    END PROCESS;
END ARCHITECTURE a_rs_ff;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY cnt_bin_n is
GENERIC (n : INTEGER := 6);
PORT (q : OUT STD_LOGIC_VECTOR (0 TO n-1);
    in_1 : IN  STD_LOGIC );
END ENTITY cnt_bin_n;
```

（接下页）

```
ARCHITECTURE behv OF cnt_bin_n IS
COMPONENT d_ff
  PORT(d, clk_s :  IN STD_LOGIC;
            Q, NQ : OUT STD_LOGIC);
END COMPONENT d_ff;
SIGNAL s : STD_LOGIC_VECTOR(0 TO n);
BEGIN
  s(0) <= in_1;
   q_1 : FOR i IN 0 TO n-1 GENERATE
   dff : d_ff PORT MAP (s(i+1), s(I), q(i), s(i+1));
  END GENERATE;
END ARCHITECTURE behv;
```

10.2 并行语句

10.2.5 生成语句

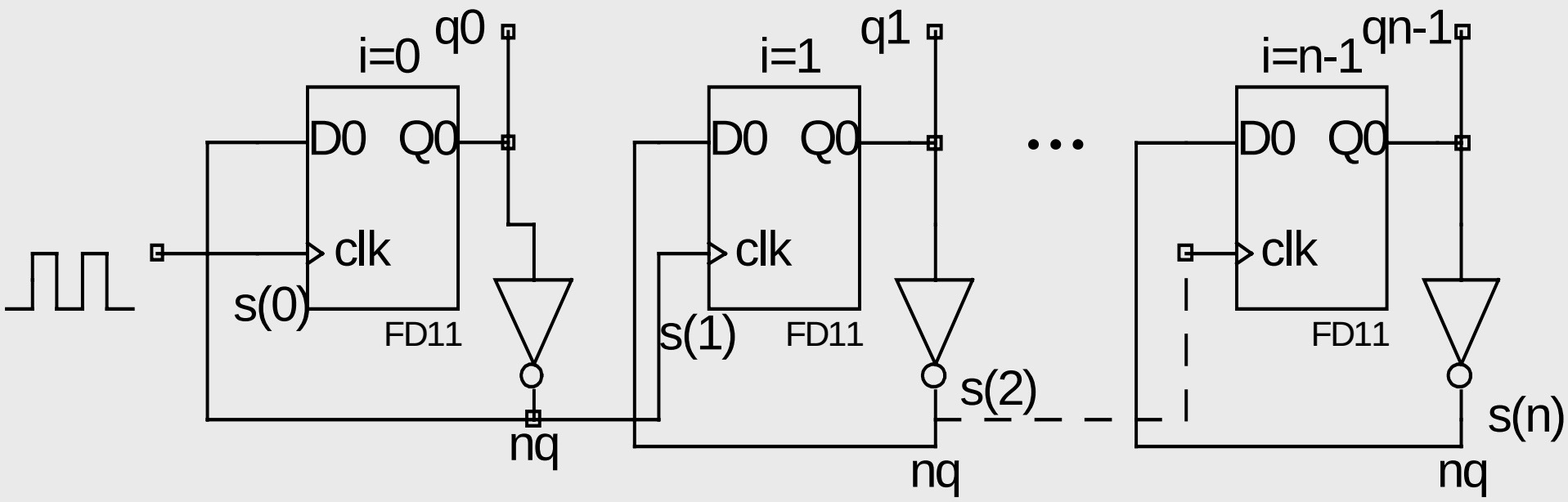


图10-4 6 位二进制计数器原理图

10.2 并行语句

10.2.6 REPORT语句

【例10-29】

```
LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  ENTITY RSFF2 IS
    PORT ( S, R : IN std_logic;
           Q, QF : OUT std_logic );
  END RSFF2;
  ARCHITECTURE BHV OF RSFF2 IS
  BEGIN
   P1: PROCESS(S,R)
     VARIABLE D : std_logic;
  BEGIN
    IF R = '1' and S = '1' THEN
      REPORT " BOTH R AND S IS '1'"; --报告出错信息
    ELSIF R = '1' and S = '0' THEN  D := '0';
    ELSIF R = '0' and S = '1' THEN  D := '1' ;
    END IF;
    Q  <= D;  QF <= NOT D;
    END PROCESS;
  END BHV;
```

10.2 并行语句

10.2.7 断言语句

ASSERT<条件表达式>

REPORT<出错信息>

SEVERITY<错误级别>；

表10-1 预定义错误等级

Note（通报）	报告出错信息，可以通过编译
Warning（警告）	报告出错信息，可以通过编译
Error（错误）	报告出错信息，暂停编译
Failure（失败）	报告出错信息，暂停编译

10.2 并行语句

10.2.7 断言语句

1. 顺序断言语句

【例10-30】

```
   P1: PROCESS(S,R)
     VARIABLE D : std_logic;
  BEGIN
ASSERT not (R='1'and S='1')
        REPORT "both R and S equal to ' 1 '"
        SEVERITY Error;
    IF  R = '1' and S = '0' THEN  D := '0';
    ELSIF R = '0' and S = '1' THEN  D := '1' ;
    END IF;
    Q  <= D;  QF <= NOT D;
    END PROCESS;
```

2. 并行断言语句

【例10-31】

```
LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
  ENTITY RSFF2 IS
  PORT(S, R : IN std_logic;
       Q,QF : OUT std_logic);
  END  RSFF2;
   ARCHITECTURE BHV OF RSFF2 IS
     BEGIN
  PROCESS(R,S)
      BEGIN
        ASSERT not (R='1'and S='1')
        REPORT "both R and S equal to ' 1 '"
        SEVERITY Error;
 END PROCESS;
 PROCESS(R,S)
   VARIABLE D : std_logic := '0';
    BEGIN
    IF R='1' and S='0' THEN D :='0';
 ELSIF R='0' and S='1' THEN D :='1'; END IF;
   Q <= D ; QF <= NOT D ;
 END PROCESS;
 END ;
```

10.3 属性描述与定义语句

1. 信号类属性

```
(NOT clock'STABLE AND clock ='1')
(clock'EVENT AND clock ='1')
```

2. 数据区间类属性

```
...
SIGNAL range1 : IN STD_LOGIC_VECTOR (0 TO 7) ;
...
FOR i IN range1'RANGE LOOP
  ...
```

10.3 属性描述与定义语句

3. 数值类属性

```
...
PROCESS (clock, a, b);
TYPE obj IS ARRAY (0 TO 15) OF BIT ;
SIGNAL ele1, ele2, ele3, ele4    : INTEGER ;
BEGIN
 ele1 <= obj'RIGNT ;
 ele2 <= obj'LEFT ;
 ele3 <= obj'HIGH ;
 ele4 <= obj'LOW  ;
...
```

【例10-32】

```
LIBRARY IEEE;--PARITY GENERATOR
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY parity IS
    GENERIC (bus_size : INTEGER := 8 );
    PORT (input_bus : IN STD_LOGIC_VECTOR(bus_size-1 DOWNTO 0);
            even_numbits, odd_numbits : OUT STD_LOGIC ) ;
END parity ;
ARCHITECTURE behave OF parity IS
BEGIN
PROCESS (input_bus)
    VARIABLE temp: STD_LOGIC;
BEGIN
    temp := '0';
    FOR i IN input_bus'LOW TO input_bus'HIGH LOOP
temp := temp XOR input_bus( i ) ;
    END LOOP ;
    odd_numbits <= temp ;
    even_numbits <= NOT temp;
END PROCESS;
END behave;
```

KX 康芯科技

10.3 属性描述与定义语句

4. 数组属性'LENGTH

```
...
TYPE arry1 ARRAY (0 TO 7) OF BIT ;
VARIABLE wth: INTEGER;
...
wth1: =arry1’LENGTH; -- wth1 = 8
...
```

10.3 属性描述与定义语句

5. 用户定义属性

```
ATTRIBUTE 属性名 : 数据类型;
ATTRIBUTE 属性名 OF 对象名 : 对象类型IS 值;
```

```
LIBRARY synplify;
USE synplicity.attributes.all;
```

5. 用户定义属性

```
LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
 ENTITY cntbuf IS
    PORT(  Dir: IN STD_LOGIC;
           Clk,Clr,OE: IN STD_LOGIC;
           A,B: INOUT STD_LOGIC_VECTOR (0 to 1);
           Q: INOUT STD_LOGIC_VECTOR (3 downto 0)  );
    ATTRIBUTE PINNUM : STRING;
    ATTRIBUTE PINNUM OF Clk: signal is "1";
    ATTRIBUTE PINNUM OF Clr: signal is "2";
    ATTRIBUTE PINNUM OF Dir: signal is "3";
    ATTRIBUTE PINNUM OF OE: signal is "11";
    ATTRIBUTE PINNUM OF Q: signal is "17,16,15,14";
END cntbuf;
```

习　题

10-1. 进程有哪几种主要类型？不完全组合进程是由什么原因引起的？有什么特点？如何避免？

10-2. 给触发器复位的方法有哪两种？如果时钟进程中用了敏感信号表，哪种复位方法要求把复位信号放在敏感信号表中？

10-3. 用**WITH_SELECT_WHEN**语句描述**4**个**16**位至**1**个**16**位输出的**4**选**1**多路选择器。

10-4. 为什么说一条并行赋值语句可以等效为一个进程？如果是这样的话，该语句怎样实现敏感信号的检测？

习　题

10-5. 下述**VHDL**代码的综合结果会有几个触发器或锁存器？

程序**1**：

```
architecture rtl of ex is
    signal a, b: std_logic_vector(3 downto 0);
begin
     process(clk)
     begin
         if clk = '1' and clk'event then
               if q(3) /= '1' then  q <= a + b;
                end if;
            end if;
     end process;
end rtl;
```

习　题

10-5. 下述VHDL代码的综合结果会有几个触发器或锁存器？

程序2：

```
architecture rtl of ex is
    signal a, b: std_logic_vector(3 downto 0);
begin
    process(clk)
        variable int: std_logic_vector(3 downto 0);
    begin
        if clk ='1' and clk'event then
            if int(3) /= '1' then  int := a + b ; q <= int;
             end if;
        end if;
    end process;
end rtl;
```

习　题

10-5. 下述VHDL代码的综合结果会有几个触发器或锁存器？

程序3：

```
architecture rtl of ex is
    signal a, b，  c, d, e: std_logic_vector(3 downto 0);
begin
    process(c, d, e, en)
    begin
       if en ='1'  then  a <= c ;  b <= d;
         else    a <= e;
       end if;
    end process;
end rtl;
```

习　题

10-6. 比较CASE语句与WITH_SELECT语句，叙述它们的异同点。

10-7. 将以下程序段转换为WHEN_ELSE语句：

```
PROCESS (a，b，c，d)
  BEGIN
  IF a= '0' AND b='1' THEN  next1 <= "1101" ;
       ELSIF  a='0' THEN  next1 <= d ;
   ELSIF  b='1' THEN  next1 <= c ;
         ELSE
         Next1 <= "1011" ;
  END  IF;
END PROCESS;
```

习　题

10-8. 说明以下两程序有何不同，哪一电路更合理？试画出它们的电路。

程序1：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY EXAP IS  PORT ( clk,a,b  : IN STD_LOGIC;
                                y : OUT STD_LOGIC );
END EXAP ;
 ARCHITECTURE behav OF EXAP IS
 SIGNAL x : STD_LOGIC;
BEGIN
 PROCESS
 BEGIN
  WAIT UNTIL CLK ='1' ;
       x <= '0';    y <= '0';
  IF a = b THEN    x <= '1';
  END IF;
  IF x='1' THEN    y <= '1' ;
  END IF ;
 END PROCESS  ;
END behav;
```

习　题

10-8. 说明以下两程序有何不同，哪一电路更合理？试画出它们的电路。

程序2：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY EXAP IS  PORT ( clk,a,b  : IN STD_LOGIC;
                                y : OUT STD_LOGIC );
END EXAP ;
 ARCHITECTURE behav OF EXAP IS
 BEGIN
 PROCESS
  VARIABLE x : STD_LOGIC;
 BEGIN
  WAIT UNTIL CLK ='1' ;
      x := '0';    y <= '0';
   IF a = b THEN  x := '1';
  END IF;
  IF x='1' THEN   y <= '1' ;
  END IF ;
 END PROCESS  ;
END behav;
```

实 验 与 设 计

10-1 移位相加硬件乘法器设计

(1) 实验目的：学习应用移位相加原理设计**8**位乘法器。

(2) 实验原理：该乘法器是由**8**位加法器构成的以时序方式设计的**8**位乘法器。原理是：乘法通过逐项移位相加来实现相乘。从被乘数的最低位开始，若为1，则乘数左移后与上一次的和相加；若为0，左移后以全零相加，直至被乘数的最高位。从图10-5的逻辑图及其乘法操作时序图图10-6(示例中的相乘数为9FH和FDH)上可以清楚地看出此乘法器的工作原理。

【例10-33】

```
LIBRARY IEEE;          -- 8位右移寄存器
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SREG8B IS
    PORT (  CLK, LOAD : IN STD_LOGIC;
             DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              QB : OUT STD_LOGIC  );
END SREG8B;
ARCHITECTURE behav OF SREG8B IS
    SIGNAL REG8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (CLK, LOAD)
    BEGIN
      IF CLK'EVENT AND CLK = '1' THEN
          IF LOAD = '1' THEN  REG8 <= DIN;
             ELSE  REG8(6 DOWNTO 0) <= REG8(7 DOWNTO 1);
            END IF;
        END IF;
    END PROCESS;
    QB <= REG8(0);                              -- 输出最低位
END behav;
```

【例10-34】

```
LIBRARY IEEE;      --8位加法器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ADDER8B IS
    PORT ( CIN : IN STD_LOGIC;
           A , B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
           COUT : OUT STD_LOGIC );
END ADDER8B;
ARCHITECTURE behav OF ADDER8B IS
    SIGNAL SINT, AA,BB : STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
AA<='0'&A;BB<='0'&B; SINT<=AA+BB+CIN;S<=SINT(7 DOWNTO 0);
COUT<=SINT(8);
END behav;
```

【例10-35】

```
LIBRARY IEEE; --1位乘法器
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ANDARITH IS                                  -- 选通与门模块
    PORT ( ABIN : IN STD_LOGIC;
           DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          DOUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END ANDARITH;
ARCHITECTURE behav OF ANDARITH IS
BEGIN
    PROCESS(ABIN, DIN)
    BEGIN
        FOR I IN 0 TO 7 LOOP          -- 循环，完成8位与1位运算
            DOUT(I) <= DIN(I) AND ABIN;
        END LOOP;
    END PROCESS;
END behav;
```

【例10-36】

```
LIBRARY IEEE;                                    --16位锁存器/右移寄存器
USE IEEE.STD_LOGIC_1164.ALL;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY REG16B IS                                  -- 16位锁存器
   PORT (CLK, CLR : IN STD_LOGIC;
      D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
      Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END REG16B;
ARCHITECTURE behav OF REG16B IS
   SIGNAL R16S : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
   PROCESS(CLK, CLR)
   BEGIN
IF CLR='1' THEN R16S<="0000000000000000";--时钟到来时，锁存输入值，并右移低8位
ELSIF CLK'EVENT AND CLK='1' THEN
 R16S(6 DOWNTO 0) <=R16S(7 DOWNTO 1);       --右移低8位
         R16S(15 DOWNTO 7) <= D;                    -- 将输入锁到高8位
      END IF;
   END PROCESS;
   Q <= R16S;
END behav;
```

【例10-37】

```
END behav;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ARICTL IS
    PORT (CLK, START : IN STD_LOGIC;
        CLKOUT,RSTALL : OUT STD_LOGIC );
END ARICTL;
ARCHITECTURE behav OF ARICTL IS
    SIGNAL CNT4B : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    PROCESS(CLK, START)
    BEGIN
        RSTALL <= START;
        IF START = '1' THEN  CNT4B <= "0000";
        ELSIF CLK'EVENT AND CLK ='1' THEN
         IF CNT4B < 8 THEN CNT4B <= CNT4B + 1; END IF;
        END IF;
    END PROCESS;
    PROCESS(CLK, CNT4B, START)
    BEGIN
        IF START = '0' THEN
            IF CNT4B < 8 THEN  CLKOUT <= CLK;
            ELSE CLKOUT <= '0';  END IF;
        ELSE  CLKOUT <= CLK; END IF;
    END PROCESS;
END behav;
```

【例10-38】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
ENTITY MULTI8X8 IS                                  -- 8位乘法器顶层设计
    PORT ( CLKK,START  : IN STD_LOGIC;
                 A,  B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                  DOUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END MULTI8X8;
ARCHITECTURE struc OF MULTI8X8 IS
COMPONENT ARICTL
    PORT (  CLK, START : IN STD_LOGIC;
         CLKOUT, RSTALL : OUT STD_LOGIC  );
END COMPONENT;
COMPONENT ANDARITH
    PORT (  ABIN : IN STD_LOGIC;
              DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             DOUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  );
END COMPONENT;
COMPONENT ADDER8B
    PORT (CIN : IN STD_LOGIC;
             A, B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
          COUT : OUT STD_LOGIC );
END COMPONENT;
COMPONENT SREG8B
    PORT (  CLK, LOAD : IN STD_LOGIC;
```

（接下页）

```
  DIN : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                QB : OUT STD_LOGIC  );
  END COMPONENT;
  COMPONENT REG16B
      PORT (   CLK, CLR : IN STD_LOGIC;
          D : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
          Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)  );
  END COMPONENT;
      SIGNAL GNDINT, INTCLK, RSTALL, NEWSTART, QB : STD_LOGIC;
      SIGNAL ANDSD : STD_LOGIC_VECTOR(7 DOWNTO 0);
       SIGNAL DTBIN : STD_LOGIC_VECTOR(8 DOWNTO 0);
      SIGNAL DTBOUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
  BEGIN
      DOUT <= DTBOUT;   GNDINT <= '0';
  PROCESS(CLKK,START)
  BEGIN
       IF START='1' THEN NEWSTART<='1';
       ELSIF CLKK='0' THEN   NEWSTART<='0'; END IF;
  END PROCESS;
  U1   :  ARICTL    PORT  MAP(CLK=>CLKK,START=>NEWSTART,  CLKOUT=>INTCLK,
RSTALL=>RSTALL);
  U2 : SREG8B  PORT MAP(CLK=>INTCLK, LOAD=>RSTALL, DIN=>B, QB=>QB  );
  U3 : ANDARITH PORT MAP(ABIN => QB, DIN => A,DOUT => ANDSD);
  U4 : ADDER8B  PORT MAP(CIN => GNDINT, A=>DTBOUT(15 DOWNTO 8), B=>ANDSD,
                  S => DTBIN(7 DOWNTO 0), COUT => DTBIN(8) );
  U5 : REG16B   PORT MAP(CLK=>INTCLK, CLR=>RSTALL, D=>DTBIN, Q=>DTBOUT );
   END struc;
```

实 验 与 设 计

10-1 移位相加硬件乘法器设计

(3) 实验内容1：根据给出的乘法器逻辑原理图及其各模块的**VHDL**描述，在**QuartusII**上完成全部设计，包括编辑、编译、综合和仿真操作等。以**87H**乘以**F5H**为例，进行仿真，对仿真波形作出详细解释，包括对**8**个工作时钟节拍中，每一节拍乘法操作的方式和结果，对照波形图给以详细说明，根据顶层设计例**10-38**，结合图**10-5**，画出乘法器的详细电路原理框图。

(4) 实验内容2：编程下载，进行实验验证。实验电路选择**No.1**，**8**位乘数用键**2**、键**1**输入；**8**位被乘数用键**4**和键**3**输入；**16**位乘积可由**4**个数码管（数码管**8**、**7**、**6**、**5**）显示；用键**8**输入**CLK**，键**7**输入**START**（注意，**START**由高到低是清**0**，由低到高电平是允许乘法计算）。详细观察每一时钟节拍的运算结果，并与仿真结果进行比较。

实 验 与 设 计

10-1 移位相加硬件乘法器设计

(5) 实验内容3：乘法时钟连接实验系统上的连续脉冲，如**clock0**，设计一个此乘法器的控制模块，接受实验系统上的连续脉冲，如**clock0**，当给定启动/清**0**信号后，能自动发出**CLK**信号驱动乘法运算，当**8**个脉冲后自动停止（参考程序：例**10-37**）。

(6) 实验内容4：设计一个纯组合电路的**8X8**等于**16**位的乘法器和一个**LPM**乘法器（选择不同的流水线方式），具体说明并比较这几种乘法器的逻辑资源占用情况和运行速度情况。

（7）实验报告：根据例10-33至例10-38，详细分析各模块的逻辑功能，及其他们工作原理，详细记录并分析实验2和实验3的过程和结果，完成实验报告。

实 验 与 设 计

10-1 移位相加硬件乘法器设计

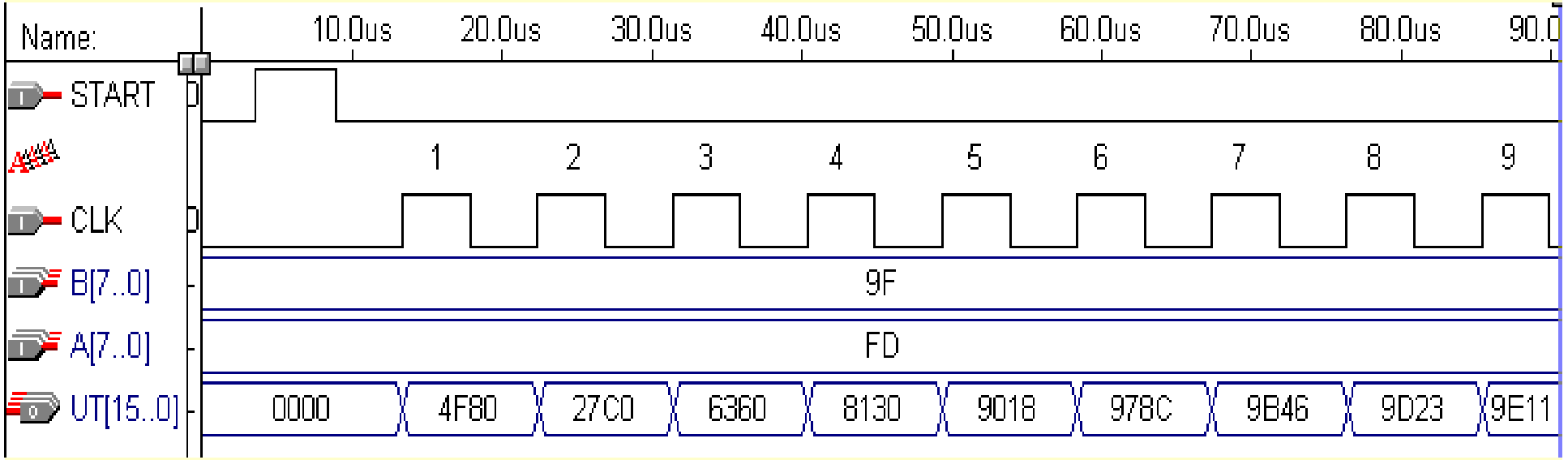


图10-5 8位移位相加乘法器运算逻辑波形图

实 验 与 设 计

10-2 等精度频率计/相位计设计

(1) 实验原理：基于传统测频原理的频率计的测量精度将随被测信号频率的下降而降低，即测量精度随被测信号的频率的变化而变化，在实用中有较大的局限性，而等精度频率计不但具有较高的测量精度，且在整个频率区域能保持恒定的测试精度。设计项目可达到的指标为：

频率测试功能：测频范围0.1Hz～100MHz。测频精度：测频全域相对误差恒为百万分之一。

脉宽测试功能：测试范围0.1μs～1s，测试精度0.01μs。

占空比测试功能：测试（显示）精度1%～99%。

相位测试功能：测试范围0～，测试精度0.。

实 验 与 设 计

1、主系统组成

等精度频率计的主系统如图**10-6**所示，主要由**6**个部分构成：

(1) 信号整形电路。用于对待测信号进行放大和整形，以作**PLD**器件的输入信号。

(2) 测频电路。是测频的核心电路模块，可以由**FPGA**器件担任。

(3) 100MHz的标准频率信号源(可通过**PLL**倍频所得)接入**FPGA**。

(4) 单片机电路模块。用于控制**FPGA**的测频操作和读取测频数据，并作出相应数据处理。安排单片机的**P0**口读取测试数据，**P2**口向**FPGA**发控制命令。

(5) 键盘模块。可以用**5**个键执行测试控制，一个是复位键，其余是命令键。

(6) 数码显示模块。可以用**7**个数码管显示测试结果，最高可表示百万分之一的精度。

考虑到提高单片机**IO**口的利用率，降低编程复杂性，提高单片机的计算速度以及降低数码显示器对主系统的干扰，可以采用串行静态显示方式或液晶显示。

1、主系统组成

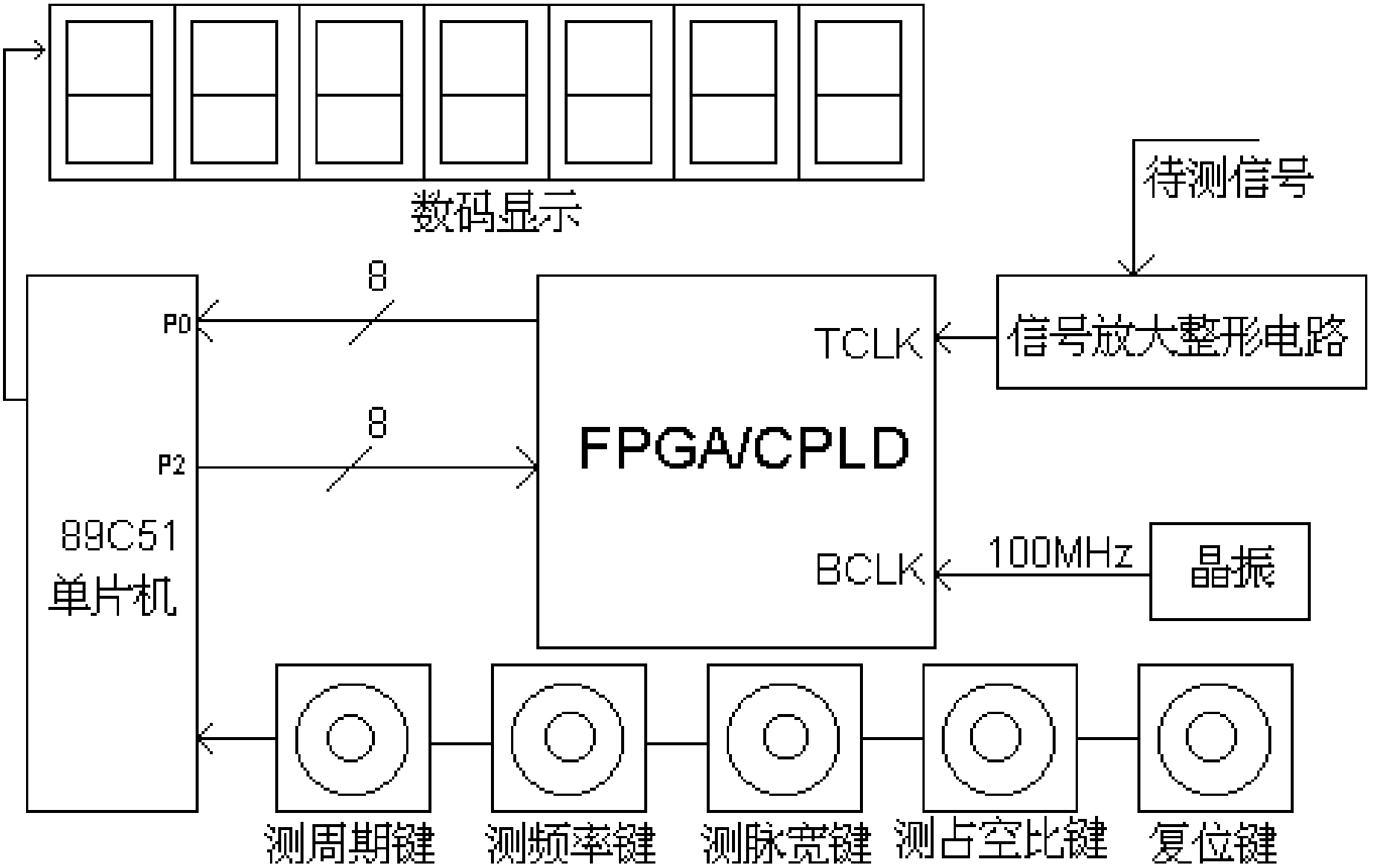


图10-6 频率计主系统电路组成

康芯科技

2、主系统组成测频原理

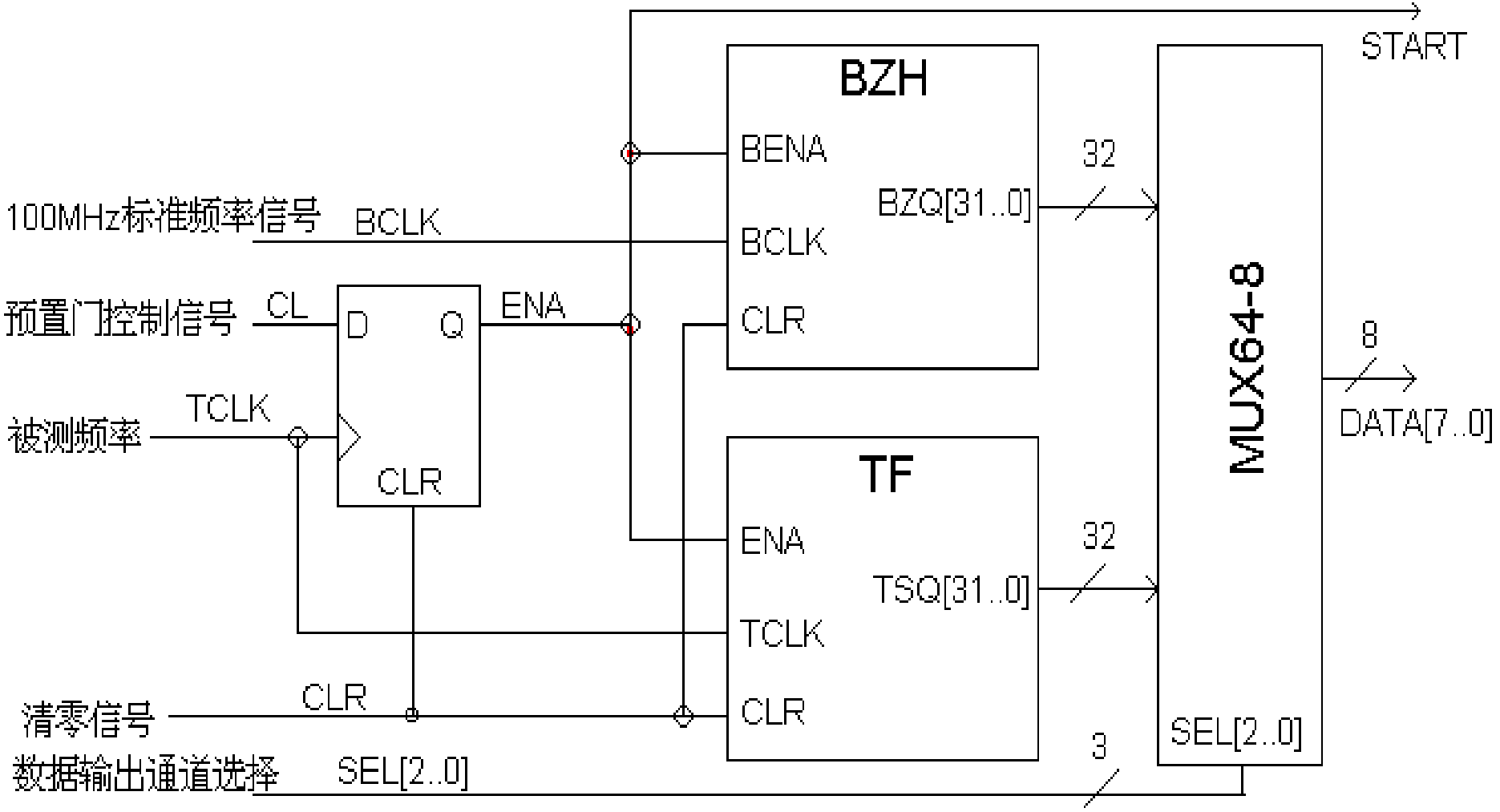


图10-7 等精度频率计主控结构

康芯科技

实 验 与 设 计

2、主系统组成测频原理

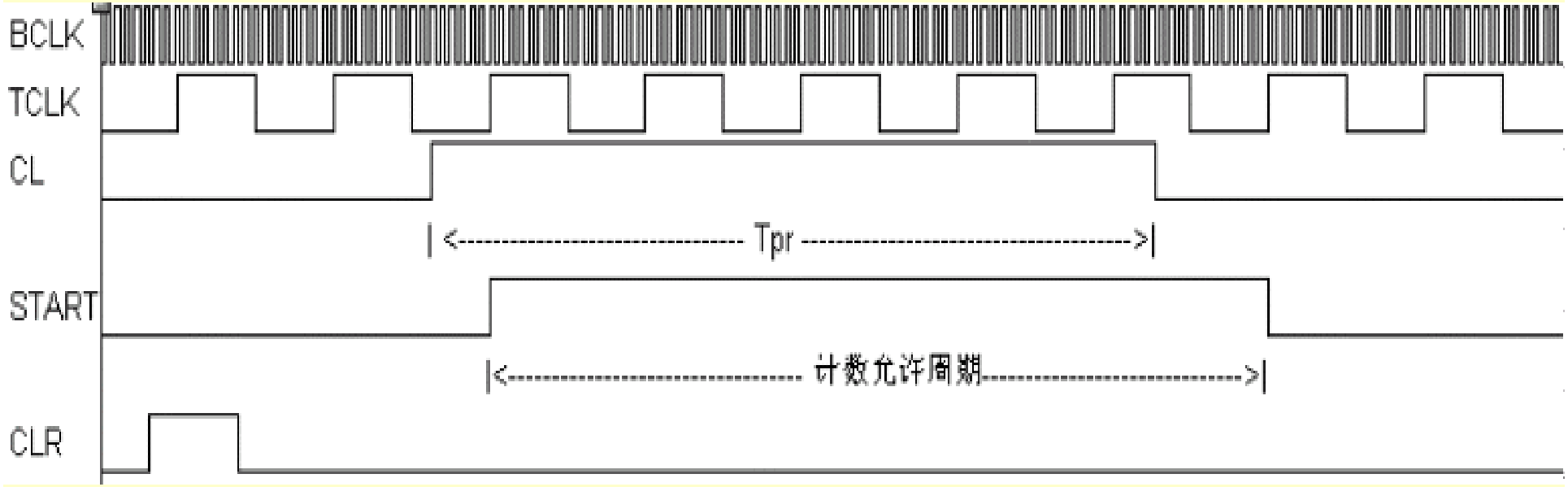


图10-8 频率计测控时序

【例10-39】

```
LIBRARY IEEE;  --等精度频率计FPGA设计部分
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY etester IS
  PORT (BCLK : IN STD_LOGIC; --标准频率时钟信号clock2，50MHZ
        TCLK : IN STD_LOGIC; --待测频率时钟信号
         CLR : IN STD_LOGIC; --清零和初始化信号
CL : IN STD_LOGIC; --当SPUL为高电平时，CL为预置门控信号，用于测频计数
--时间控制当SPUL为低电平时，CL为测脉宽控制信号，
--CL高电平时测高电平脉宽而当CL为低电平时，测低电平脉宽。
     SPUL : IN STD_LOGIC; --测频或测脉宽控制
    START : OUT STD_LOGIC;--起始计数标志信号
         EEND : OUT STD_LOGIC; --由低电平变到高电平时指示脉宽计数结束，
          SEL : IN STD_LOGIC_VECTOR(2 DOWNTO 0); --数据读出选同控制
     DATA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --8位数据读出
END etester;
ARCHITECTURE behav OF etester IS
SIGNAL BZQ  : STD_LOGIC_VECTOR(31 DOWNTO 0);  --标准计数器
SIGNAL TSQ  : STD_LOGIC_VECTOR(31 DOWNTO 0);  --测频计数器
SIGNAL ENA  : STD_LOGIC;                      --计数使能
SIGNAL  MA, CLK1, CLK2, CLK3 : STD_LOGIC;
SIGNAL   Q1, Q2, Q3，  BENA, PUL : STD_LOGIC;
SIGNAL   SS : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
START <= ENA ;
DATA <= BZQ(7  DOWNTO  0) WHEN SEL="000" ELSE -- 标准频率计数低8位输出
```

（接下页）

```
BZQ(15 DOWNTO  8) WHEN SEL="001" ELSE
        BZQ(23 DOWNTO 16) WHEN SEL="010" ELSE
       BZQ(31 DOWNTO 24) WHEN SEL="011" ELSE -- 标准频率计数最高8位输出
       TSQ(7 DOWNTO   0) WHEN SEL="100" ELSE --待测频率计数值最低8位输出
       TSQ(15 DOWNTO  8) WHEN SEL="101" ELSE
       TSQ(23 DOWNTO 16) WHEN SEL="110" ELSE
       TSQ(31 DOWNTO 24) WHEN SEL="111" ELSE --待测频率计数值最高8位输出
       TSQ(31 DOWNTO 24) ;
BZH :  PROCESS(BCLK, CLR)              --标准频率测试计数器，标准计数器
    BEGIN
      IF CLR = '1' THEN   BZQ <= ( OTHERS=>'0' ) ;
      ELSIF BCLK'EVENT AND BCLK = '1' THEN
       IF BENA = '1' THEN   BZQ <= BZQ + 1;   END IF;
      END IF;
   END PROCESS;
TF : PROCESS(TCLK, CLR, ENA)               --待测频率计数器，测频计数器
  BEGIN
   IF CLR = '1' THEN   TSQ <= ( OTHERS=>'0' );
   ELSIF TCLK'EVENT AND TCLK = '1' THEN
    IF ENA = '1' THEN   TSQ <= TSQ + 1;  END IF;
   END IF;
END PROCESS;
PROCESS(TCLK,CLR)
 BEGIN
  IF CLR = '1' THEN   ENA <= '0' ;
  ELSIF TCLK'EVENT AND TCLK='1' THEN ENA <= CL ;  END IF;
END PROCESS;
```

（接下页）

```
MA<=(TCLK AND CL) OR NOT(TCLK OR CL) ; --测脉宽逻辑
 CLK1<=NOT MA ; CLK2<=MA AND Q1 ; CLK3<=NOT CLK2; SS<=Q2 & Q3 ;
DD1:  PROCESS(CLK1,CLR)
    BEGIN
     IF CLR = '1' THEN   Q1 <= '0' ;
      ELSIF CLK1'EVENT AND CLK1 = '1' THEN  Q1 <= '1' ; END IF;
     END PROCESS;
DD2:  PROCESS(CLK2,CLR)
  BEGIN
    IF CLR = '1' THEN   Q2 <= '0' ;
    ELSIF CLK2'EVENT AND CLK2 = '1' THEN  Q2 <= '1' ;  END IF;
  END PROCESS;
DD3:  PROCESS(CLK3,CLR)
   BEGIN
    IF CLR = '1' THEN   Q3 <= '0' ;
    ELSIF CLK3'EVENT AND CLK3 = '1' THEN  Q3 <= '1' ;  END IF;
 END PROCESS;
PUL<='1' WHEN SS="10" ELSE --当SS=“10”时，PUL高电平，允许标准计数器计数，
    '0' ;                  --禁止计数
EEND<='1' WHEN SS="11" ELSE --EEND为低电平时，表示正在计数，由低电平变到高电平
     '0' ;                  --时，表示计数结束，可以从标准计数器中读数据了
BENA<=ENA WHEN SPUL='1' ELSE--标准计数器时钟使能控制信号，当SPUL为1时，测频率
     PUL WHEN SPUL='0' ELSE--当SPUL为0时，测脉宽和占空比
     PUL ;
END behav;
```

3、VHDL测试程序设计

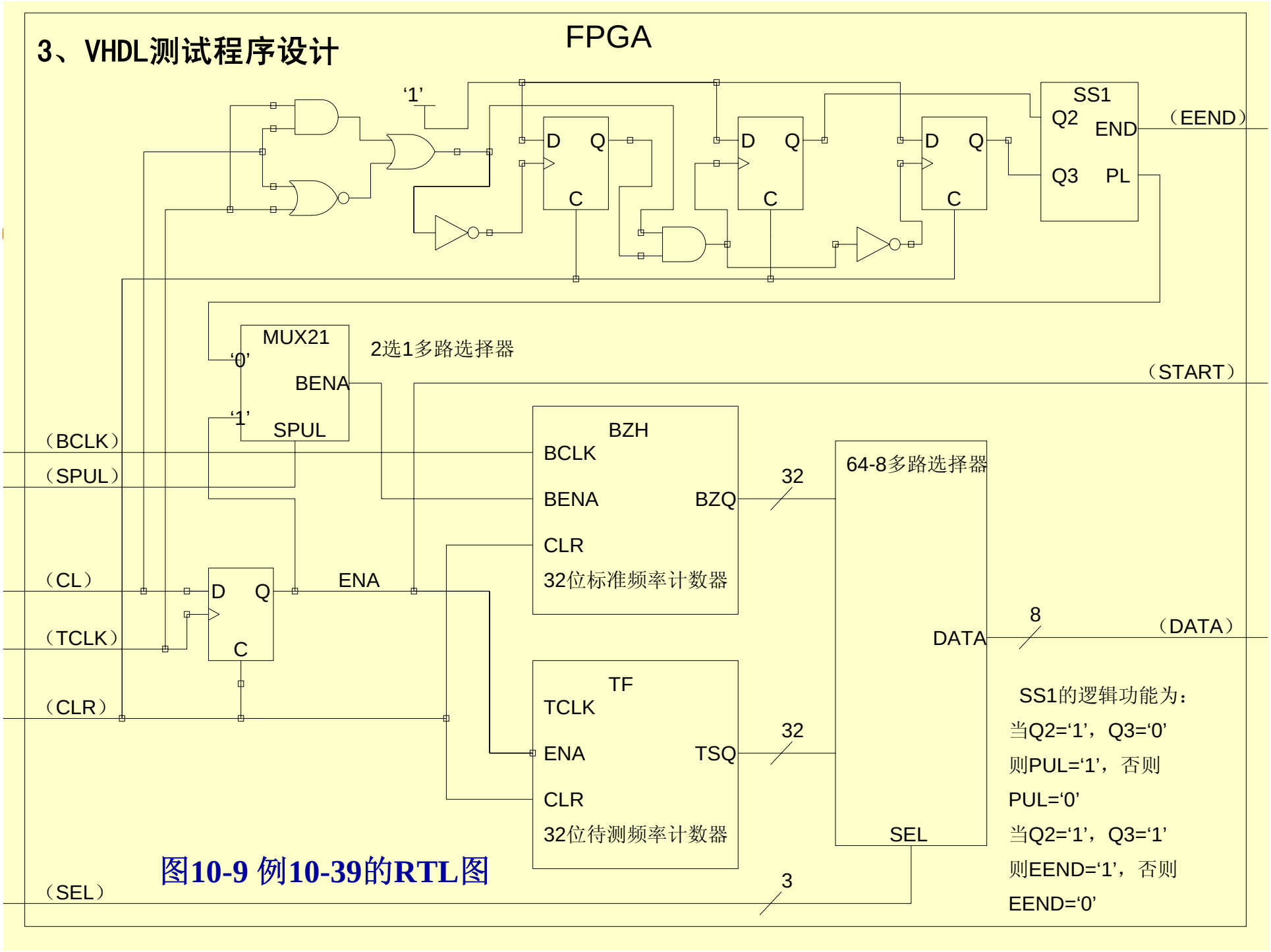


图10-9 例10-39的RTL图

实 验 与 设 计

3、VHDL测试程序设计

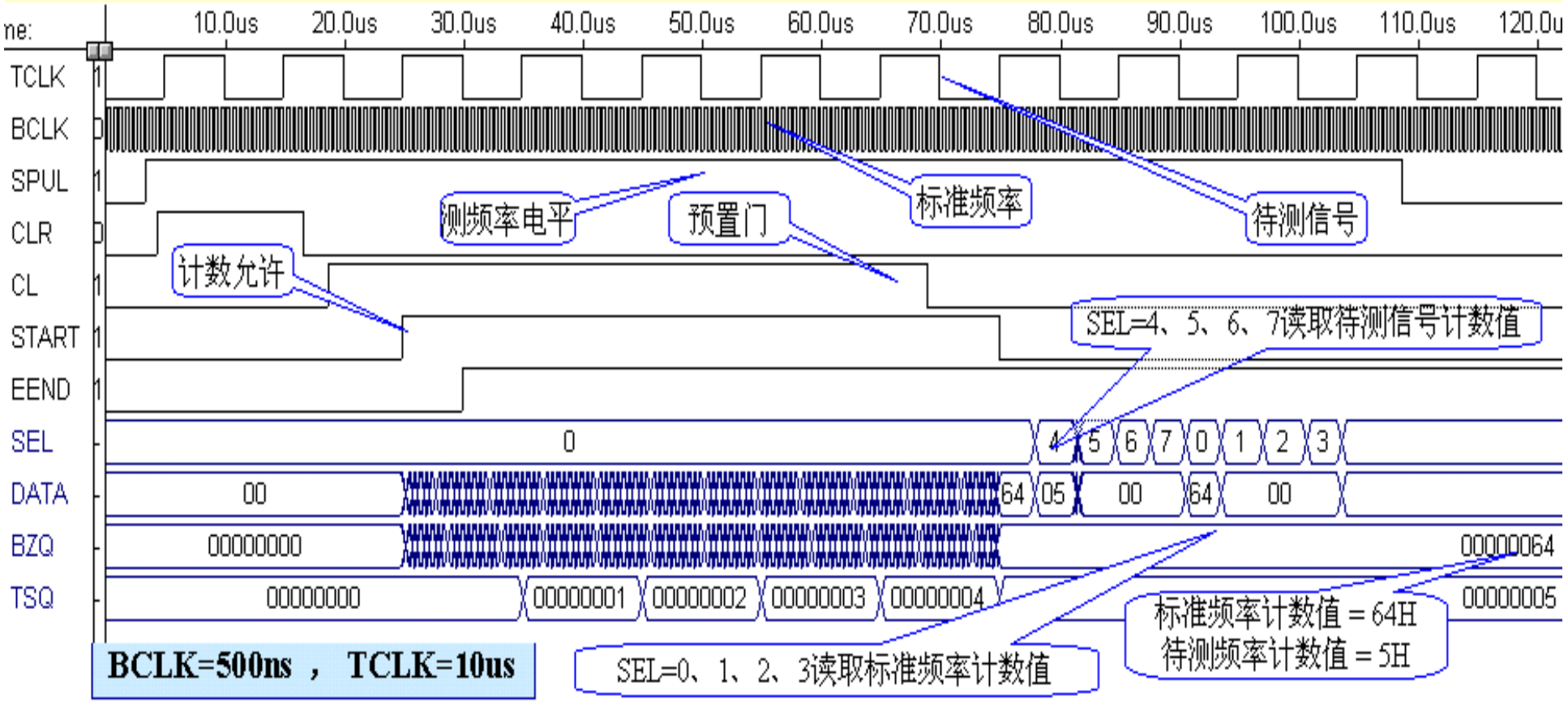


图10-10 等精度频率计测频时序图

实 验 与 设 计

3、VHDL测试程序设计

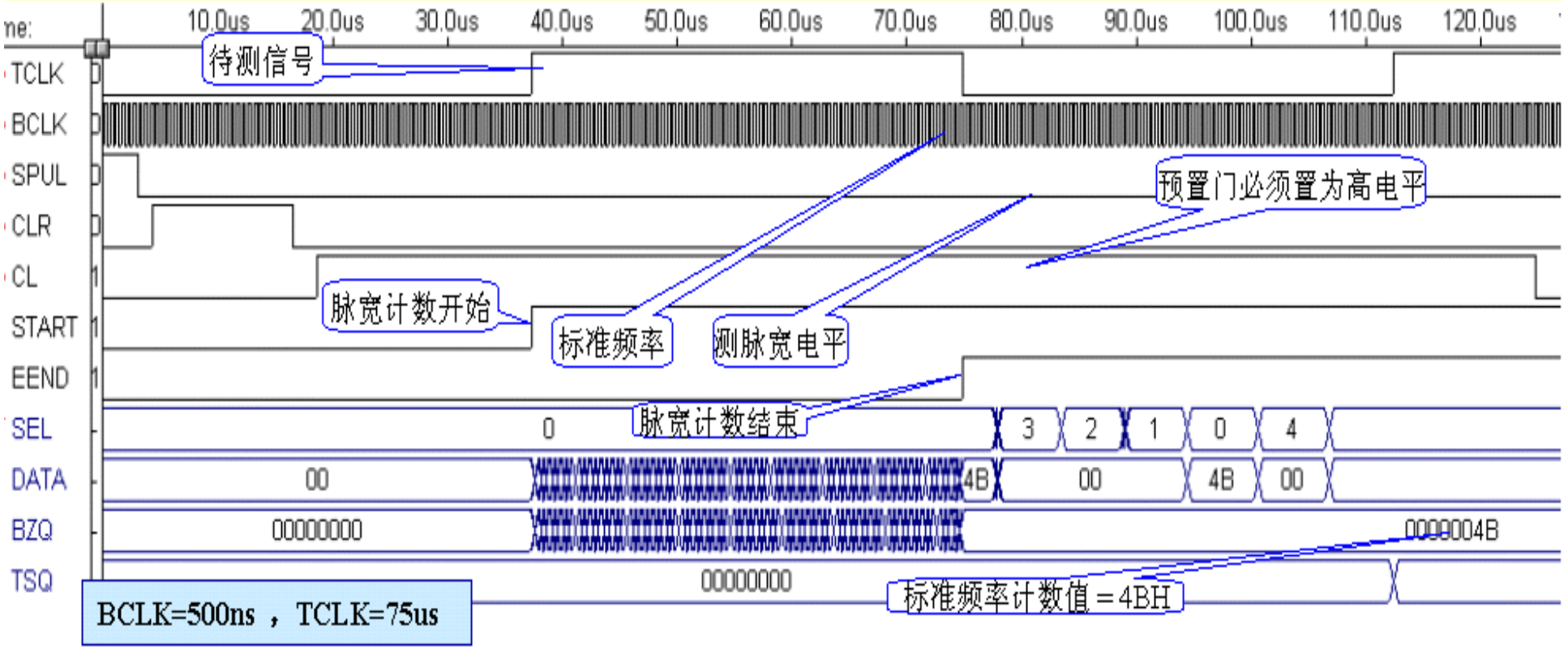


图10-11 等精度频率计测脉宽时序图

Kx 康芯科技

实 验 与 设 计

4、主系统组成测试与实现

5、相位测试

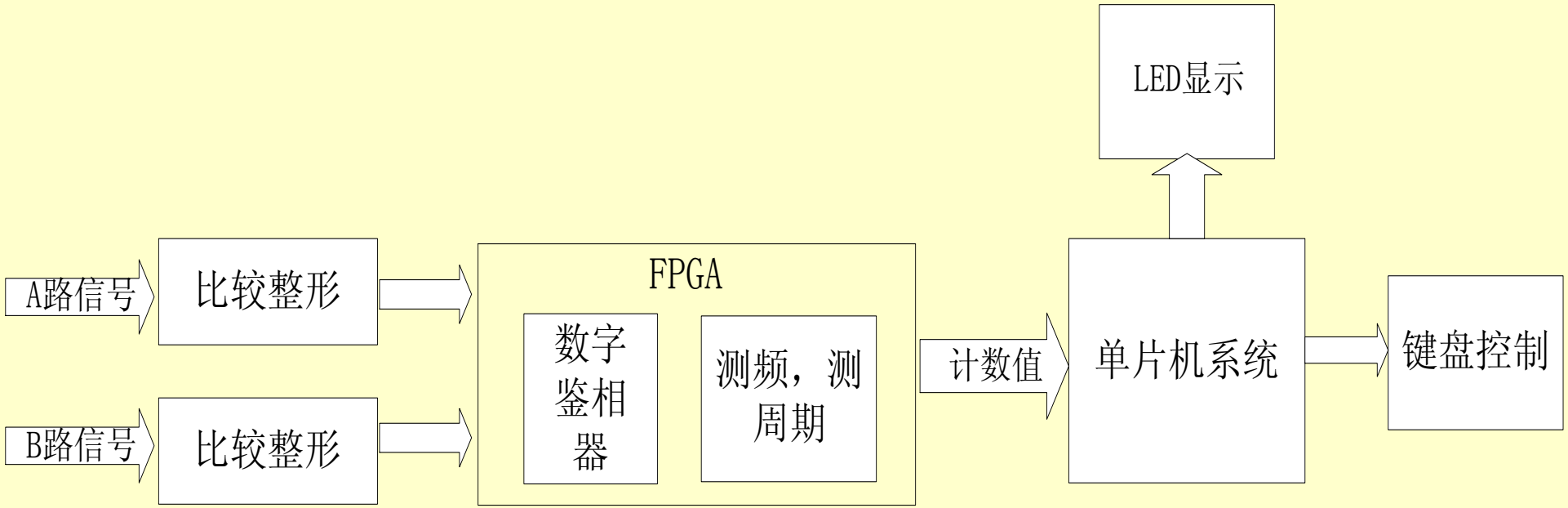


图10-12 测相仪模型

实 验 与 设 计

10-2 等精度频率计/相位计设计

（2）实验内容1：根据以上给出步骤首先完成等精度频率计FPGA的设计，按照图10-11和图10-12的时序，在GW48系统上硬件验证例10-39的各项功能：等精度测频率、测脉宽、测占空比。与GW48系统上给出的标准待测频率，计算误差，并与理论误差值比较。

（3）实验内容2：根据图10-11和图10-12和式10-2、10-3设计单片机程序，完成单片机与FPGA的接口程序、控制程序和计算显示程序的设计。完成等精度频率计独立系统的设计，控制键可以参考图10-6的电路，每一个键控制一种功能。

（4）实验内容3：根据图10-14、10-13、10-12和10-10，修改原设计，增加测相位功能，并在系统上增加一个键，控制测相差和显示。被测信号可以用前面设计的移相信号发生器产生。

（5）实验内容4：用嵌入式锁相环PLL的LPM模块对实验系统的50MHz或20MHz时钟源倍频，达到100MHz，PLL的输出信号作为频率计的标准频率源。注意PLL的输入时钟必须是器件的专用时钟输入脚CLK0→pin16（对于EP1C3TC144）。

实 验 与 设 计

10-2 等精度频率计/相位计设计

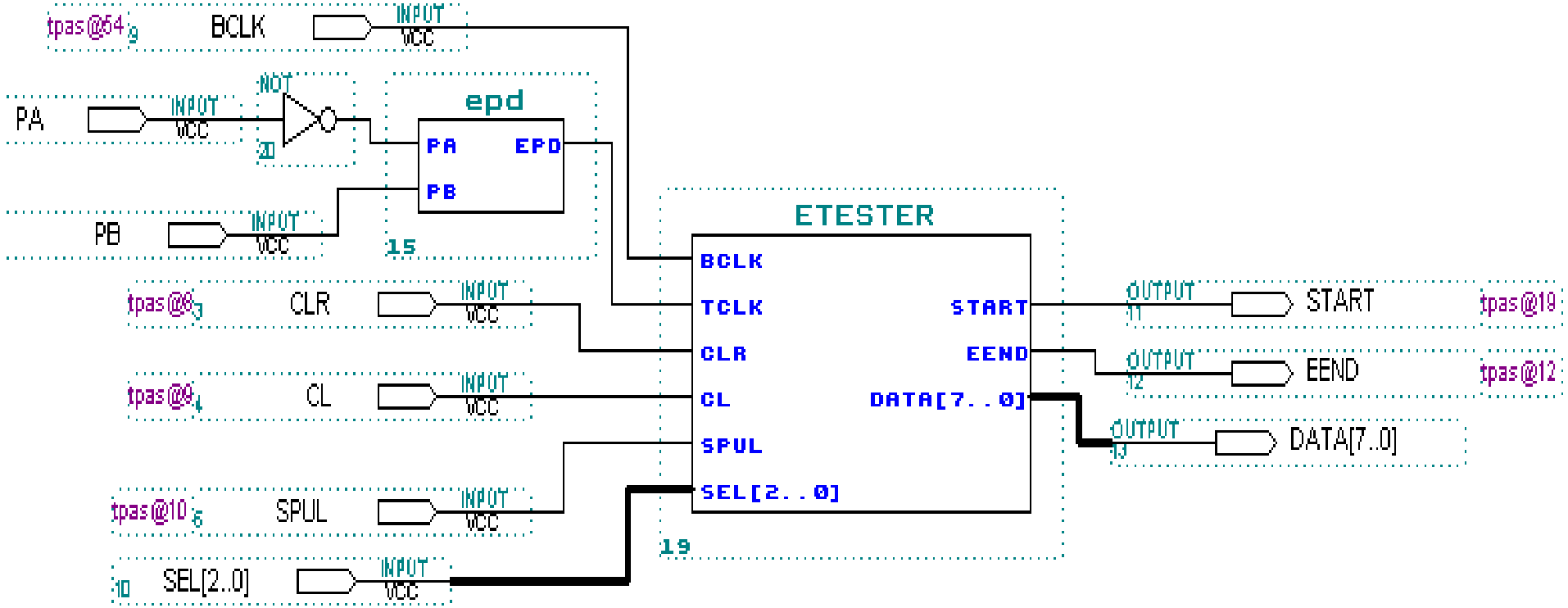


图10-13 测相仪电路原理图（TPAS.bdf工程）

实 验 与 设 计

10-2 等精度频率计/相位计设计

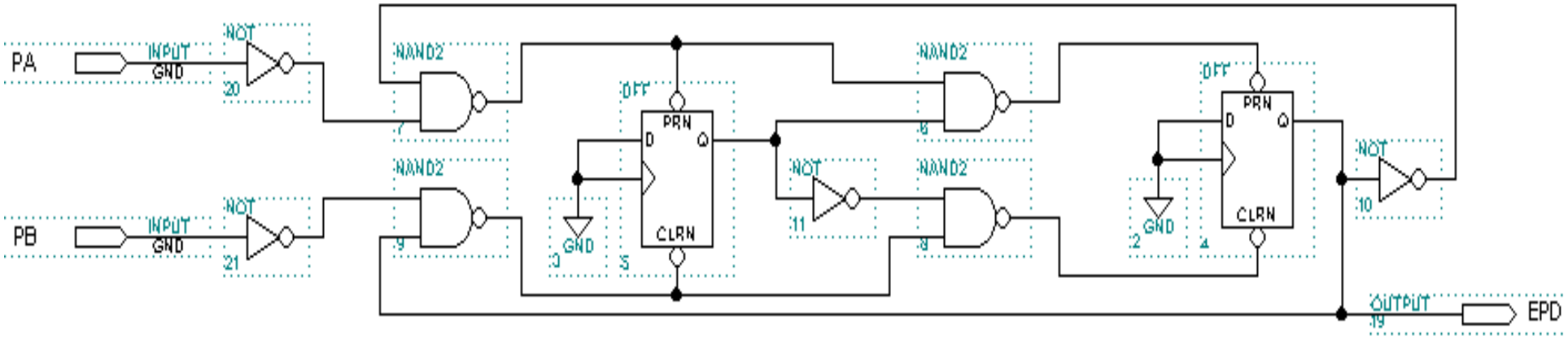


图10-14 相位检测原理图epd

实 验 与 设 计

10-2 等精度频率计/相位计设计

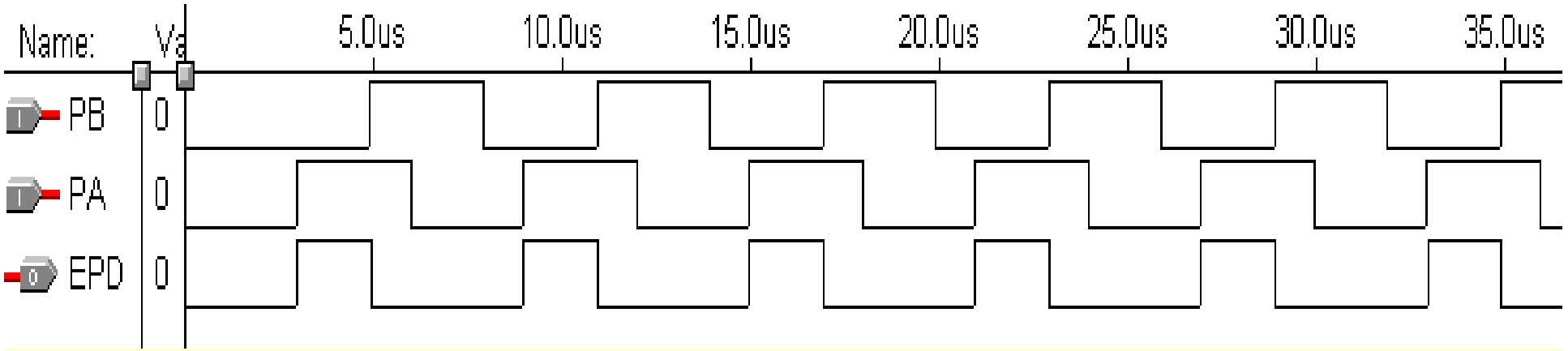


图10-15 鉴相器EPD的仿真波形

实 验 与 设 计

10-3 基于8051单片机IP核的等精度频率计单片系统设计（LCD显示）

(1) 实验原理：如果利用**8051**单片机核，则能将图**10-6**主要元件集成在单片**FPGA**中。图**10-16**是一个含有等精度频率计测试模块的**8051**单片机系统，其中的单片机核已在第**7**章中作了介绍，图中的**ETESTER**模块的**VHDL**源程序是例**10-39**。该模块与单片机的接口方式与前面完全一样 ，因此，除了显示部分，单片机中的汇编程序也完全相同。

（2）实验内容1：对图10-16所示的工程完成验证性实验。设定实验系统为模式3。用QuartusII打开工程文件：/D8051_COPY/GW48PK_LCD/MCU8951，编译后下载，按复位键（键9　0->1->0），用一短线将适配板上方P208与clock0相接，作为等精度频率计信号输入口。

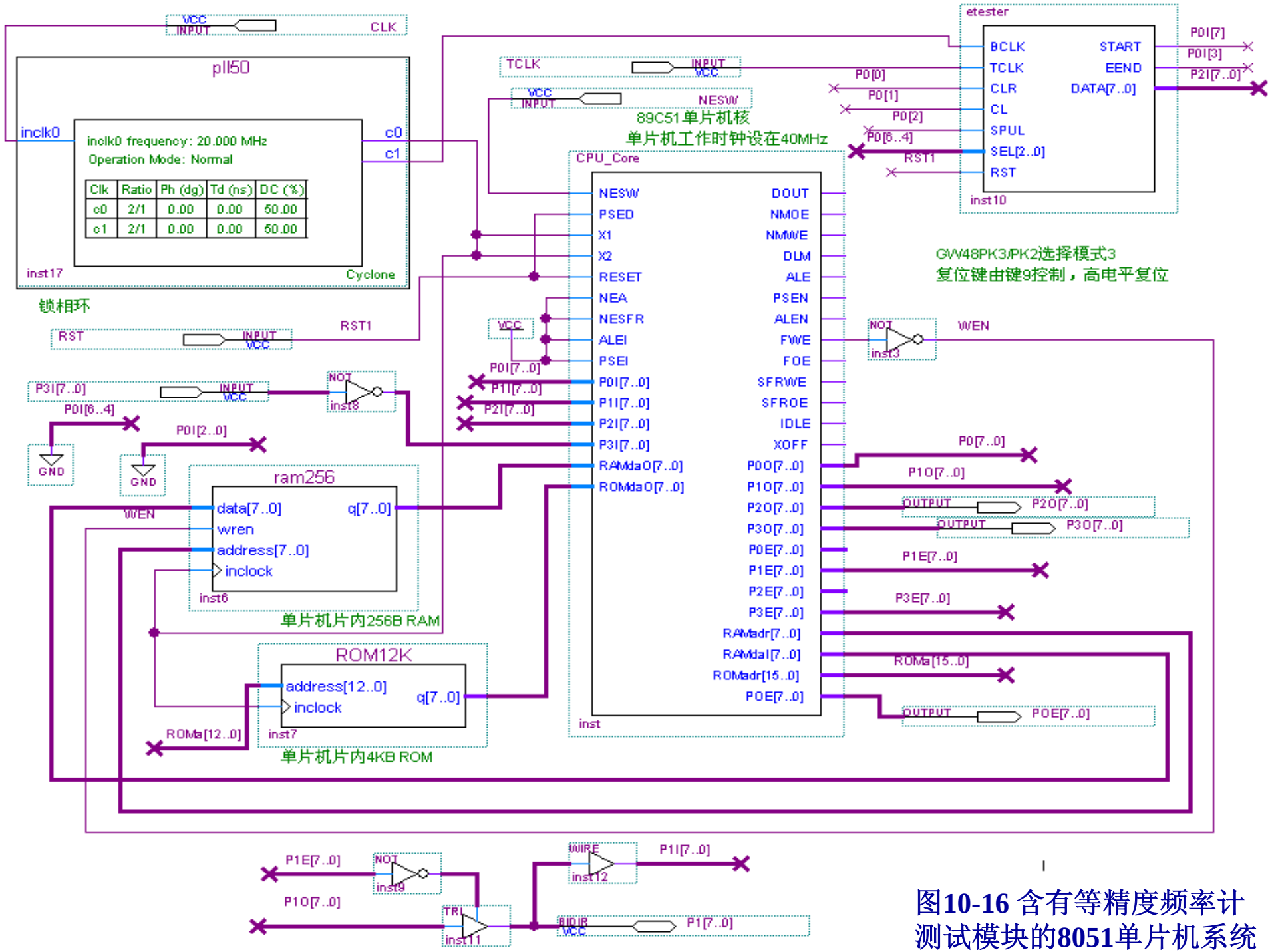


图10-16 含有等精度频率计测试模块的8051单片机系统

实 验 与 设 计

10-3 基于8051单片机IP核的等精度频率计单片系统设计（LCD显示）

（3）实验内容2：修改单片机示例程序，增加一些测试功能，编译、下载，调试软件。

（4）实验内容3：基于图10-16，修改硬件平台，比如ROM大小、模块etester的功能，增加其他接口模块，完成进一步软硬件调试实验。注意，使用单片机核的双向口时必须设置FPGA的端口为上拉！

（5）实验内容4：根据图10-13，加入一个鉴相器模块，增加测相功能。

10-4 基于8051单片机IP核的等精度频率计单片系统设计（LED显示）

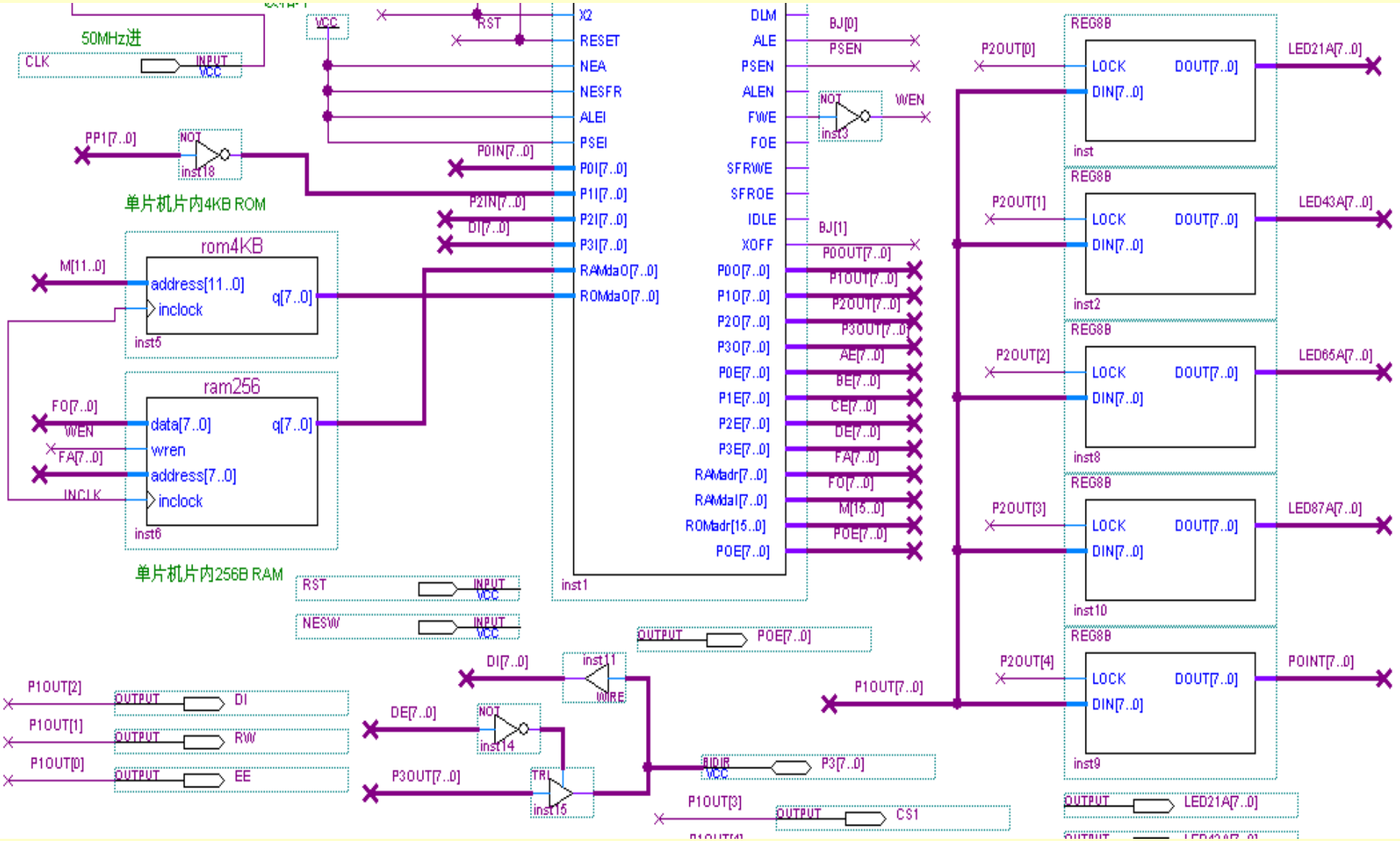


图10-17 含有数码管显示控制电路的等精度频率计单片测试系统

EDA 技术实用教程

第 9 章

VHDL结构与要素

9.1 实体

9.1.1 实体语句结构

实体说明单元的一般语句结构：

```
ENTITY   实体名   IS
 [GENERIC ( 参数名：数据类型 );]
 [PORT ( 端口表 );]
END ENTITY 实体名;
```

9.1 实体

9.1.2 参数传递说明语句

参数传递说明语句的一般书写格式如下：

```
GENERIC([ 常数名 : 数据类型 [ : 设定值 ]
{;常数名:数据类型[:设定值]} );
```

9.1 实体

9.1.2 参数传递说明语句

【例9-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY andn IS
  GENERIC (  n : INTEGER );                          --定义类属参量及其数据类型
   PORT(a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);--用类属参量限制矢量长度
         c : OUT STD_LOGIC);
 END;
 ARCHITECTURE behav OF andn IS
    BEGIN
      PROCESS (a)
          VARIABLE int : STD_LOGIC;
      BEGIN
         int := '1';
            FOR i IN a'LENGTH - 1 DOWNTO 0 LOOP   --循环语句
              IF a(i)='0' THEN   int := '0';
              END IF;
            END LOOP;
              c <=int ;
      END PROCESS;
END;
```

9.1 实体

9.1.2 参数传递说明语句

【例9-2】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY exn IS
   PORT(d1,d2,d3,d4,d5,d6,d7 : IN STD_LOGIC;
                               q1,q2  : OUT STD_LOGIC);
END;
 ARCHITECTURE exn_behav OF exn IS
     COMPONENT andn                    --调用例10-1的元件调用声明
          GENERIC (  n : INTEGER);
         PORT(a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
                C : OUT STD_LOGIC);
      END COMPONENT ;
 BEGIN
   u1: andn GENERIC MAP (n =>2) -- 参数传递映射语句，定义类属变量，n赋值为2
              PORT MAP (a(0)=>d1,a(1)=>d2,c=>q1);
   u2: andn GENERIC MAP (n =>5)      -- 定义类属变量，n赋值为5
              PORT MAP (a(0)=>d3,a(1)=>d4,a(2)=>d5,
                        a(3)=>d6,a(4)=>d7, c=>q2);
END;
```

9.1 实体

9.1.3 参数传递映射语句

【例9-3】

```
LIBRARY IEEE;                                    --待例化元件
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_arith.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY addern IS
    PORT (a, b: IN STD_LOGIC_VECTOR;
        result: out STD_LOGIC_VECTOR);
END addern;
 ARCHITECTURE behave OF addern IS
  BEGIN
      result <= a + b;
 END;
```

9.1.3 参数传递映射语句

【例9-4】

```
LIBRARY IEEE;                          --顶层设计
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_arith.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY adders IS
    GENERIC(msb_operand: INTEGER := 15; msb_sum: INTEGER :=15);
    PORT(b: IN STD_LOGIC_VECTOR (msb_operand DOWNTO 0);
         result: OUT STD_LOGIC_VECTOR (msb_sum DOWNTO 0));
END adders;
 ARCHITECTURE behave OF adders IS
    COMPONENT addern
     PORT (   a, b: IN STD_LOGIC_VECTOR;
            result: OUT STD_LOGIC_VECTOR);
    END COMPONENT;
    SIGNAL a: STD_LOGIC_VECTOR (msb_sum /2 DOWNTO 0);
    SIGNAL twoa: STD_LOGIC_VECTOR (msb_operand DOWNTO 0);
   BEGIN
    twoa <= a & a;
   U1: addern PORT MAP (a => twoa, b => b, result => result);
   U2: addern  PORT MAP (a=>b(msb_operand downto msb_operand/2 +1),
            b=>b(msb_operand/2 downto 0), result => a);
END behave;
```

9.1 实体

9.1.3 参数传递映射语句

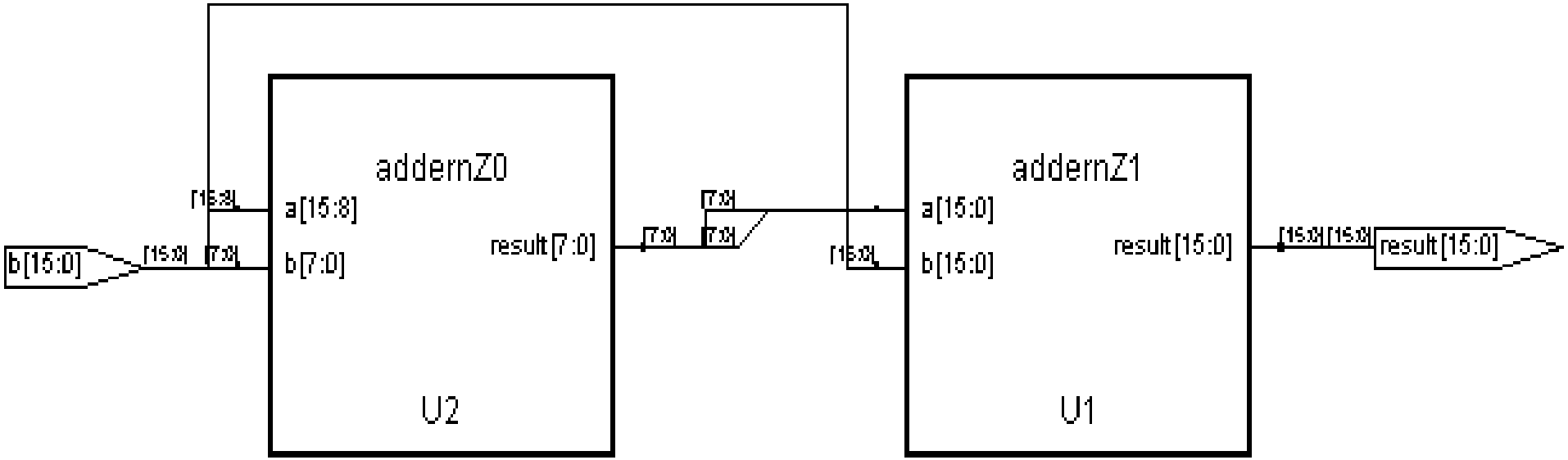


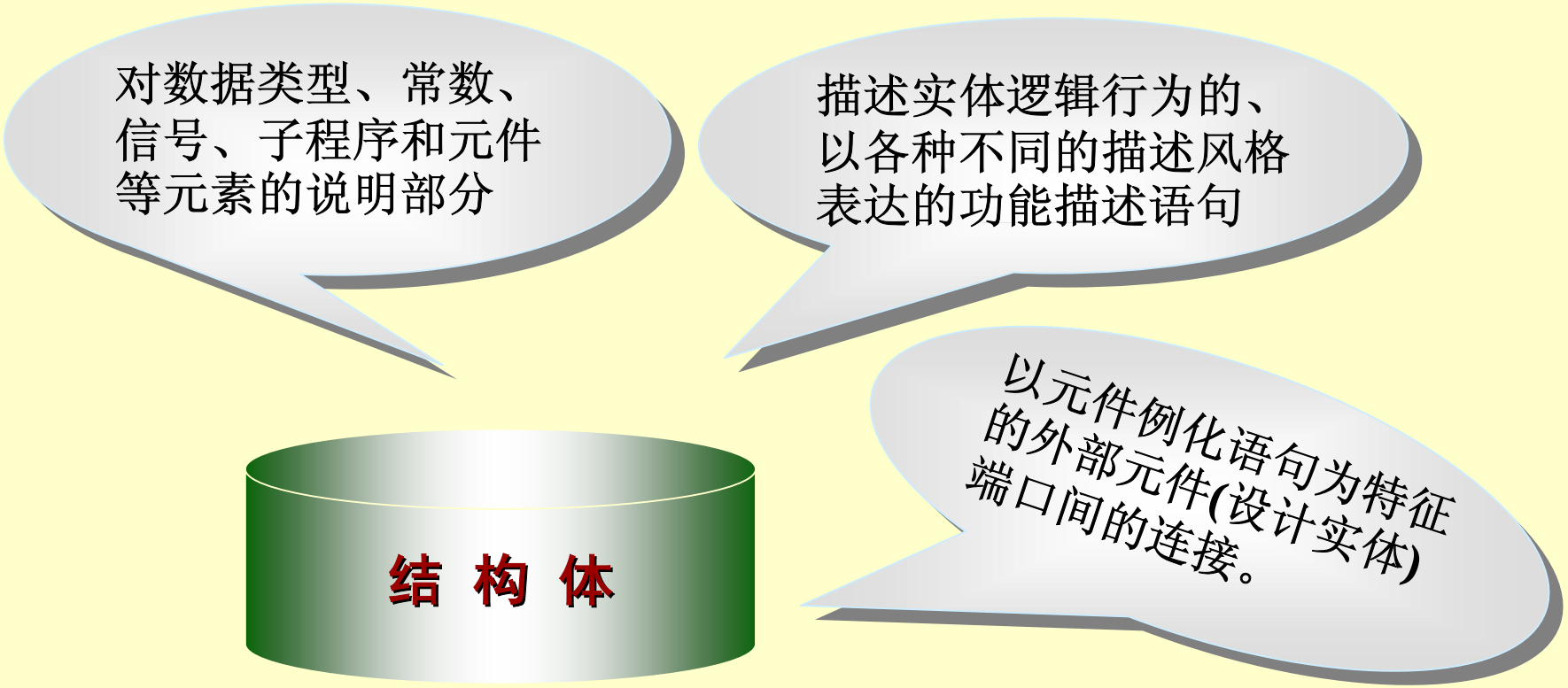
图9-1 例9-4的RTL电路图（Synplify综合）

9.1 实体

9.1.4 端口说明语句

```
PORT ( 端口名 : 端口模式 数据类型 ;
       {端口名:端口模式 数据类型} );
```

9.2 结构体


对数据类型、常数、
信号、子程序和元件
等元素的说明部分
描述实体逻辑行为的、
以各种不同的描述风格
表达的功能描述语句
以元件例化语句为特征
的外部元件(设计实体)
端口间的连接。
结 构 体

9.2 结构体

1. 结构体的一般语言格式

结构体的语句格式如下：

```
ARCHITECTURE 结构体名 OF 实体名 IS
        [说明语句]
BEGIN
        [功能描述语句]
END ARCHITECTURE 结构体名;
```

9.2 结构体

2. 结构体说明语句

3. 功能描述语句结构

- 进程语句
- 信号赋值语句
- 子程序调用语句
- 元件例化语句

9.3 子程序

9.3.1 函数

函数的语句表达格式如下：

```
FUNCTION  函数名(参数表)  RETURN   数据类型          --函数首
FUNCTION   函数名(参数表)RETURN   数据类型  IS      --  函数体
    [  说明部分  ]
    BEGIN
    顺序语句  ;
  END FUNCTION  函数名;
```

【例9-5】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE  packexp IS                                        --定义程序包
   FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)   --定义函数首
     RETURN STD_LOGIC_VECTOR ;
   FUNCTION  func1 ( a,b,c : REAL )              --定义函数首
     RETURN REAL ;
   FUNCTION  "*"  ( a ,b : INTEGER )             --定义函数首
     RETURN INTEGER  ;
   FUNCTION  as2  (SIGNAL  in1 ,in2 : REAL )    --定义函数首
     RETURN REAL  ;
 END ;
PACKAGE  BODY packexp IS
   FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)    --定义函数体
     RETURN STD_LOGIC_VECTOR IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;
     END IF;
   END FUNCTION max;          --结束FUNCTION语句
 END;                            --结束PACKAGE  BODY语句
```

（接下页）

```
LIBRARY IEEE; -- 函数应用实例
 USE IEEE.STD_LOGIC_1164.ALL;
 USE WORK.packexp.ALL ;
 ENTITY  axamp IS
   PORT(dat1,dat2 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dat3,dat4 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        out1,out2 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
  END;
  ARCHITECTURE bhv OF axamp IS
    BEGIN
     out1 <=  max(dat1,dat2);  --用在赋值语句中的并行函数调用语句
    PROCESS(dat3,dat4)
    BEGIN
     out2 <= max(dat3,dat4);       --顺序函数调用语句
    END PROCESS;
 END;
```

9.3 子程序

9.3.1 函数

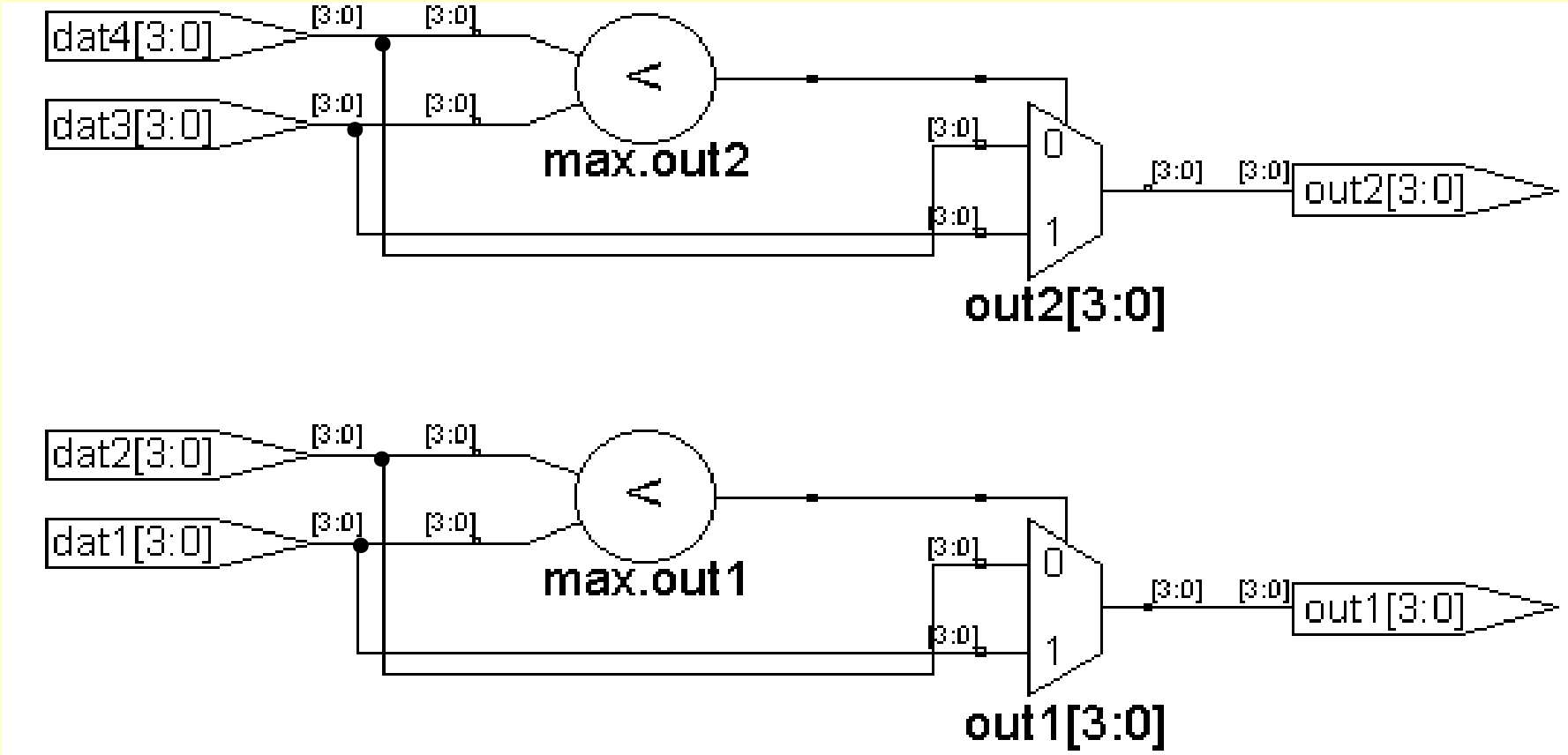


图9-2 例9-5的逻辑电路图

【例9-6】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY func IS
  PORT  ( a :   IN STD_LOGIC_VECTOR (0 to 2 ) ;
           m :  OUT STD_LOGIC_VECTOR (0 to 2 ) );
 END ENTITY func ;
 ARCHITECTURE demo OF func IS
 FUNCTION sam(x ,y ,z : STD_LOGIC) RETURN STD_LOGIC IS
 BEGIN
   RETURN ( x AND y ) OR z  ;
 END FUNCTION sam ;
 BEGIN
   PROCESS ( a )
   BEGIN
    m(0) <= sam( a(0),  a(1),  a(2) ) ;
    m(1) <= sam( a(2),  a(0),  a(1) ) ;
    m(2) <= sam( a(1),  a(2),  a(0) ) ;
   END PROCESS ;
END ARCHITECTURE demo ;
```

9.3 子程序

9.3.2 重载函数

【例9-7】（MaxplusII不支持本例）

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
PACKAGE  packexp IS                                    --定义程序包
 FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)    --定义函数首
   RETURN STD_LOGIC_VECTOR ;
 FUNCTION  max( a,b : IN BIT_VECTOR)           --定义函数首
   RETURN BIT_VECTOR ;
 FUNCTION  max( a,b : IN INTEGER )             --定义函数首
   RETURN INTEGER ;
 END;
PACKAGE BODY packexp IS
 FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)    --定义函数体
   RETURN STD_LOGIC_VECTOR IS
 BEGIN
   IF a > b THEN RETURN  a;
```

（接下页）

```
ELSE          RETURN  b;     END IF;
    END FUNCTION max;                          --结束FUNCTION语句
FUNCTION  max( a,b : IN INTEGER)                 --定义函数体
     RETURN INTEGER IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;     END IF;
    END FUNCTION max;                            --结束FUNCTION语句
    FUNCTION  max( a,b : IN BIT_VECTOR)            --定义函数体
     RETURN BIT_VECTOR IS
   BEGIN
     IF a > b THEN RETURN  a;
       ELSE          RETURN  b;     END IF;
    END FUNCTION max;                            --结束FUNCTION语句
  END;                                        --结束PACKAGE  BODY语句

  -- 以下是调用重载函数max的程序:
  LIBRARY IEEE ;
  USE IEEE.STD_LOGIC_1164.ALL ;
   USE WORK.packexp.ALL;
  ENTITY  axamp IS
```

（接下页）

```
  PORT(a1,b1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            a2,b2 : IN BIT_VECTOR(4 DOWNTO 0);
            a3,b3 : IN INTEGER RANGE 0 TO 15;
                c1 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                c2 : OUT BIT_VECTOR(4 DOWNTO 0);
                c3 : OUT INTEGER RANGE 0 TO 15);
     END;
    ARCHITECTURE bhv OF axamp IS
      BEGIN
      c1 <=  max(a1,b1); --对函数max( a,b : IN STD_LOGIC_VECTOR)的
调用
      c2 <=  max(a2,b2); --对函数max( a,b : IN BIT_VECTOR) 的调用
      c3 <=  max(a3,b3); --对函数max( a,b : IN INTEGER)  的调用
   END;
```

【例9-8】

```
LIBRARY IEEE ;                                          -- 程序包首
USE IEEE.std_logic_1164.all ;
USE IEEE.std_logic_arith.all ;
PACKAGE STD_LOGIC_UNSIGNED is
function "+" (L : STD_LOGIC_VECTOR ; R : INTEGER)
                      return STD_LOGIC_VECTOR ;
function "+" (L : INTEGER;  R : STD_LOGIC_VECTOR)
                      return STD_LOGIC_VECTOR ;
function "+" (L : STD_LOGIC_VECTOR ; R : STD_LOGIC )
return STD_LOGIC_VECTOR ;
function SHR (ARG : STD_LOGIC_VECTOR ;
  COUNT : STD_LOGIC_VECTOR ) return STD_LOGIC_VECTOR ;
...
end STD_LOGIC_UNSIGNED  ;

LIBRARY IEEE ;                                          --  程序包体
```

（接下页）

```
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
package body STD_LOGIC_UNSIGNED is
function maximum (L, R :  INTEGER) return  INTEGER  is
begin
    if L > R then  return L;
        else          return R;
    end if;
end;
function "+" (L : STD_LOGIC_VECTOR ;  R : INTEGER)
return  STD_LOGIC_VECTOR  is
Variable result : STD_LOGIC_VECTOR (L'range) ;
Begin
  result :=     UNSIGNED(L) + R ;
  return std_logic_vector(result) ;
end ;
...
end STD_LOGIC_UNSIGNED ;
```

9.3 子程序

9.3.3 转换函数

表9-1 IEEE库类型转换函数表

函数名	功能
程序包: STD_LOGIC_1164	
to_stdlogicvector(A)	由bit_vector类型的A转换为std_logic_vector
to_bitvector(A)	由std_logic _vector转换为bit_vector
to_stdlogic (A)	由bit转换成std_logic
to_bit(A)	由std_logic转换成bit
程序包: STD_LOGIC_ARITH	
conv_std_logic_vector(A, 位长)	将整数integer转换成std_logic_vector类型，A是整数
conv_integer(A)	将std_logic_vector转换成整数integer
程序包: STD_LOGIC_UNSIGNED	
conv_integer(A)	由std_logic_vector转换成integer

KX 康芯科技

9.3 子程序

9.3.3 转换函数

【例9-9】

```
LIBRARY IEEE;
USE IEEE. std_logic_1164.ALL;
ENTITY exg IS
  PORT (a,b : in bit_vector(3 downto 0);
          q   : out std_logic_vector(3 downto 0));
end ;
architecture rtl of exg is
begin
  q<= to_stdlogicvector(a and b);--将位矢量数据类型转换成标准逻辑位矢量数据
end;
```

9.3 子程序

9.3.3 转换函数

【例9-10】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;--注意使用了此程序包
ENTITY  axamp IS
   PORT(a,b,c : IN integer range 0 to 15 ;
        q   : OUT std_logic_vector(3 downto 0) );
  END;
ARCHITECTURE bhv OF axamp IS
    BEGIN
     q  <=   conv_std_logic_vector(a,4)  when  conv_integer(c)=8
else
  conv_std_logic_vector(b,4) ;
  END;
```

【例9-11】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE n_pack IS
  SUBTYPE nat IS Integer range 0 to 255;   --定义一个Integer的子类型
    TYPE Bit8 IS array (7 downto 0) OF std_logic;-- 定义一个数据类型
  FUNCTION nat_to_Bit8 (s: nat) RETURN Bit8;
End n_pack;
  PACKAGE BODY n_pack IS
  FUNCTION nat_to_Bit8 (s: nat) RETURN Bit8 IS
    VARIABLE Din: Integer range 255 downto 0;
    VARIABLE Rut: Bit8;
    VARIABLE Rig: Integer :=2**7;
 BEGIN
    Din := s;
  FOR I in 7 downto 0 LOOP
    IF Din/Rig > 1 THEN Rut(i) := '1'; Din := Din-Rig;
      ELSE Rut (i):= '0';  END IF;
        Rig := Rig / 2;
```

（接下页）

```
END LOOP;
  RETURN Rut;
 END nat_to_Bit8;
END n_pack;

LIBRARY IEEE; -- 用户定义转换函数应用实例
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.n_pack.ALL ;
ENTITY  axamp IS
   PORT(dat : IN nat;    --注意数据类型的定义
        ou  : OUT Bit8); --注意数据类型的定义
  END;
ARCHITECTURE bhv OF axamp IS
    BEGIN
     ou <=  nat_to_Bit8(dat);
END;
```

9.3 子程序

9.3.4 决断函数

9.3.5 过程

```
PROCEDURE 过程名(参数表)                          -- 过程首

PROCEDURE 过程名(参数表) IS
    [说明部分]
    BIGIN                                          -- 过程体
      顺序语句;
  END PROCEDURE 过程名;
```

9.3 子程序

9.3.5 过程

```
PROCEDURE  pro1  (VARIABLE  a, b :  INOUT REAL) ;
PROCEDURE  pro2 (CONSTANT  a1 :  IN INTEGER ;
                         VARIABLE  b1 :  OUT INTEGER ) ;
PROCEDURE  pro3  (SIGNAL sig :  INOUT BIT) ;
```

【例9-12】

```
PROCEDURE  prg1(VARIABLE value:INOUT BIT_VECTOR(0 TO 7)) IS
BEGIN
 CASE value IS
  WHEN "0000" => value: "0101" ;
  WHEN "0101" => value: "0000" ;
  WHEN OTHERS => value: "1111" ;
     END CASE ;
END PROCEDURE  prg1 ;
```

9.3 子程序

9.3.5 过程

【例9-13】

```
  PROCEDURE  comp ( a, r : IN REAL;
                       m : IN INTEGER ;
                   v1, v2: OUT REAL) IS
  VARIABLE cnt : INTEGER ;
  BEGIN
  v1 := 1.6 * a ;                                  -- 赋初始值
  v2 := 1.0 ;                                      -- 赋初始值
  Q1 :  FOR  cnt  IN  1 TO m  LOOP
   v2 :=  v2 * v1 ;
EXIT  Q1  WHEN  v2 > v1;          -- 当v2 > v1，跳出循环LOOP
   END LOOP Q1
 ASSERT (v2 < v1 )
     REPORT  "OUT OF RANGE"                -- 输出错误报告
         SEVERITY  ERROR ;
END PROCEDURE comp ;
```

【例9-14】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE axamp IS                                          过程首定义
 PROCEDURE nand4a (SIGNAL a,b,c,d : IN STD_LOGIC ;
                   SIGNAL  y : OUT  STD_LOGIC );
 END axamp;
PACKAGE BODY axamp IS                                  --过程体定义
 PROCEDURE nand4a  (SIGNAL a,b,c,d : IN STD_LOGIC ;
                   SIGNAL  y : OUT  STD_LOGIC ) IS
 BEGIN
   y<= NOT(a AND b AND c AND d);
 RETURN;
END nand4a;
END axamp;
LIBRARY IEEE;                                            --主程序
USE IEEE.STD_LOGIC_1164.ALL;
USE WORK.axamp.ALL;
 ENTITY EX IS
  PORT( e,f,g,h : IN STD_LOGIC ;
                x : OUT  STD_LOGIC );
 END;
ARCHITECTURE bhv OF EX IS
    BEGIN
     nand4a(e,f,g,h,x) ;                                 并行调用过程
END;
```

9.3 子程序

9.3.6 重载过程

【例9-15】

```
PROCEDURE  calcu (  v1, v2 : IN REAL ;
                    SIGNAL out1 : INOUT INTEGER) ;
PROCEDURE  calcu (  v1, v2 : IN INTEGER ;
                    SIGNAL out1 : INOUT REAL) ;
...
calcu (20.15, 1.42, signl) ;    -- 调用第一个重载过程calcu
calcu (23,  320, sign2 ) ;      -- 调用第二个重载过程 calcu
```

…

9.4 VHDL库

9.4.1 库的种类

1. IEEE库

2. STD库

```
LIBRARY STD ;
USE STD.STANDARD.ALL ;
```

3. WORK库

4. VITAL库

9.4 VHDL库

9.4.2 库的用法

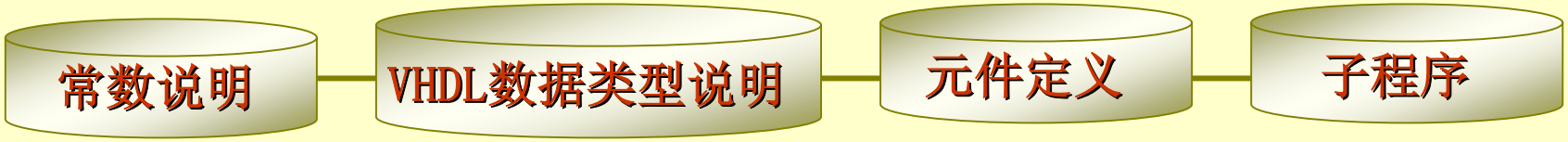
USE 库名.程序包名.项目名；

USE 库名.程序包名.ALL；

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.STD_ULOGIC ;
USE IEEE.STD_LOGIC_1164.RISING_EDGE ;
```

```
USE WORK.std_logic_1164.ALL;
```

9.5 程序包



定义程序包的一般语句结构如下：

```
PACKAGE   程序包名   IS                    --   程序包首
      程序包首说明部分
END   程序包名;
PACKAGE BODY   程序包名   IS          --   程序包体
程序包体说明部分以及包体内
    END 程序包名;
```

9.5 程序包

【例9-16】

```
PACKAGE pacl IS                                          -- 程序包首开始
  TYPE byte IS RANGE 0 TO 255 ;                          -- 定义数据类型byte
  SUBTYPE nibble IS byte RANGE 0 TO 15 ;                 -- 定义子类型nibble
CONSTANT byte_ff  : byte := 255 ;                        -- 定义常数byte_ff
SIGNAL addend : nibble ;                                 -- 定义信号addend
COMPONENT byte_adder                                     -- 定义元件
PORT( a, b : IN byte ;
        c : OUT byte  ;
  overflow : OUT BOOLEAN ) ;
  END COMPONENT ;
  FUNCTION my_function (a : IN byte) Return byte ;       -- 定义函数
END pacl ;                                               -- 程序包首结束
```

【例9-17】

```
PACKAGE  seven IS
    SUBTYPE segments is BIT_VECTOR(0 TO 6) ;
    TYPE bcd IS RANGE 0 TO 9 ;
END seven ;
USE WORK.seven.ALL ; -- WORK库默认是打开的,
ENTITY decoder IS
    PORT (input: bcd; drive : out segments) ;
END decoder ;
ARCHITECTURE simple OF decoder IS
BEGIN
    WITH input SELECT
      drive <=  B"1111110"  WHEN 0 ,
                B"0110000"  WHEN 1 ,
                B"1101101"  WHEN 2 ,
                B"1111001"  WHEN 3 ,
                B"0110011"  WHEN 4 ,
                B"1011011"  WHEN 5 ,
                B"1011111"  WHEN 6 ,
                B"1110000"  WHEN 7 ,
                B"1111111"  WHEN 8 ,
                B"1111011"  WHEN 9 ,
                B"0000000"  WHEN  OTHERS ;
END simple ;
```

9.5 程序包

(1) STD_LOGIC_1164程序包。

(2) STD_LOGIC_ARITH程序包。

(3) STD_LOGIC_UNSIGNED和STD_LOGIC_SIGNED程序包。

(4) STANDARD和TEXTIO程序包。

9.6 配置

配置语句的一般格式如下：

CONFIGURATION 配置名 **OF** 实体名 **IS**

配置说明

END 配置名;

9.7 VHDL文字规则

9.7.1 数字

整数 — 5， 678， 0， 156E2(=15600)， 45_234_287 (=45234287)

实数 — 1.335， 88_670_551.453_909(=88670551.453909)，1.0，44.99E-2(=0.4499)

以数制基数表示的文字

```
SIGNAL d1,d2,d3,d4,d5, : INTEGER RANGE 0 TO 255;
d1 <= 10#170# ;              -- (十进制表示，等于 170)
d2 <= 16#FE#    ;            -- (十六进制表示，等于 254)
d3 <= 2#1111_1110#;          -- (二进制表示，等于 254)
d4 <= 8#376#        ;        -- (八进制表示，等于 254)
d5 <= 16#E#E1    ;      -- (十六进制表示，等于2#1110000#，等于224)
```

物理量文字(VHDL综合器不接受此类文字) — 60s (60秒)， 100m (100米)， k (千欧姆)， 177A (177安培)

康芯科技

9.7 VHDL文字规则

9.7.2 字符串

(1) 文字字符串

"ERROR" ， "Both S and Q equal to 1" ， "X" ， "BB$CC"

(2) 数位字符串

```
data1 <= B"1_1101_1110"          -- 二进制数数组，位矢数组长度是9
data2 <= O"15"                   -- 八进制数数组，位矢数组长度是6
data3 <= X"AD0"                  -- 十六进制数数组，位矢数组长度是12
data4 <= B"101_010_101_010"      -- 二进制数数组，位矢数组长度是12
data5 <= "101_010_101_010"       --表达错误，缺B。
data6 <= "0AD0"                  --表达错误，缺X。
```

9.7 VHDL文字规则

9.7.3 标识符

合法的标识符：Decoder_1， FFT， Sig_N， Not_Ack， State0 ， Idle

非法的标识符：

```
_Decoder_1          -- 起始为非英文字母
 2FFT               -- 起始为数字
Sig_#N              -- 符号“#”不能成为标识符的构成
Not-Ack             -- 符号“-” 不能成为标识符的构成
RyY_RST_            -- 标识符的最后不能是下划线“_”
data_ _BUS          -- 标识符中不能有双下划线
return              -- 关键词
```

9.7 VHDL文字规则

9.7.4 下标名

标识符(表达式)

```
SIGNAL  a, b : BIT_VECTOR (0 TO 3) ;
SIGNAL  m    : INTEGER  RANGE 0 TO 3 ;
SIGNAL  y, z : BIT ;
y <= a(m) ;                    --  不可计算型下标表示
z <= b(3) ;          -- 可计算型下标表示
```

9.8 数据类型

- 标量型(Scalar Type) — 实数类型、整数类型、枚举类型、时间类型
- 复合类型(Composite Type) — 数组型(Array)、记录型(Record)
- 存取类型(Access Type) — 为给定的数据类型的数据对象提供存取方式
- 文件类型(Files Type) — 用于提供多值存取类型

9.8 数据类型

9.8.1 预定义数据类型

1. 布尔类型

```
TYPE BOOLEAN IS (FALSE, TRUE);
```

2. 位数据类型

```
TYPE BIT IS ('0', '1');
```

3. 位矢量类型

```
TYPE BIT_VECTOR IS ARRAY (Natural Range <> ) OF  BIT ;
```

4. 字符类型

```
'A'
```

5. 整数类型

－2147483647～＋2147483647

9.8 数据类型

6. 实数类型

1.0	十进制浮点数
0.0	十进制浮点数
65971.333333	十进制浮点数
65_971.333_3333	与上一行等价
8#43.6#e+4	八进制浮点数
43.6E－4	十进制浮点数

7. 字符串类型

```
VARIABLE string_var : STRING (1 TO 7 ) ;
string_var := "a b c d" ;
```

9.8 数据类型

8. 时间类型

```
TYPE time IS RANGE －2147483647 TO 2147483647
    units
        fs ;  -- 飞秒，VHDL中的最小时间单位
        ps = 1000 fs ;          -- 皮秒
        ns = 1000 ps ;          -- 纳秒
        us = 1000 ns ;          -- 微秒
        ms = 1000 us ;          -- 毫秒
        sec = 1000 ms ;         -- 秒
        min = 60 sec ;          -- 分
        hr = 60 min ;           -- 时
end units ;
```

9.8 数据类型

9. 文件类型

```
PROCEDUER Readline (F: IN TEXT; L: OUT LINE);
PROCEDUER Writeline (F: OUT TEXT; L: IN LINE);
PROCEDUER Read (   L: INOUT LINE; Value: OUT std_logic;
Good: OUT BOOLEAN);
PROCEDUER Read (L: INOUT LINE; Value: OUT std_logic);
PROCEDUER Read (   L: INOUT LINE; Value: OUT std_logic_ vector;
Good: OUT BOOLEAN);
PROCEDUER Read (L: INOUT LINE; Value: OUT std_logic_ vector;
PROCEDUER Write (          L: INOUT LINE; Value: IN std_logic;
Justiaied: IN SIDE :=Right;field; IN WIDTH :=0);
PROCEDUER Write (L: INOUT LINE; Value: IN std_logic _ vector,
  Justiaied: IN SIDE :=Right;field; IN WIDTH :=0);
```

9.8 数据类型

9.8.2 IEEE预定义标准逻辑位与矢量

1. 标准逻辑位数据类型

2. 标准逻辑矢量数据类型

```
TYPE STD_LOGIC_VECTOR IS ARRAY ( NATURAL RANGE <> ) OF STD_LOGIC ;
```

9.8 数据类型

9.8.3 其他预定义标准数据类型

1. 无符号数据类型(UNSIGNED TYPE)　　UNSIGNED'("1000")

```
VARIABLE var : UNSIGNED(0 TO 10) ;
SIGNAL  sig : UNSIGNED(5 TO 0) ;
```

2. 有符号数据类型(SIGNED TYPE)

SIGNED'("0101") 代表 +5，5

SIGNED'("1011") 代表 –5

```
VARIABLE var : SIGNED(0 TO 10);
```

9.8 数据类型

9.8.4 数组类型

TYPE 数组名IS ARRAY(数组范围)OF数据类型

TYPE stb IS ARRAY (7 DOWNTO 0) of STD_LOGIC ;

```
TYPE x is (low，high) ;
```

TYPE data_bus IS ARRAY (0 TO 7，x) of BIT ;

TYPE 数组名IS ARRAY (数组下标名RANGE <>)OF 数据类型 ;

9.8 数据类型

9.8.4 数组类型

【例9-18】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY  amp IS
   PORT (     a1，a2 : IN BIT_VECTOR(3 DOWNTO 0);
          c1，c2,c3 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
           b1，b2，b3 : INTEGER RANGE 0 TO 15;
         d1,d2,d3,d4 : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)     );
END amp;
 d1 <= TO_STDLOGICVECTOR(a1 AND a2);                --(1)
 d2 < = CONV_STD_LOGIC_VECTOR(b1,4) WHEN CONV_INTEGER(b2)=9
          else   CONV_STD_LOGIC_VECTOR(b3,4);       --(2)
 d3 < = c1 WHEN CONV_INTEGER(c2)= 8 ELSE c3;      --(3)
 d4 < = c1 WHEN  c2 = 8 else c3;                  --(4)
```

9.8 数据类型

9.8.4 数组类型

【例9-19】

```
 LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY decoder3to8 IS
 PORT (  input: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
              output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decoder3to8;
ARCHITECTURE behave OF decoder3to8 IS
  BEGIN
    PROCESS (input)
      BEGIN
             output <= (OTHERS => '0');
            output(CONV_INTEGER(input)) <= '1';
    END PROCESS;
 END behave;
```

9.8 数据类型

【例9-20】

```
  FUNCTION To_bit ( s : std_ulogic; xmap : BIT := '0' ) RETURN BIT ;
  FUNCTION To_bitvector ( s : std_logic_vector ;
                          xmap : BIT := '0' ) RETURN BIT_VECTOR ;
  FUNCTION To_bitvector ( s : std_ulogic_vector ;
                          xmap : BIT := '0' ) RETURN BIT_VECTOR ;
```

下面是转换函数To_bitvector的函数体：

```
FUNCTION To_bitvector ( s : std_logic_vector ;
                        xmap : BIT := '0' )
                        RETURN BIT_VECTOR IS
    ALIAS sv : std_logic_vector(s'LENGTH-1 DOWNTO 0 ) IS s ;
    VARIABLE result : BIT_VECTOR(s'LENGTH-1 DOWNTO 0 );
BEGIN
    FOR i IN result'RANGE LOOP
        CASE sv(i) IS
            WHEN '0'|'L' => result(i) := '0';
            WHEN '1'|'H' => result(i) := '1';
            WHEN OTHERS => result(i) := xmap;
        END CASE ;
    END LOOP ;
    RETURN result ;
END;
```

9.9 操作符

9.9.1 逻辑操作符

表9-2 VHDL操作符列表

类　型	操作符	功　能	操作数数据类型
算术操作符	+	加	整数
	—	减	整数
	&	并置	一维数组
	*	乘	整数和实数(包括浮点数)
	/	除	整数和实数(包括浮点数)
	MOD	取模	整数
	REM	取余	整数
	SLL	逻辑左移	BIT或布尔型一维数组
	SRL	逻辑右移	BIT或布尔型一维数组
	SLA	算术左移	BIT或布尔型一维数组

（接下页）

（接上页）

算术操作符	SRA	算术右移	BIT或布尔型一维数组
	ROL	逻辑循环左移	BIT或布尔型一维数组
	ROR	逻辑循环右移	BIT或布尔型一维数组
	**	乘方	整数
	ABS	取绝对值	整数
关系操作符	=	等于	任何数据类型
	/=	不等于	任何数据类型
	<	小于	枚举与整数类型，及对应的一维数组
	>	大于	枚举与整数类型，及对应的一维数组
	<=	小于等于	枚举与整数类型，及对应的一维数组
	>=	大于等于	枚举与整数类型，及对应的一维数组
逻辑操作符	AND	与	BIT，BOOLEAN，STD_LOGIC
	OR	或	BIT，BOOLEAN，STD_LOGIC
	NAND	与非	BIT，BOOLEAN，STD_LOGIC
	NOR	或非	BIT，BOOLEAN，STD_LOGIC
	XOR	异或	BIT，BOOLEAN，STD_LOGIC
	XNOR	异或非	BIT，BOOLEAN，STD_LOGIC
	NOT	非	BIT，BOOLEAN，STD_LOGIC
符号操作符	+	正	整数
	—	负	整数

9.9 操作符

9.9.1 逻辑操作符

表9-3 VHDL操作符优先级

运算符	优先级
NOT，ABS，**	最高优先级
* ，/ ，MOD， REM	
+(正号)， －(负号)	
+ ， － ， &	
SLL， SLA， SRL， SRA， ROL， ROR	
=， /=， <， <=， >， >=	
AND， OR， NAND， NOR， XOR， XNOR	最低优先级

9.9 操作符

9.9.1 逻辑操作符

【例9-21】

```
SIGNAL a , b, c : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL d, e, f, g : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
SIGNAL h, I, j, k : STD_LOGIC ;
SIGNAL l, m, n, o, p : BOOLEAN ;
...
a<=b AND c;  --b、c 相与后向a赋值，a、b、c的数据类型同属4位长的位矢量
d<=e OR f OR g ;                -- 两个操作符OR相同，不需括号
h<=(i NAND j)NAND k ;     -- NAND不属上述三种算符中的一种，必须加括号
l<=(m XOR n)AND(o XOR p); -- 操作符不同，必须加括号
h<=i AND j AND k ;              -- 两个操作符都是AND，不必加括号
h<=i AND j OR k ;               -- 两个操作符不同，未加括号，表达错误
a<=b AND e ;                 -- 操作数b 与 e的位矢长度不一致，表达错误
h<=i OR l ;                -- i 的数据类型是位STD_LOGIC，而l的数据类型是
…                -- 布尔量BOOLEAN，因而不能相互作用，表达错误。
```

KX 康心科技

9.9 操作符

9.9.2 关系操作符

- “ = ” (等于)
- “/= ” (不等于)
- “>” (大于)
- “< ” (小于)
- “>=” (大于等于)
- “<=” (小于等于)

9.9 操作符

9.9.2 关系操作符

【例9-22】

```
ENTITY relational_ops_1 IS
  PORT ( a, b : IN BIT__VECTOR (0 TO 3) ;
               m : OUT BOOLEAN) ;
END relational_ops_1 ;
ARCHITECTURE example OF relational_ops_1 IS
BEGIN
  output <= (a = b) ;
END example ;
```

9.9 操作符

9.9.2 关系操作符

【例9-23】

```
ENTITY relational_ops_2 IS
  PORT (a, b : IN INTEGER RANGE 0 TO 3 ;
            m : OUT BOOLEAN) ;
END relational_ops_2 ;
ARCHITECTURE example OF relational_ops_2 IS
BEGIN
  output <= (a >= b) ;
END example ;
```

9.9 操作符

9.9.3 算术操作符

表9-4 算术操作符分类表

	类　别	算术操作符分类
1	求和操作符(Adding operators)	+(加)，－(减)， &(并置)
2	求积操作符(Multiplying operators)	*， / ， MOD ， REM
3	符号操作符(Sign operators)	+(正)， － (负)
4	混合操作符(Miscellaneous operators)	**， ABS
5	移位操作符(Shift operators)	SLL，SRL，SLA，SRA，ROL，ROR

9.9 操作符

9.9.3 算术操作符

1. 求和操作符

【例9-24】

```
VARIABLE a, b , c ,d , e ,f : INTEGER RANGE 0 TO 255 ;
...
a := b + c ;     d := e – f  ;
```

【例9-25】

```
PROCEDURE adding_e (a: IN INTEGER;  b: INOUT INTEGER ) IS
...
b := a + b ;
```

9.9 操作符

9.9.3 算术操作符

1. 求和操作符

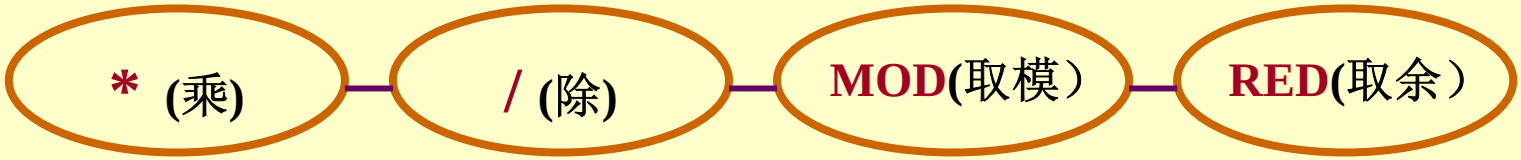
【例9-26】

```
PACKAGE example_arithmetic IS
   TYPE small_INt IS RANGE 0 TO 7 ;
END example_arithmetic ;
USE WORK.example_arithmetic.ALL ;
ENTITY arithmetic IS
   PORT (a, b : IN SMALL_INT ;
              c : OUT SMALL_INT) ;
END arithmetic ;
ARCHITECTURE example OF arithmetic IS
BEGIN
   c <= a + b ;
END example ;
```

9.9 操作符

9.9.3 算术操作符

2. 求积操作符



3. 符号操作符

z := x*(－y) ;

9.9 操作符

9.9.3 算术操作符

4. 混合操作符

【例9-27】

```
SIGNAL a, b : INTEGER RANGE －8 to 7 ;
SIGNAL    c : INTEGER RANGE 0 to 15 ;
SIGNAL    d : INTEGER RANGE 0 to 3 ;
a <= ABS(b) ;
c <= 2 ** d;
```

9.9 操作符

5. 移位操作符

【例9-28】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY decoder3to8 IS
    port (   input: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
             output: OUT BIT_VECTOR (7 DOWNTO 0));
END decoder3to8;
ARCHITECTURE behave OF decoder3to8 IS
BEGIN
output <=  "00000001" SLL CONV_INTEGER(input);    --被移位部分是常数!
END behave;
```

习 题

9-1. 说明实体，设计实体概念。

9-2. 举例说明**GENERIC**说明语句和**GENERIC**映射语句有何用处，并举例说明。

9-3. 说明端口模式**INOUT**和**BUFFER**有何异同点。

9-4. 什么是重载？重栽函数有何用处？

9-5. 在以下数据类型中，**VHDL**综合器支持哪些类型：

STRING、TIME、REAL、BIT

9-6. 详细说明例**10-28**中的语句作用和程序实现的功能。

9-7. 表式**C <= A + B**中，**A、B**和**C**的数据类型都是**STD_LOGIC_VECTOR**，是否能直接进行加法运算？说明原因和解决方法。

9-8. **VHDL**中有哪**3**种数据对象？详细说明它们的功能特点以及使用方法，举例说明数据对象与数据类型的关系。

习 题

9-9. 能把任意一种进制的值向一整数类型的数据对象赋值吗？如果能，怎样做？

9-10. 判断下列VHDL标识符是否合法，如果有误则指出原因：

`16#0FA#， 10#12F#， 8#789#， 8#356#， 2#0101010#`

`74HC245 ， \74HC574\， CLR/RESET， \IN 4/SCLK\， D100%`

9-11. 数据类型BIT、INTEGER和BOOLEAN分别定义在哪个库中？哪些库和程序包总是可见的？

9-12. 函数与过程的设计与功能上有什么区别？调用上有什么区别？

9-13. 回答有关Bit和Boolean数据类型的问题：

(1) 解释Bit和Boolean类型的区别；

(2) 对于逻辑操作应使用哪种类型？

(3) 关系操作的结果为哪种类型？

(4) IF语句测试的表达式是哪种类型？

习 题

9-14. 运算符重载函数通常要调用转换函数，以便能够利用已有的数据类型。下面给出一个新的数据类型AGE，并且下面的转换函数已经实现：

```
function CONV_INTEGER(ARG: AGE) return INTEGER;
```

仿照本章中的示例，利用此函数编写一个“+”运算符重载函数，支持下面的运算：

```
SIGNAL  a，c : AGE;

...

c <= a + 20;
```

9-15. 用两种方法设计8位比较器，比较器的输入是两个待比较的8位数A=[A7..A0]和B=[B7..B0]，输出是 D、E、F。当A=B时D=1；当A>B时E=1；当A<B时F=1。第一种设计方案是常规的比较器设计方法，即直接利用关系操作符进行编程设计；第二种设计方案是利用减法器来完成，通过减法运算后的符号和结果来判别两个被比较值的大小。对两种设计方案的资源耗用情况进行比较，并给以解释。

实 验 与 设 计

9-1 乐曲硬件演奏电路设计

(1) 实验目的：学习利用实验6-3的数控分频器设计硬件乐曲演奏电路。

(2) 实验原理：主系统由3个模块组成，例9-29是顶层设计文件，其内部有3个功能模块(如图9-3所示)：TONETABA.VHD、NOTETABS.VHD和SPEAKER.VHD。

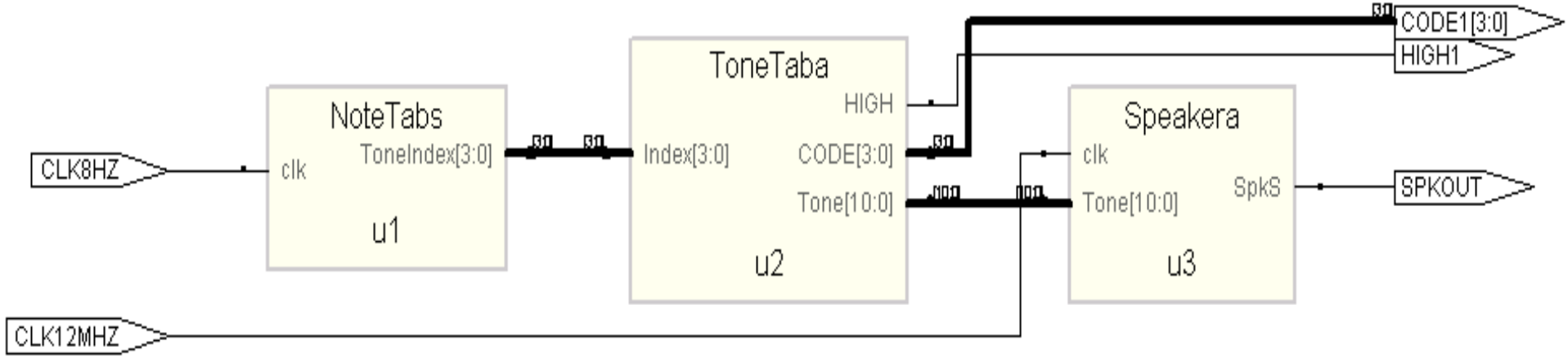


图9-3 硬件乐曲演奏电路结构（Synplify综合）

实 验 与 设 计

9-1 乐曲硬件演奏电路设计

(3) 实验内容1：定制例9-32的NoteTabs模块中的音符数据ROM“music”。该ROM中的音符数据已列在例9-33中。注意该例数据表中的数据位宽、深度和数据的表达类型。此外，为了节省篇幅，例中的数据都横排了，实用中应该以每一分号为一行来展开，否则会出错。

最后对该ROM进行仿真，确认例9-33中的音符数据已经进入ROM中。

(4) 实验内容2：根据给出的乘法器逻辑原理图及其各模块的VHDL描述，在QuartusII上完成全部设计，包括编辑、编译、综合和仿真操作等。给出仿真波形，并作出详细说明。

(5) 实验内容3：硬件验证。先将引脚锁定，使CLK12MHz与clock9相接，接受12MHz时钟频率（用短路帽在clock9接“12MHz”）；CLK8Hz 与clock2相接，接受4Hz频率；发音输出SPKOUT接Speaker；与演奏发音相对应的简谱码输出显示可由CODE1在数码管5显示；HIGH1为高八度音指示，可由发光管D5指示，最后向目标芯片下载适配后的SOF逻辑设计文件。实验电路结构图为NO.1。

实 验 与 设 计

(6) 实验内容4： 填入新的乐曲，如“采茶舞曲”、或其它熟悉的乐曲。操作步骤如下：

1、根据所填乐曲可能出现的音符，修改例9-3的音符数据表格，同时注意每一音符的节拍长短；

2、如果乐曲比较长，可增加模块NOTETABA中计数器的位数，如9位时可达512个基本节拍。

(7) 实验内容5： 争取可以在一个ROM装上多首歌曲，可手动或自动选择歌曲。

(8) 实验内容6： 根据此项实验设计一个电子琴，硬件测试可用电路结构图NO.3。

(9) 思考题1： 用LFSR设计可编程分频器，对本实验中的音阶发生电路的可编程计数器(实现可编程分频功能)用LFSR替代。

(10) 思考题2： 例9-30中的进程DelaySpkS对扬声器发声有什么影响？

(11) 思考题3： 在电路上应该满足哪些条件，才能用数字器件直接输出的方波驱动扬声器发声？

(12) 实验报告： 用仿真波形和电路原理图，详细叙述硬件电子琴的工作原理及其4个VHDL文件中相关语句的功能，叙述硬件实验情况。

【例9-29】

```
LIBRARY IEEE; -- 硬件演奏电路顶层设计
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Songer IS
    PORT (   CLK12MHZ : IN STD_LOGIC;                    --音调频率信号
             CLK8HZ   : IN STD_LOGIC;                    --节拍频率信号
               CODE1  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);-- 简
谱码输出显示
               HIGH1  : OUT STD_LOGIC; --高8度指示
             SPKOUT   : OUT STD_LOGIC );--声音输出
 END;
ARCHITECTURE one OF Songer IS
   COMPONENT NoteTabs
     PORT ( clk     : IN STD_LOGIC;
         ToneIndex : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
    END COMPONENT;
    COMPONENT ToneTaba
        PORT ( Index :  IN  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
               CODE  : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
               HIGH  : OUT STD_LOGIC;
               Tone  : OUT  STD_LOGIC_VECTOR (10 DOWNTO 0) );
    END COMPONENT;
```

（接下页）

```
  COMPONENT Speakera
          PORT  ( clk  : IN STD_LOGIC;
                  Tone : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
                  SpkS : OUT STD_LOGIC  );
      END COMPONENT;
      SIGNAL Tone : STD_LOGIC_VECTOR (10 DOWNTO 0);
      SIGNAL ToneIndex : STD_LOGIC_VECTOR (3 DOWNTO 0);
   BEGIN
  u1 : NoteTabs  PORT MAP (clk=>CLK8HZ, ToneIndex=>ToneIndex);
  u2 : ToneTaba PORT MAP
(Index=>ToneIndex,Tone=>Tone,CODE=>CODE1,HIGH=>HIGH1);
  u3 : Speakera PORT MAP(clk=>CLK12MHZ,Tone=>Tone, SpkS=>SPKOUT
);
  END;
```

【例9-30】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY Speakera IS
    PORT (   clk  : IN STD_LOGIC;
             Tone : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
             SpkS : OUT STD_LOGIC  );
END;
ARCHITECTURE one OF Speakera IS
    SIGNAL PreCLK, FullSpkS : STD_LOGIC;
BEGIN
 DivideCLK : PROCESS(clk)
        VARIABLE Count4 : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
    BEGIN
        PreCLK <= '0';  --  将CLK进行16分频，PreCLK为CLK的16分频
        IF Count4>11 THEN PreCLK <= '1';  Count4 := "0000";
        ELSIF clk'EVENT AND clk = '1' THEN  Count4 := Count4 + 1;
        END IF;
    END PROCESS;
    GenSpkS : PROCESS(PreCLK, Tone)-- 11位可预置计数器
        VARIABLE Count11 : STD_LOGIC_VECTOR (10 DOWNTO 0);
BEGIN
      IF PreCLK'EVENT AND PreCLK = '1' THEN
       IF Count11 = 16#7FF# THEN Count11 := Tone ; FullSpkS <= '1';
```

（接下页）

```
  ELSE Count11 := Count11 + 1; FullSpkS <= '0'; END IF;
          END IF;
     END PROCESS;
   DelaySpkS : PROCESS(FullSpkS)--将输出再2分频，展宽脉冲，使扬声器有足够功率发
音
          VARIABLE Count2 : STD_LOGIC;
  BEGIN
      IF FullSpkS'EVENT AND FullSpkS = '1' THEN  Count2 := NOT Count2;
              IF Count2 = '1' THEN  SpkS <= '1';
              ELSE SpkS <= '0';  END IF;
          END IF;
      END PROCESS;
  END;
```

【例9-31】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ToneTaba IS
    PORT ( Index :  IN  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
           CODE  : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0) ;
           HIGH  : OUT STD_LOGIC;
           Tone  : OUT  STD_LOGIC_VECTOR (10 DOWNTO 0) );
END;
ARCHITECTURE one OF ToneTaba IS
BEGIN
    Search : PROCESS(Index)
    BEGIN
        CASE Index IS     -- 译码电路，查表方式，控制音调的预置数
  WHEN "0000" => Tone<="11111111111" ; CODE<="0000"; HIGH <='0';-- 2047
  WHEN "0001" => Tone<="01100000101" ; CODE<="0001"; HIGH <='0';-- 773;
  WHEN "0010" => Tone<="01110010000" ; CODE<="0010"; HIGH <='0';-- 912;
  WHEN "0011" => Tone<="10000001100" ; CODE<="0011"; HIGH <='0';--1036;
  WHEN "0101" => Tone<="10010101101" ; CODE<="0101"; HIGH <='0';--1197;
  WHEN "0110" => Tone<="10100001010" ; CODE<="0110"; HIGH <='0';--1290;
  WHEN "0111" => Tone<="10101011100" ; CODE<="0111"; HIGH <='0';--1372;
  WHEN "1000" => Tone<="10110000010" ; CODE<="0001"; HIGH <='1';--1410;
  WHEN "1001" => Tone<="10111001000" ; CODE<="0010"; HIGH <='1';--1480;
  WHEN "1010" => Tone<="11000000110" ; CODE<="0011"; HIGH <='1';--1542;
  WHEN "1100" => Tone<="11001010110" ; CODE<="0101"; HIGH <='1';--1622;
  WHEN "1101" => Tone<="11010000100" ; CODE<="0110"; HIGH <='1';--1668;
  WHEN "1111" => Tone<="11011000000" ; CODE<="0001"; HIGH <='1';--1728;
  WHEN OTHERS => NULL;
  END CASE;
  END PROCESS;
END;
```

【例9-32】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY NoteTabs IS
    PORT ( clk    : IN STD_LOGIC;
        ToneIndex : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
END;
ARCHITECTURE one OF NoteTabs IS
COMPONENT MUSIC                    --音符数据ROM
 PORT(address : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      inclock : IN STD_LOGIC ;
            q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END COMPONENT;
    SIGNAL Counter :  STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    CNT8 : PROCESS(clk, Counter)
    BEGIN
        IF Counter=138 THEN  Counter <= "00000000";
        ELSIF  (clk'EVENT  AND  clk  =  '1')  THEN  Counter  <=
Counter+1; END IF;
    END PROCESS;
  u1   :   MUSIC   PORT   MAP(address=>Counter   ,   q=>ToneIndex,
inclock=>clk);
END;
```

【例9-33】

```
WIDTH = 4 ; --“梁祝”乐曲演奏数据
DEPTH = 256 ;
ADDRESS_RADIX = DEC ;
DATA_RADIX = DEC ;
CONTENT  BEGIN
 00: 3 ; 01: 3 ; 02: 3 ; 03: 3; 04: 5; 05: 5; 06:  5;07: 6; 08: 8; 09: 8;
 10: 8 ; 11: 9 ; 12: 6 ; 13: 8; 14: 5; 15: 5; 16: 12;17: 12;18: 12; 19:15;
 20:13 ; 21:12 ; 22:10 ; 23:12; 24: 9; 25: 9; 26: 9; 27: 9; 28: 9; 29: 9;
 30: 9 ; 31: 0 ; 32: 9 ; 33: 9; 34: 9; 35:10; 36: 7; 37: 7; 38: 6; 39: 6;
 40: 5 ; 41: 5 ; 42: 5 ; 43: 6; 44: 8; 45: 8; 46: 9; 47: 9; 48: 3; 49: 3;
 50: 8 ; 51: 8 ; 52: 6 ; 53: 5; 54: 6; 55: 8; 56: 5; 57: 5; 58: 5; 59: 5;
 60: 5 ; 61: 5 ; 62: 5 ; 63: 5; 64:10; 65:10; 66:10; 67:12; 68: 7; 69: 7;
 70: 9 ; 71: 9 ; 72: 6 ; 73: 8; 74: 5; 75: 5; 76: 5; 77: 5; 78: 5; 79: 5;
 80: 3 ; 81: 5 ; 82: 3 ; 83: 3; 84: 5; 85: 6; 86: 7; 87: 9; 88: 6; 89: 6;
 90: 6 ; 91: 6 ; 92: 6 ; 93: 6; 94: 5; 95: 6; 96: 8; 97: 8; 98: 8; 99: 9;
100:12 ;101:12 ;102:12 ;103:10;104: 9;105: 9;106:10;107: 9;108: 8;109: 8;
110: 6 ;111: 5 ;112: 3 ;113: 3;114: 3;115: 3;116: 8;117: 8;118: 8;119: 8;
120: 6 ;121: 8 ;122: 6 ;123: 5;124: 3;125: 5;126: 6;127: 8;128: 5;129: 5;
130: 5 ;131: 5 ;132: 5 ;133: 5;134: 5;135: 5;136: 0;137: 0;138: 0;
END ;
```

实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(1) 实验目的：学习利用FPGA控制高速ADC、示波器显示控制方法等。

(2) 实验原理：图9-4所示的是基于大学生电子设计竞赛赛题的存储示波器结构图，FPGA中的A/D采样控制器负责对A/D对模拟信号的采样控制，并将A/D转换好的数据送到FPGA的内部RAM中存储；RAM的地址信号由地址发生计数器产生。当完成1至数个周期的被测信号的采样后，在地址发生计数器的地址扫描下，将存于RAM中的数据通过外部的D/A进入示波器的Y端；与此同时，地址发生计数器的地址信号分频后通过另一个D/A构成锯齿波信号，进入示波器的X端。从而实现存储示波器的功能。

实 验 与 设 计

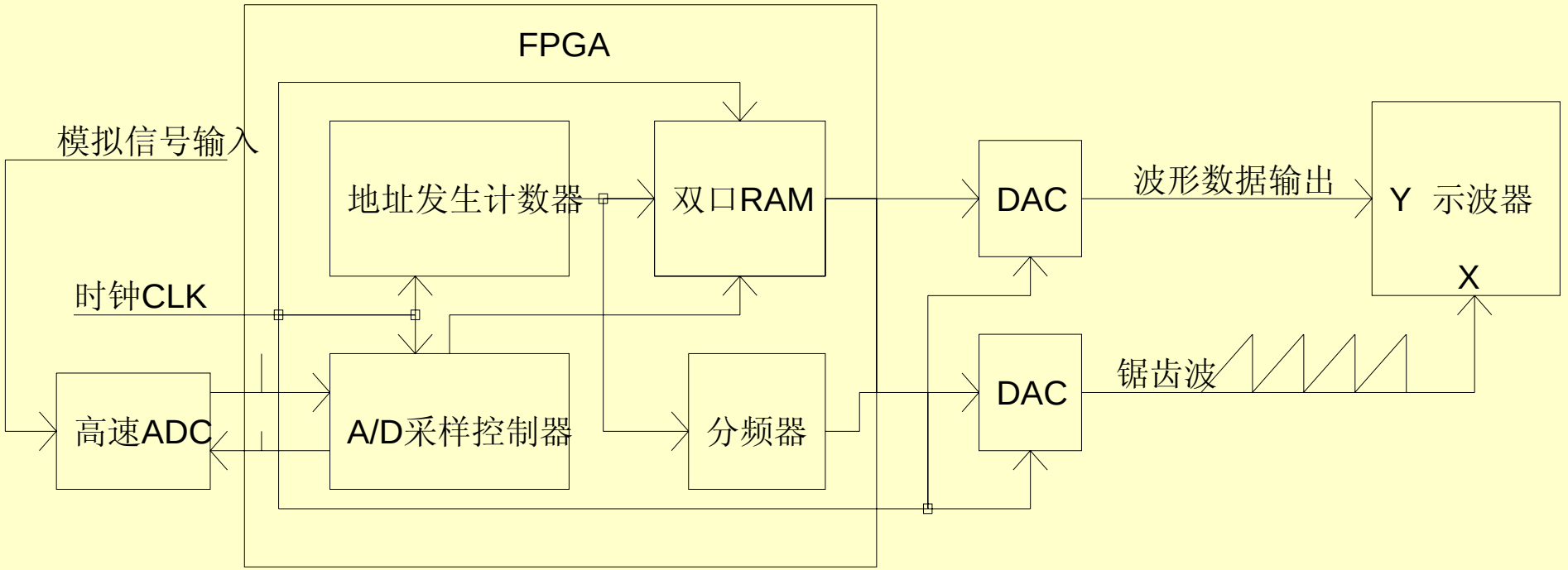


图9-4 存储示波器结构简图

实 验 与 设 计

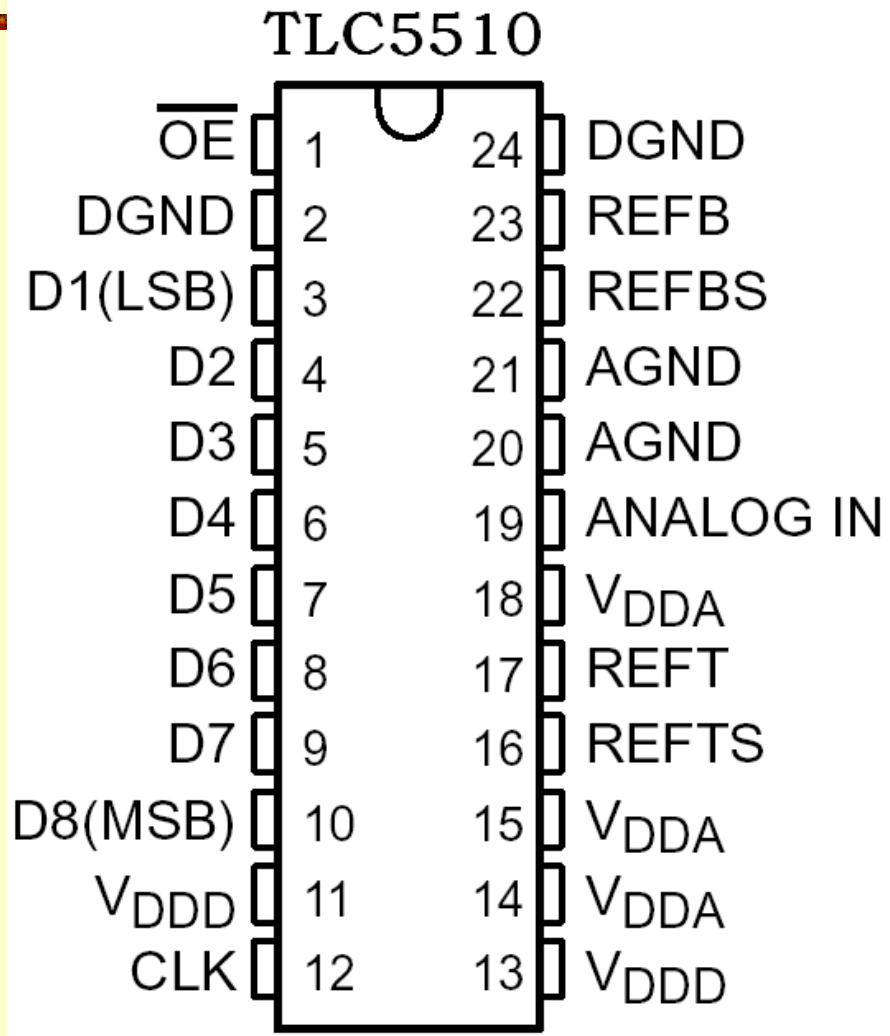


图9-5 TLC5510引脚图

实 验 与 设 计

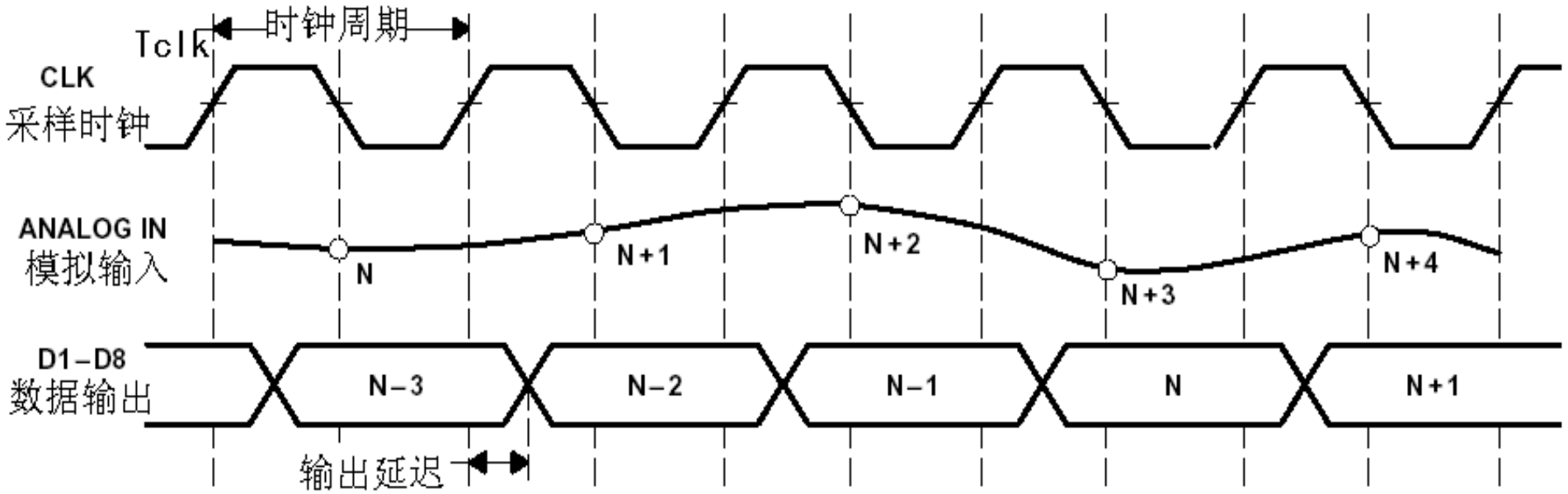


图9-6 TLC5510采样时序图

实 验 与 设 计

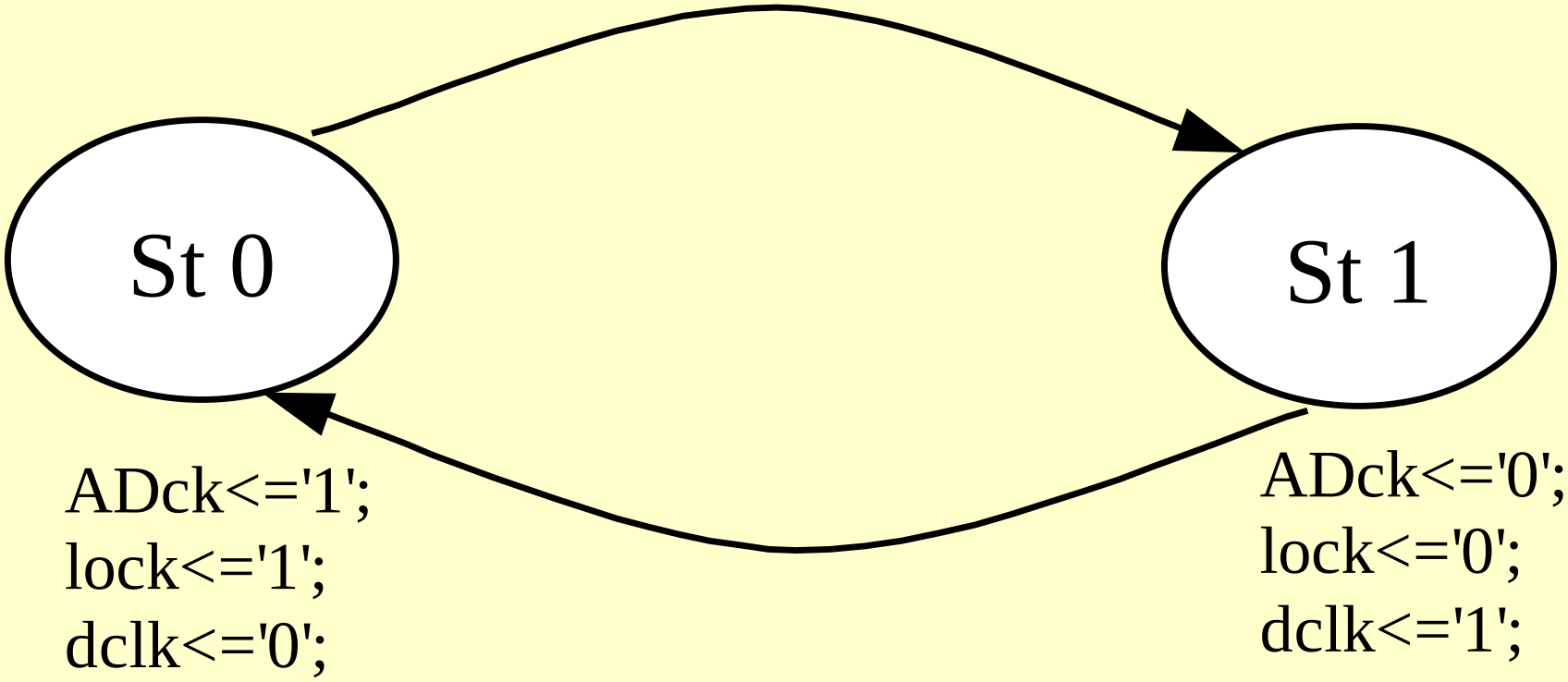


图9-7 TLC5510采样控制状态图

实 验 与 设 计

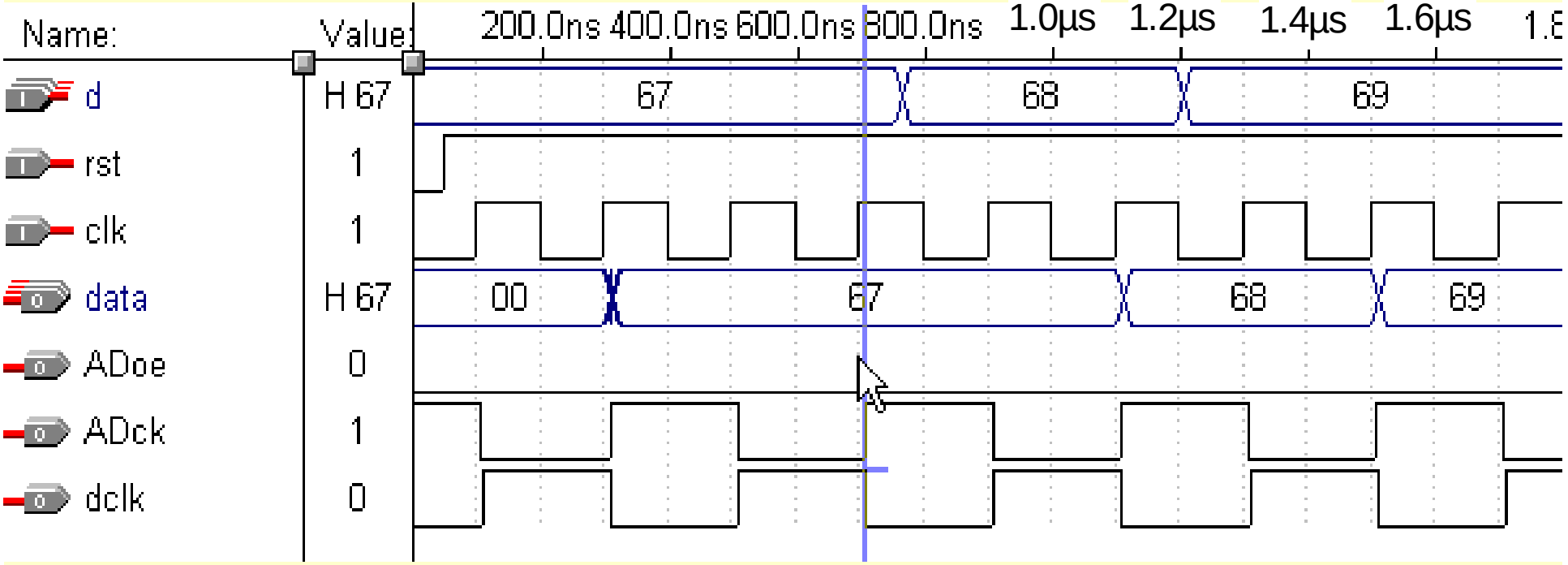


图9-8 A/D转换仿真波形

【例9-34】-- TLC5510 采样控制示例

```
library IEEE;
use     IEEE.STD_LOGIC_1164.ALL;
entity ad5510 is
     port( rst, clk : in  std_logic; --rst 复位;clk采样时钟输入
             d    : in    std_logic_vector(7 downto 0);-- 8位A/D数据
     Adck，ADoe  : out std_logic;-- Adck，Adoe分别为TLC5510的CLK和的OE
          data   : out    std_logic_vector(7 downto 0);-- 8位数据
          dclk   : out    std_logic  );              -- 数据输出锁存信号
end ad5510;
architecture ADCTRL of ad5510 is
        type adsstates is (sta0,sta1);              --定义两个状态变量
        signal    ads_state,next_ads_state : adsstates;
        signal    lock    : std_logic;
begin
ads : PROCESS( ads_state)   -- A/D 采样控制状态机
BEGIN
       CASE ads_state IS
  WHEN sta0 => ADck<='1'; lock<='1'; dclk<='0';next_ads_state <= sta1;
  WHEN sta1 => ADck<='0'; lock<='0'; dclk<='1';next_ads_state <= sta0;
  WHEN OTHERS => ADck<='0'; lock<='0'; dclk<='1';next_ads_state <= sta0;
    END CASE ;
END PROCESS;
PROCESS (CLK,rst)
```

（接下页 ）

```
BEGIN
        IF RST ='0' THEN   ads_state <= sta0;
          ELSIF ( CLK'EVENT AND CLK='1')  THEN
             ads_state <= next_ads_state;  -- 在时钟上升沿，转换至下一状态
       END IF;
END PROCESS;
PROCESS (lock,rst) -- 此进程中，在lock的上升沿，将转换好的数据锁入
       BEGIN
        IF RST ='0' THEN   data <= (others => '0');
          ELSIF lock'EVENT AND lock='1' THEN   data <= D ;   END IF;
END PROCESS ;
ADoe <= '0';
end ADCTRL;
```

【例9-35】 -- TLC5510的另一种采样控制方法

```
...
lock <= clk;  ADck <= clk; dclk <= not lock; ADoe <= '0';
PROCESS (lock,rst) -- 此进程中，在lock的上升沿，将转换好的数据锁入
BEGIN
       if rst <= '0' then  data <= (others => '0');
       ELSIF lock'EVENT AND lock='1' THEN  data <= D ; END IF;
END PROCESS ;
end logi;
```

【例9-36】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY RESERV IS
      PORT(CLK, KEY1 : IN  STD_LOGIC;--时钟（20MHz）和采样/显示控制键
                TRAG : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);--扫描锯齿波输出
           DOUT : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);--波形输出
            ADIN   : IN  STD_LOGIC_VECTOR (7 DOWNTO 0)  ); --A/D采样数据输入
END;
ARCHITECTURE DACC OF RESERV IS
COMPONENT DPRAM   --采样数据存储用RAM
        PORT ( address : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
          inclock, wren: IN STD_LOGIC ;
                  data  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
                        q   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;
      SIGNAL    Q1   : STD_LOGIC_VECTOR (9 DOWNTO 0);
      SIGNAL MD0,DIN : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
PROCESS(CLK)
      BEGIN
        IF rising_edge(CLK) THEN  Q1 <= Q1 + 1;  END IF; --生成锯齿波计数器
END PROCESS;
process(CLK, ADIN)
begin
     if rising_edge(CLK) then DIN <= ADIN ; end if;  --采样数据锁存进RAM
end process;
    DOUT(9 DOWNTO 2)<=MD0; TRAG<=Q1; DOUT(1 DOWNTO 0) <= "00";
  u1 : DPRAM PORT MAP(data=>DIN, wren=>KEY1, address=>Q1, q=>MD0, inclock=>CLK);
END;
```

实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(3) 实验内容1：对例9-36给出仿真波形，并分析；引脚锁定后，进行硬件验证（包括定制LPM_RAM）。时钟首先输入13MHz。注意打开系统的+/-13V电源开关。用示波器的Y1（X）端接GWADDA板的5651 D/A输出；GWADDA板上5510 A/D口的“AIN”接受来自主系统模拟波形，即接主系统板上右侧“JP17”的“OUTPUT”端，然后将主系统板上“JP18”的“INPUT”端与系统右下角的时钟65536或32768HZ等相接。“JL11”的3针座短路“H_F”端，调节“JP15”电位器，使得主系统右侧的“WAVE OUT”端输出正常信号波形（用示波器监视，在4V上下）。GWADDA板上跳线JAD的短路帽插“20MHz”端，这时VHDL程序时钟CLK锁定在137脚（EP1C3）；此时AD5510和DA5651的工作时钟与FPGA的工作时钟完全一样，都是20MHz。电路图选择No.5，选择键1高电平为采样，低电平为存储显示。

波形输出：Y1（X）端接GWADDA板的5651的A输出端，作X作锯齿波输出；

波形输出：Y2（Y）端接GWADDA板的5651的B输出端，作Y作波形输出。

实 验 与 设 计

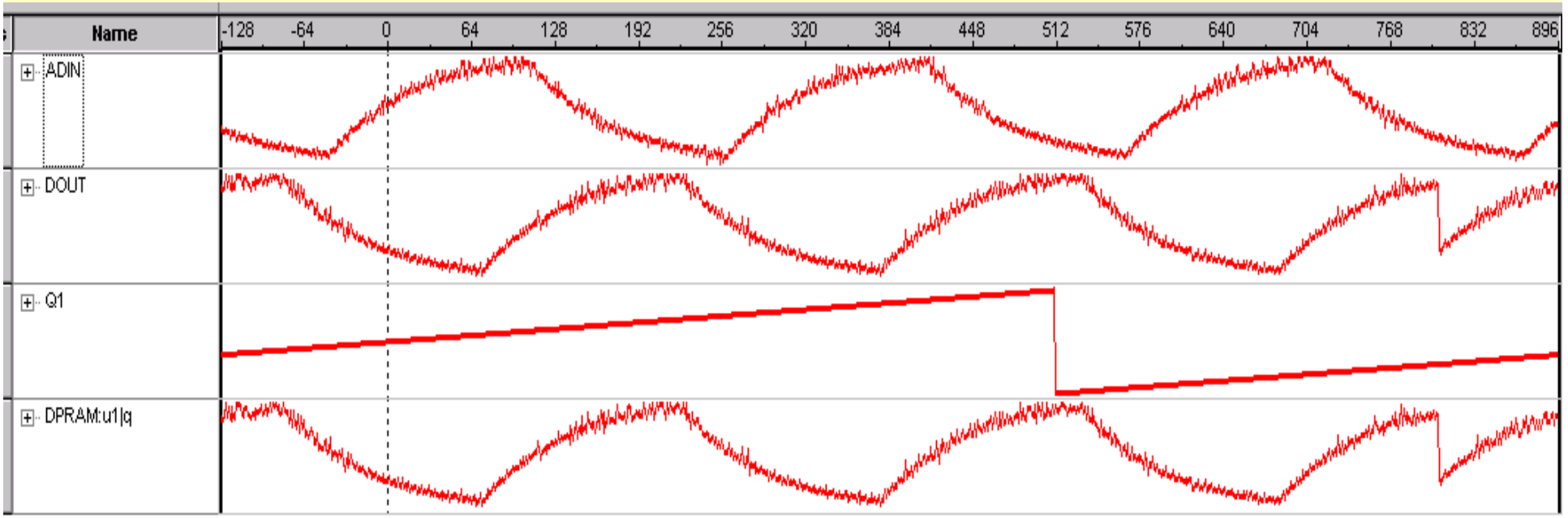


图9-9 存储示波器采样电路系统内SignalTapII数据波形

实 验 与 设 计

9-2采用高速A/D的存储示波器设计

(4) 实验内容2： 图9-9所示的是此存储示波器采样电路系统内SignalTapII给出的数据波形，其中ADIN是直接来自5510的采样波形；Q1是RAM的地址波形；q是DPRAM输出的存储波形，显然此波形有缺陷，原因是每一次采如RAM中的波形数据都不是整数倍的波形。为了改善输出波形的显示效果，设计一个状态机，严格控制进入RAM中的数据在地址发生器一个扫描周期中含被采样信号整数个完整周期的采样数据。

实 验 与 设 计

9-3 循环冗余校验(CRC)模块设计

(1) 实验目的：设计一个在数字传输中常用的校验、纠错模块：循环冗余校验CRC模块，学习使用FPGA器件完成数据传输中的差错控制。

(2) 实验原理：CRC即Cyclic Redundancy Check 循环冗余校验，是一种数字通信中的信道编码技术。经过CRC方式编码的串行发送序列码，可称为CRC码，共由两部分构成：k位有效信息数据和r位CRC校验码。其中r位CRC校验码是通过k位有效信息序列被一个事先选择的r+1位“生成多项式”相“除”后得到的(r位余数即是CRC校验码)，这里的除法是“模2运算”。CRC校验码一般在有效信息发送时产生，拼接在有效信息后被发送；在接收端，CRC码用同样的生成多项式相除，除尽表示无误，弃掉r位CRC校验码，接收有效信息；反之，则表示传输出错，纠错或请求重发。

实 验 与 设 计

sdata：12位的待发送信息；　　**datald**：sdata的装载信号；

error：误码警告信号　　；　　**datafini**：数据接收校验完成；

rdata：接收模块(检错模块)接收的12位有效信息数据；　　**clk**：时钟信号；

datacrc：附加上5位CRC校验码的17位CRC码，在生成模块被发送，在接收模块被接收；

hsend、hrecv：生成、检错模块的握手信号，协调相互之间关系；

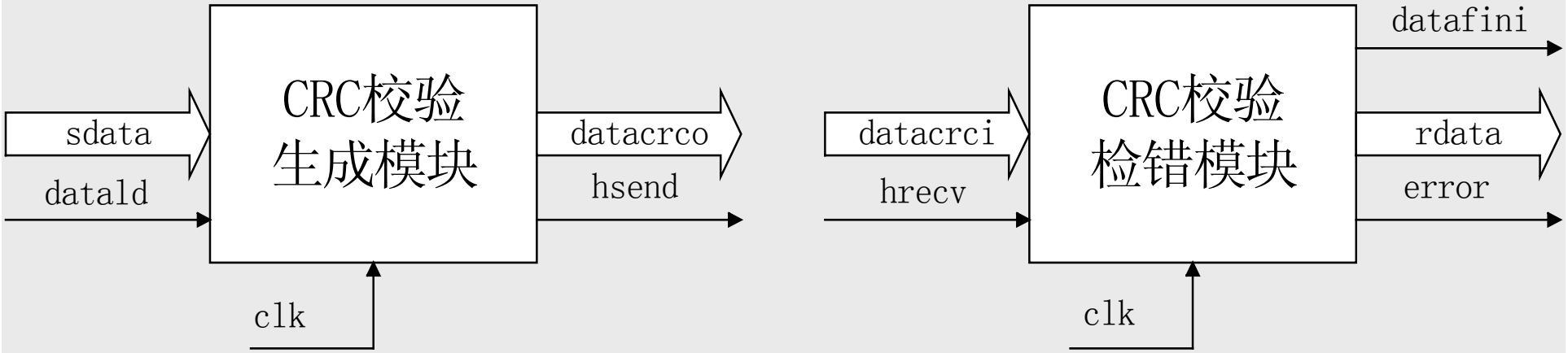


图9-10 CRC模块

【例9-37】

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;
ENTITY crcm IS
    PORT (clk, hrecv，datald : IN std_logic;
          sdata     : IN std_logic_vector(11 DOWNTO 0);
          datacrco  : OUT std_logic_vector(16 DOWNTO 0);
          datacrci  : IN std_logic_vector(16 DOWNTO 0);
            rdata   : OUT std_logic_vector(11 DOWNTO 0);
          datafini  : OUT std_logic;
          ERROR0, hsend  : OUT std_logic);
END crcm;
ARCHITECTURE comm OF crcm IS
    CONSTANT multi_coef : std_logic_vector(5 DOWNTO 0) := "110101";
            -- 多项式系数, MSB一定为'1'
    SIGNAL  cnt,rcnt : std_logic_vector(4 DOWNTO 0);
    SIGNAL  dtemp,sdatam,rdtemp   : std_logic_vector(11 DOWNTO 0);
    SIGNAL  rdatacrc: std_logic_vector(16 DOWNTO 0);
    SIGNAL  st,rt  : std_logic;
BEGIN
PROCESS(clk)
    VARIABLE crcvar : std_logic_vector(5 DOWNTO 0);
BEGIN
    IF(clk'event AND clk = '1') THEN
    IF(st = '0' AND datald = '1') THEN  dtemp <= sdata;
```

（接下页）

```
sdatam <= sdata; cnt <= (OTHERS => '0');  hsend <= '0';  st <= '1';
        ELSIF(st = '1' AND cnt < 7) THEN  cnt <= cnt + 1;
        IF(dtemp(11) = '1') THEN  crcvar := dtemp(11 DOWNTO 6) XOR multi_coef;
           dtemp <= crcvar(4 DOWNTO 0) & dtemp(5 DOWNTO 0) & '0';
            ELSE  dtemp <= dtemp(10 DOWNTO 0) & '0';   END IF;
     ELSIF(st='1' AND cnt=7) THEN datacrco<=sdatam & dtemp(11 DOWNTO 7);
            hsend <= '1';  cnt <= cnt + 1;
        ELSIF(st='1' AND cnt=8) THEN   hsend<= '0';    st<='0';
        END IF;
    END IF;
END PROCESS;
PROCESS(hrecv,clk)
    VARIABLE rcrcvar : std_logic_vector(5 DOWNTO 0);
BEGIN
 IF(clk'event AND clk = '1') THEN
  IF(rt = '0' AND hrecv = '1') THEN  rdtemp <= datacrci(16 DOWNTO 5);
   rdatacrc <= datacrci;  rcnt <= (OTHERS => '0');
   ERROR0 <= '0';    rt <= '1';
   ELSIF(rt= '1' AND rcnt < 7) THEN  datafini <= '0';  rcnt <= rcnt + 1;
            rcrcvar := rdtemp(11 DOWNTO 6) XOR multi_coef;
            IF(rdtemp(11) = '1') THEN
             rdtemp <= rcrcvar(4 DOWNTO 0) & rdtemp(5 DOWNTO 0) & '0';
            ELSE  rdtemp <= rdtemp(10 DOWNTO 0) & '0';
            END IF;
        ELSIF(rt = '1' AND rcnt = 7) THEN  datafini <= '1';
            rdata <= rdatacrc(16 DOWNTO 5);  rt <= '0';
            IF(rdatacrc(4 DOWNTO 0) /= rdtemp(11 DOWNTO 7)) THEN
              ERROR0 <= '1'; END IF;
        END IF;
    END IF;
END PROCESS;
END comm;
```

实 验 与 设 计

9-3 循环冗余校验(CRC)模块设计

(3) 实验内容1：编译以上示例文件，给出仿真波形。

(4) 实验内容2：建立一个新的设计，调入**crcm**模块，把其中的**CRC**校验生成模块和**CRC**校验查错模块连接在一起，协调工作。引出必要的观察信号，锁定引脚，并在**EDA**实验系统上实现之。

(5) 思考题1：例中对**st**、**rt**有不妥之处，试解决之(提示：复位**reset**信号的引入有助于问题的解决)。

(6) 思考题2：如果输入数据、输出**CRC**码都是串行的，设计该如何实现(提示：采用**LFSR**)。

(7) 思考题3：在例子程序中需要**8**个时钟周期才能完成一次**CRC**校验，试重新设计使得在一个**clk**周期内完成。

(8) 实验报告：叙述**CRC**的工作原理，将设计原理、程序设计与分析、仿真分析和详细实验过程。

EDA 技术实用教程

第 8 章

状态机设计

8.1 一般有限状态机设计

8.1.1 数据类型定义语句

TYPE语句的用法如下：

TYPE 数据类型名 IS 数据类型定义 OF 基本数据类型 ;

或

TYPE 数据类型名 IS 数据类型定义;

```
TYPE st1 IS ARRAY ( 0 TO 15 ) OF STD_LOGIC ;
TYPE week IS (sun，mon，tue，wed，thu，fri，sat);
```

8.1 一般有限状态机设计

8.1.1 数据类型定义语句

```
TYPE m_state IS  ( st0，st1，st2，st3，st4，st5 ) ;
SIGNAL present_state，next_state ： m_state ;
```

```
TYPE BOOLEAN  IS (FALSE，TRUE) ;
```

```
TYPE my_logic IS  ( '1' , 'Z' , 'U' , '0' ) ;
SIGNAL s1 : my_logic ;
s1 <=  'Z' ;
```

```
SUBTYPE  子类型名 IS 基本数据类型 RANGE 约束范围;
```

```
SUBTYPE digits IS INTEGER RANGE 0 to 9 ;
```

8.1 一般有限状态机设计

8.1.2 为什么要使用状态机

- 状态机克服了纯硬件数字系统顺序方式控制不灵活的缺点
- 状态机可以定义符号化枚举类型的状态
- 状态机容易构成性能良好的同步时序逻辑模块
- 状态机的VHDL表述丰富多样、程序层次分明，易读易懂
- 在高速运算和控制方面，状态机更有其巨大的优势
- 高可靠性

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

1. 说明部分

```
ARCHITECTURE ...IS
TYPE FSM_ST IS (s0, s1, s2, s3);
SIGNAL current_state, next_state:   FSM_ST;
 …
```

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

2. 主控时序进程

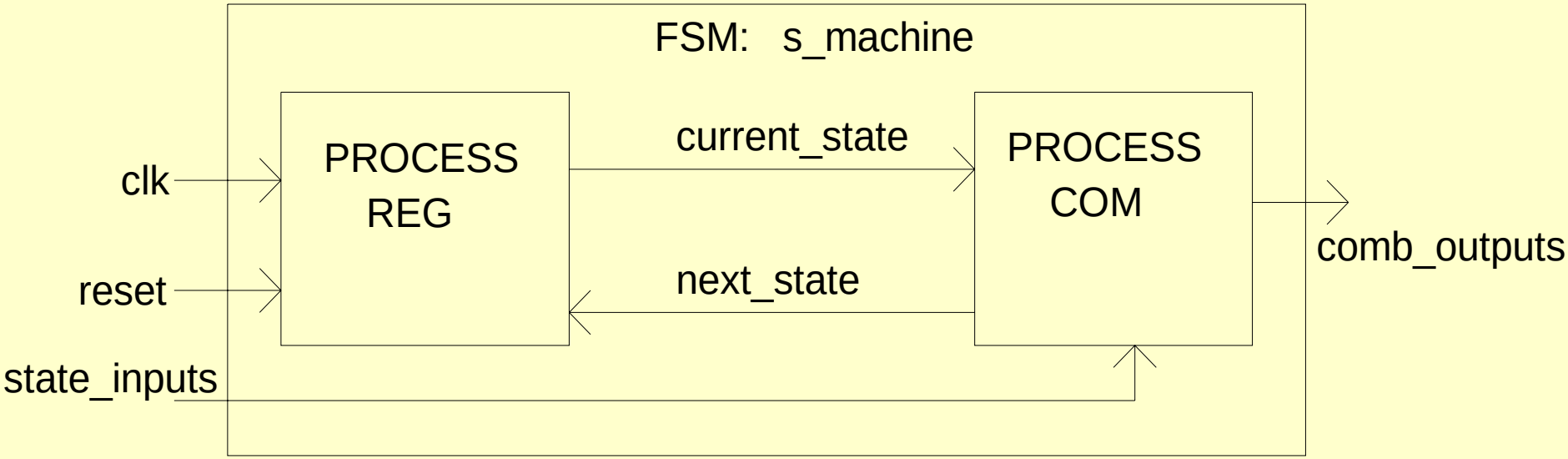


图8-1 一般状态机结构框图

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

3. 主控组合进程

4. 辅助进程

【例8-1】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY s_machine IS
  PORT ( clk,reset    : IN STD_LOGIC;
         state_inputs : IN STD_LOGIC_VECTOR (0 TO 1);
         comb_outputs : OUT INTEGER RANGE 0 TO 15 );
END s_machine;
ARCHITECTURE behv OF s_machine IS
  TYPE FSM_ST IS (s0, s1, s2, s3);  --数据类型定义，状态符号化
  SIGNAL current_state, next_state: FSM_ST;--将现态和次态定义为新的数据类型
BEGIN
 REG: PROCESS (reset,clk)                    --主控时序进程
```

（接下页）

```
BEGIN
    IF reset = '1' THEN   current_state <= s0;--检测异步复位信号
    ELSIF clk='1' AND clk'EVENT THEN
      current_state <= next_state;
    END IF;
  END PROCESS;
COM:PROCESS(current_state, state_Inputs)     --主控组合进程
  BEGIN
    CASE current_state IS
      WHEN s0 => comb_outputs<= 5;
        IF state_inputs = "00" THEN  next_state<=s0;
           ELSE  next_state<=s1;
        END IF;
      WHEN s1 =>  comb_outputs<= 8;
        IF state_inputs = "00" THEN  next_state<=s1;
        ELSE  next_state<=s2;
        END IF;
      WHEN s2 =>   comb_outputs<= 12;
        IF state_inputs = "11" THEN  next_state <= s0;
        ELSE  next_state <= s3;
        END IF;
      WHEN s3 =>  comb_outputs <= 14;
        IF state_inputs = "11" THEN  next_state <= s3;
        ELSE  next_state <= s0;
        END IF;
    END case;
  END PROCESS;
END behv;
```

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

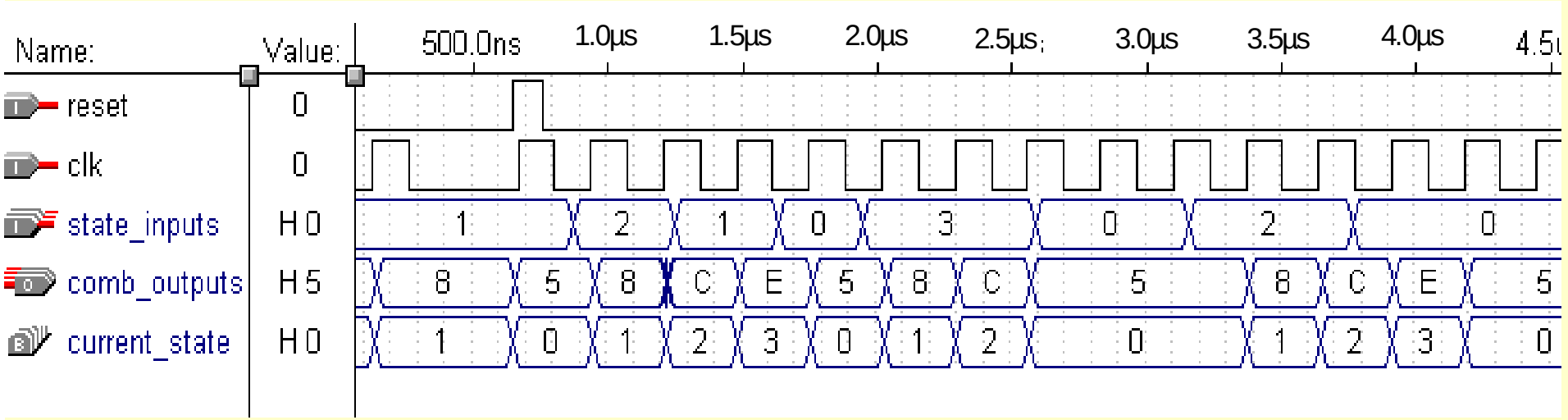


图8-2a 例8-1状态机的工作时序

KX 康芯科技

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

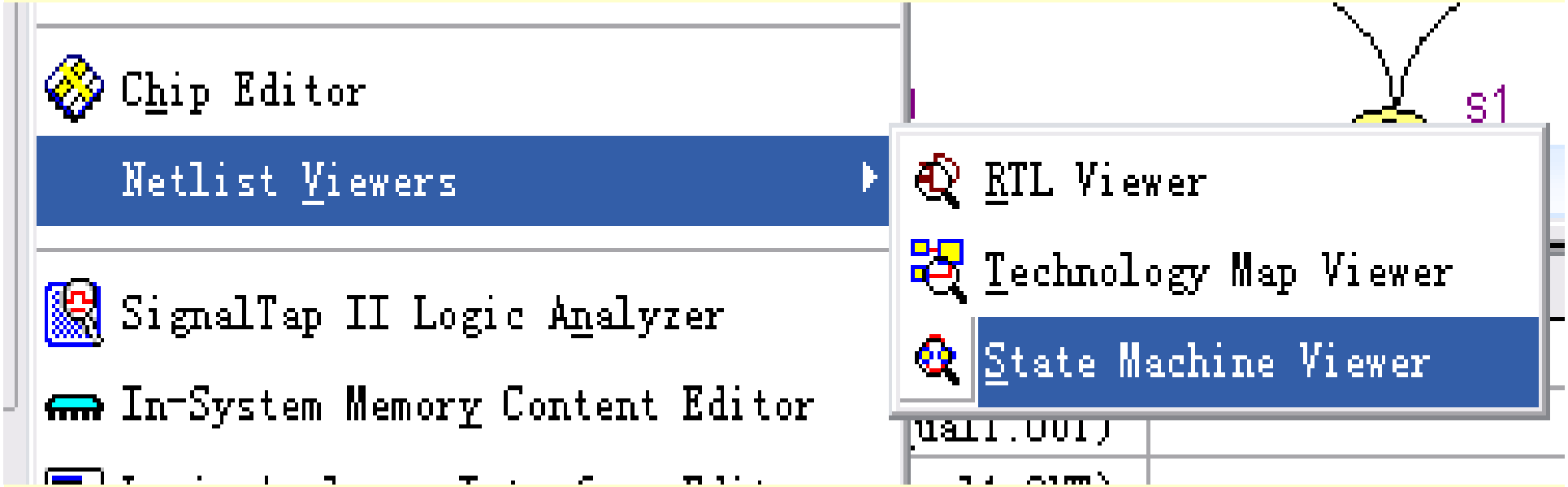


图8-2b 打开QuartusII状态图观察器

8.1 一般有限状态机设计

8.1.3 一般有限状态机的设计

4. 辅助进程

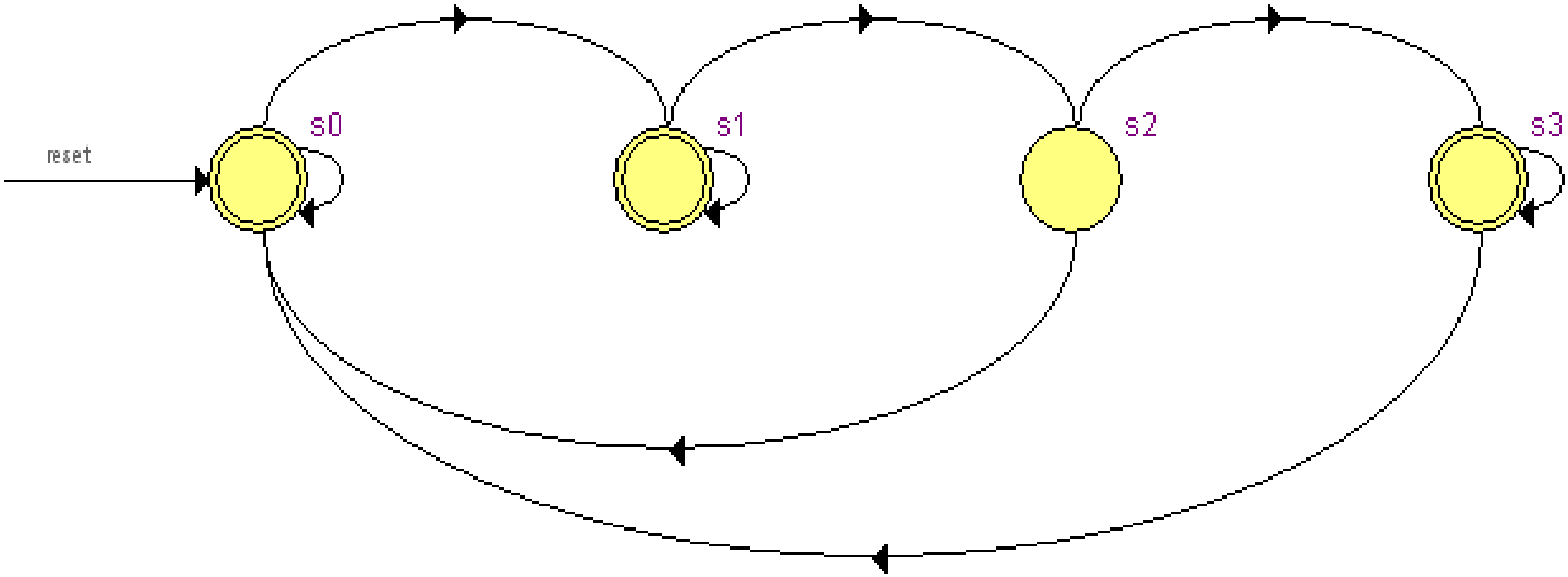


图8-2c 例8-1的状态图

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

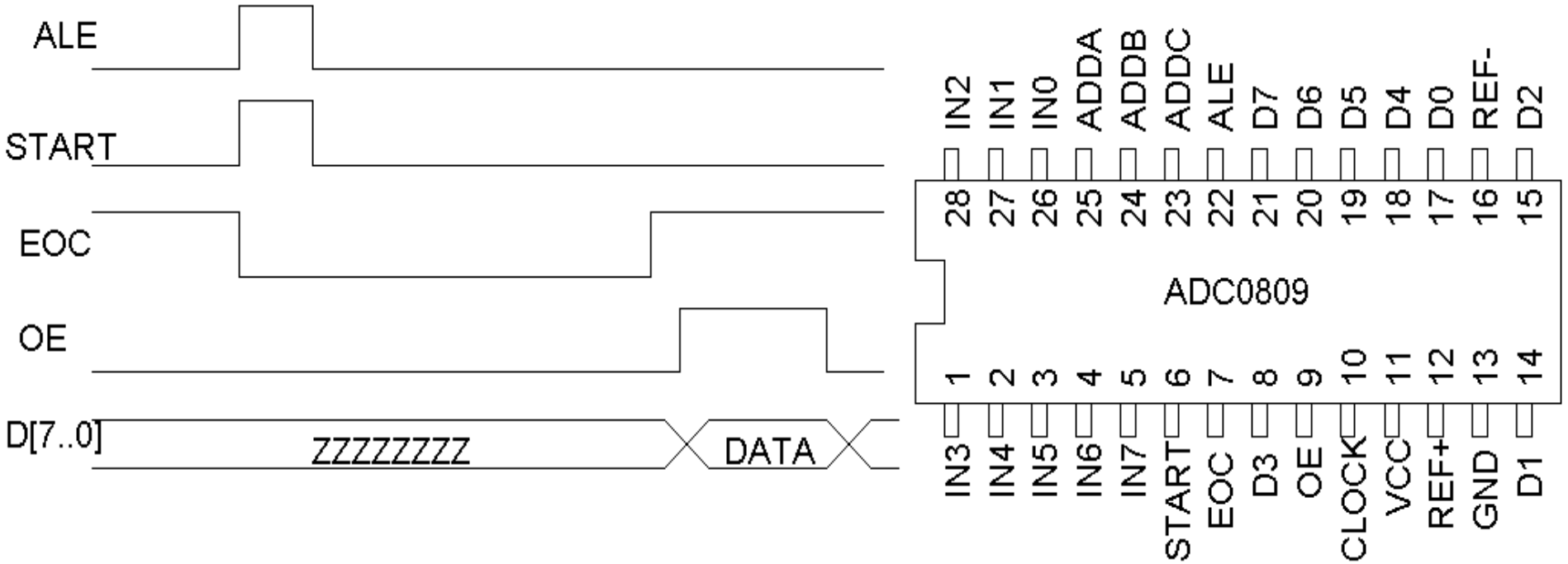


图8-3 ADC0809工作时序

康芯科技

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

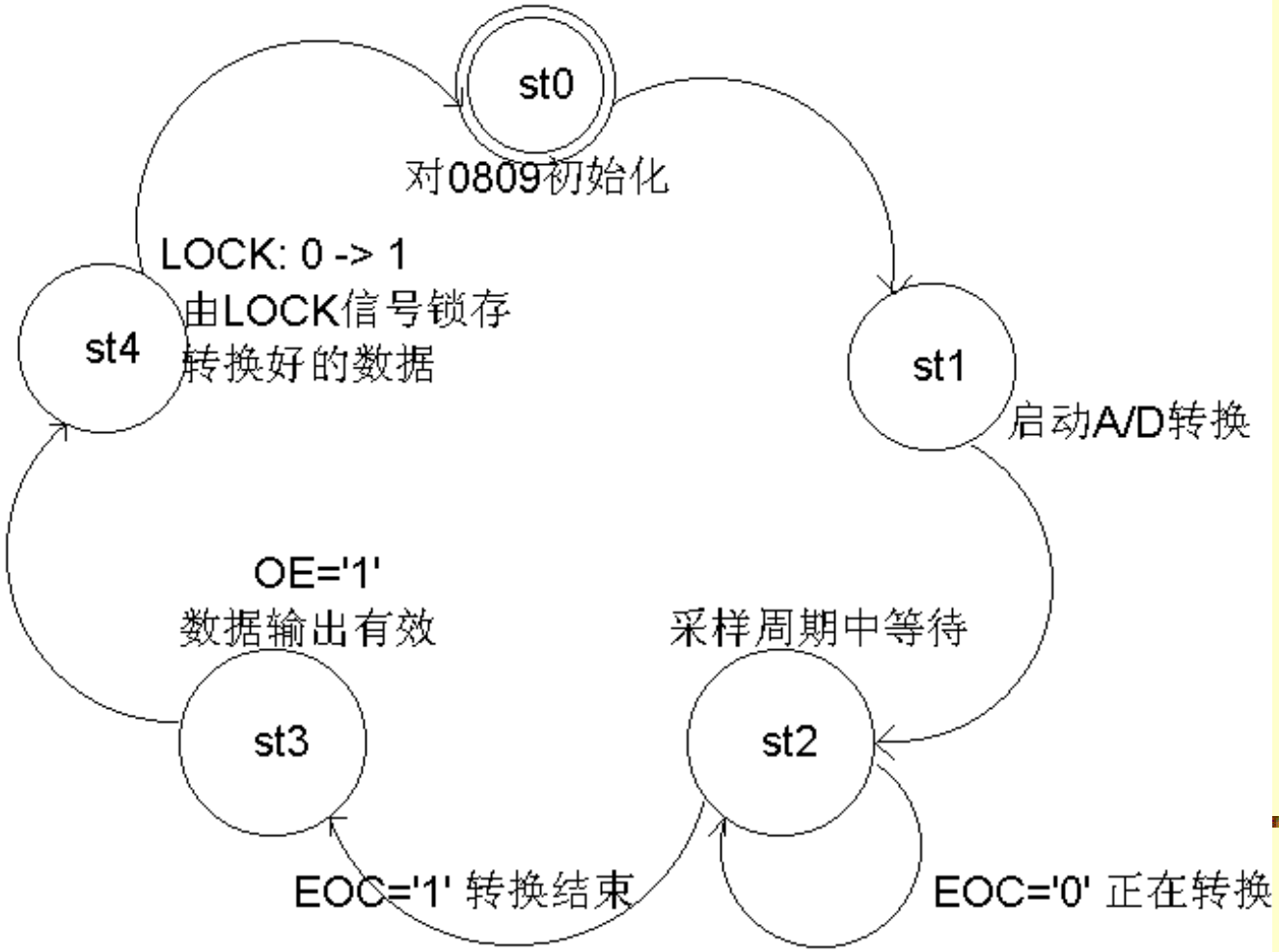


图8-4 控制ADC0809采样状态图

8.2.1 多进程有限状态机

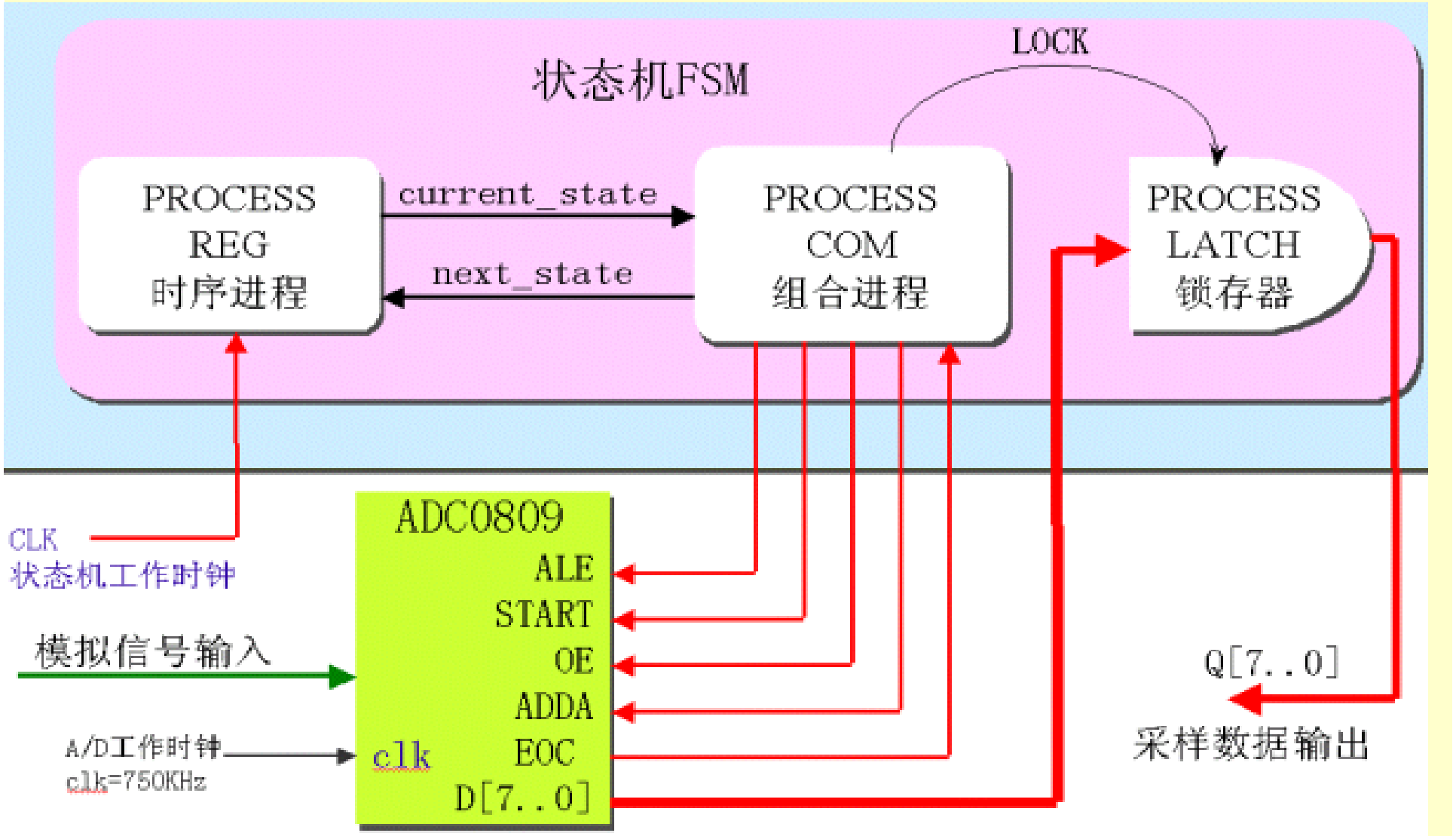


图8-5 采样状态机结构框图

【例8-2】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ADCINT IS
    PORT(D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);  --来自0809转换好的8位数据
CLK  : IN STD_LOGIC;                          --状态机工作时钟
EOC  : IN STD_LOGIC;                --转换状态指示，低电平表示正在转换
ALE  : OUT STD_LOGIC;              --8个模拟信号通道地址锁存信号
START  : OUT STD_LOGIC;             --转换开始信号
OE  : OUT STD_LOGIC;                --数据输出3态控制信号
ADDA  : OUT STD_LOGIC;              --信号通道最低位控制信号
LOCK0  : OUT STD_LOGIC;              --观察数据锁存时钟
Q  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --8位数据输出
END ADCINT;
ARCHITECTURE behav OF ADCINT IS
TYPE states IS (st0, st1, st2, st3,st4) ; --定义各状态子类型
  SIGNAL current_state, next_state: states :=st0 ;
  SIGNAL REGL       : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL LOCK       : STD_LOGIC; -- 转换后数据输出锁存时钟信号
 BEGIN
ADDA <= '1';--当ADDA<='0'，模拟信号进入通道IN0；当ADDA<='1'，则进入通道IN1
Q <= REGL; LOCK0 <= LOCK ;
  COM: PROCESS(current_state,EOC)  BEGIN  --规定各状态转换方式
   CASE current_state IS
   WHEN st0=>ALE<='0';START<='0';LOCK<='0';OE<='0';
        next_state <= st1; --0809初始化
```

（接下页）

```
WHEN st1=>ALE<='1';START<='1';LOCK<='0';OE<='0';
next_state <= st2; --启动采样
   WHEN st2=> ALE<='0';START<='0';LOCK<='0';OE<='0';
     IF (EOC='1') THEN next_state <= st3; --EOC=1表明转换结束
         ELSE next_state <= st2;  END IF ;    --转换未结束，继续等待

   WHEN st3=> ALE<='0';START<='0';LOCK<='0';OE<='1';
next_state <= st4;--开启OE,输出转换好的数据
   WHEN st4=> ALE<='0';START<='0';LOCK<='1';OE<='1'; next_state <= st0;
   WHEN OTHERS => next_state <= st0;
   END CASE ;
 END PROCESS COM ;
  REG: PROCESS (CLK)
    BEGIN
     IF (CLK'EVENT AND CLK='1') THEN current_state<=next_state; END IF;
   END PROCESS REG ;-- 由信号current_state将当前状态值带出此进程:REG
  LATCH1: PROCESS (LOCK) -- 此进程中，在LOCK的上升沿，将转换好的数据锁入
         BEGIN
        IF LOCK='1' AND LOCK'EVENT THEN   REGL <= D ; END IF;
      END PROCESS LATCH1 ;
  END behav;
```

【例8-3】

```
COM1: PROCESS(current_state,EOC)  BEGIN
   CASE current_state IS
   WHEN st0=> next_state <= st1;
   WHEN st1=> next_state <= st2;
   WHEN st2=> IF (EOC='1') THEN next_state <= st3;
                 ELSE next_state <= st2; END IF ;
   WHEN st3=> next_state <= st4;--开启OE
   WHEN st4=> next_state <= st0;
   WHEN OTHERS => next_state <= st0;
   END CASE ;
 END PROCESS COM1 ;
 COM2: PROCESS(current_state)  BEGIN
   CASE current_state IS
   WHEN st0=>ALE<='0';START<='0';LOCK<='0';OE<='0' ;
   WHEN st1=>ALE<='1';START<='1';LOCK<='0';OE<='0' ;
   WHEN st2=>ALE<='0';START<='0';LOCK<='0';OE<='0' ;
   WHEN st3=>ALE<='0';START<='0';LOCK<='0';OE<='1' ;
   WHEN st4=>ALE<='0';START<='0';LOCK<='1';OE<='1' ;
   WHEN OTHERS => ALE<='0';START<='0';LOCK<='0';
   END CASE ;
 END PROCESS COM2 ;
```

KX 康心科技

8.2 Moore型有限状态机设

8.2.1 多进程有限状态机

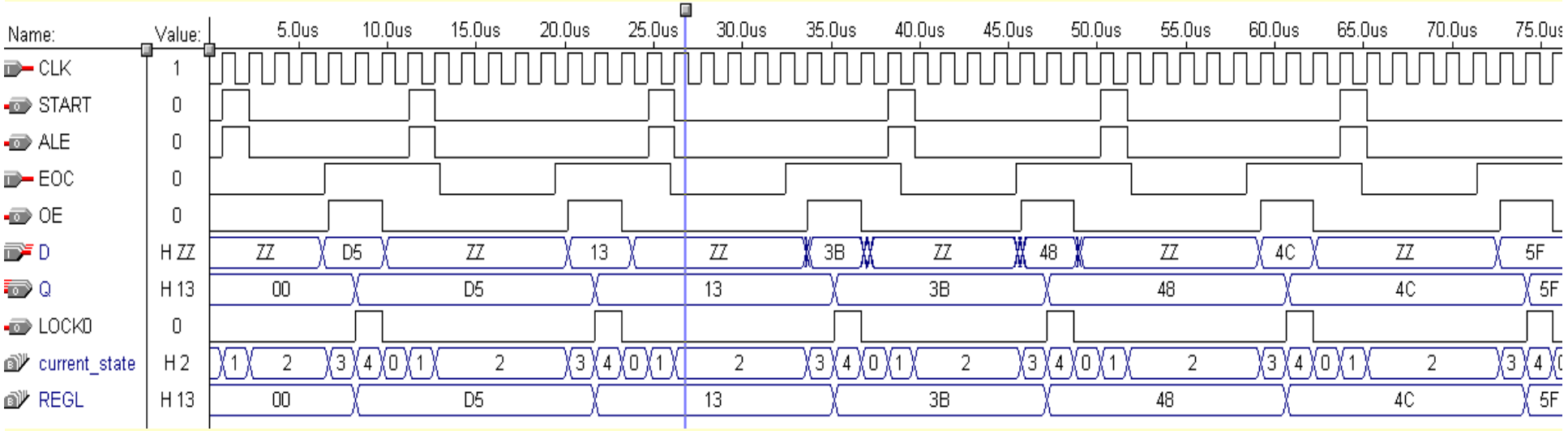


图8-6 ADC0809采样状态机工作时序

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

【例8-4】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MOORE1 IS
  PORT (DATAIN  :IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        CLK,RST : IN STD_LOGIC;
              Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END MOORE1;
ARCHITECTURE behav OF MOORE1 IS
  TYPE ST_TYPE IS (ST0, ST1, ST2, ST3,ST4);
   SIGNAL C_ST : ST_TYPE ;
    BEGIN
    PROCESS(CLK,RST)
     BEGIN
    IF RST ='1' THEN  C_ST <= ST0 ; Q<= "0000" ;
      ELSIF CLK'EVENT AND CLK='1' THEN
```

（接下页）

```
CASE C_ST IS
          WHEN ST0 => IF DATAIN ="10" THEN C_ST <= ST1 ;
                ELSE C_ST <= ST0 ; END IF;
                Q <= "1001" ;
          WHEN ST1 => IF DATAIN ="11" THEN C_ST <= ST2 ;
                ELSE C_ST <= ST1 ;END IF;
                Q <= "0101" ;
          WHEN ST2 => IF DATAIN ="01" THEN C_ST <= ST3 ;
                ELSE C_ST <= ST0 ;END IF;
                Q <= "1100" ;
          WHEN ST3 => IF DATAIN ="00" THEN C_ST <= ST4 ;
                ELSE C_ST <= ST2 ;END IF;
                Q <= "0010" ;
          WHEN ST4 => IF DATAIN ="11" THEN C_ST <= ST0 ;
                ELSE C_ST <= ST3 ;END IF;
                  Q <= "1001" ;
          WHEN OTHERS => C_ST <= ST0;
        END CASE;
    END IF;
  END PROCESS;
END behav;
```

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

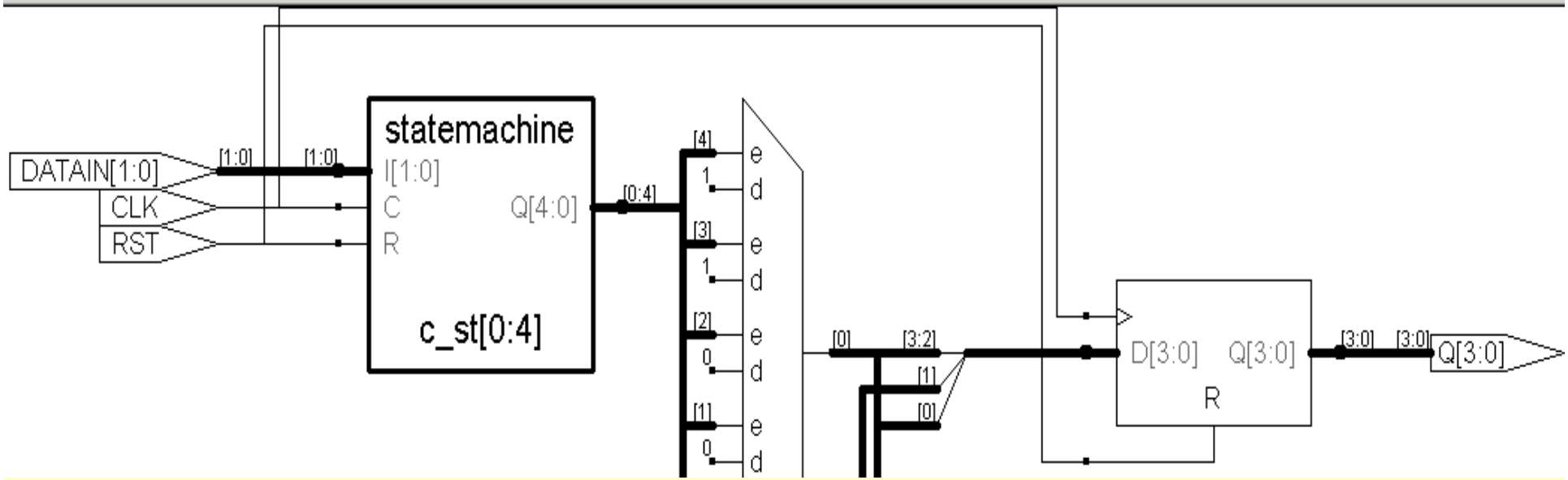


图8-7 例8-4状态机综合后的部分主要RTL电路模块（Synplify综合）

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

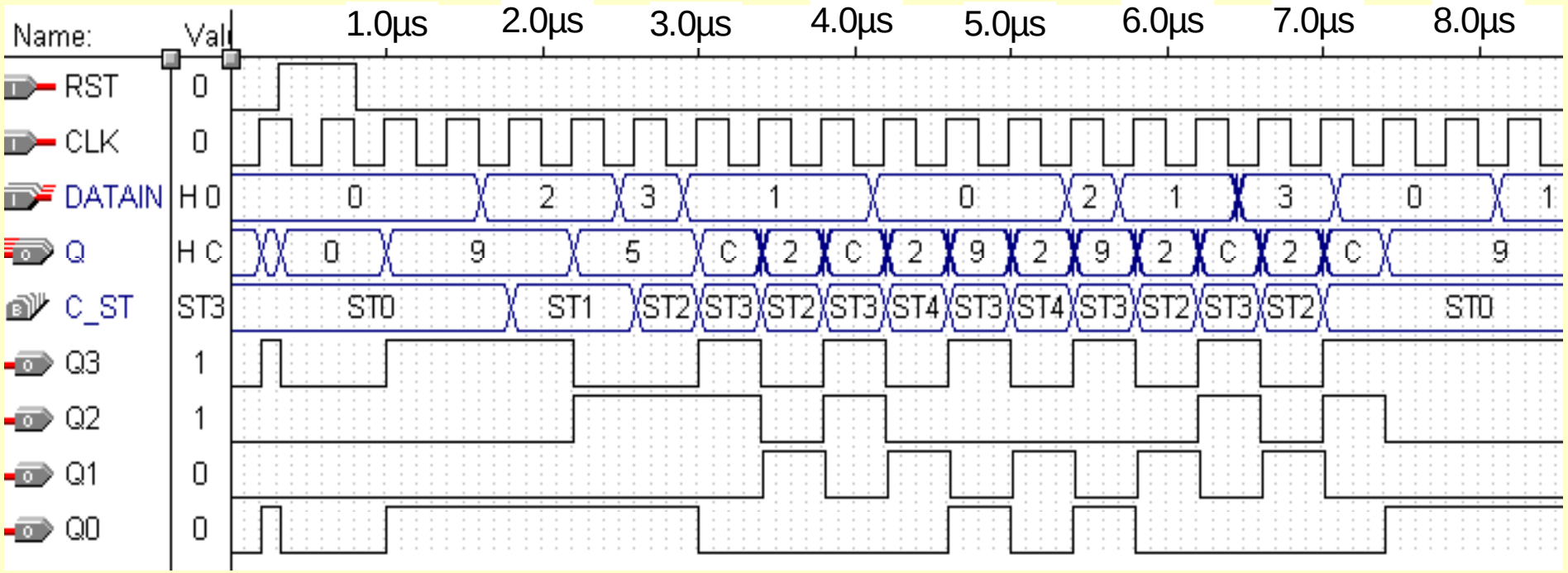


图8-8 例8-4单进程状态机工作时序

KX 康芯科技

8.2 Moore型有限状态机设

8.2.2 单进程Moore型有限状态机

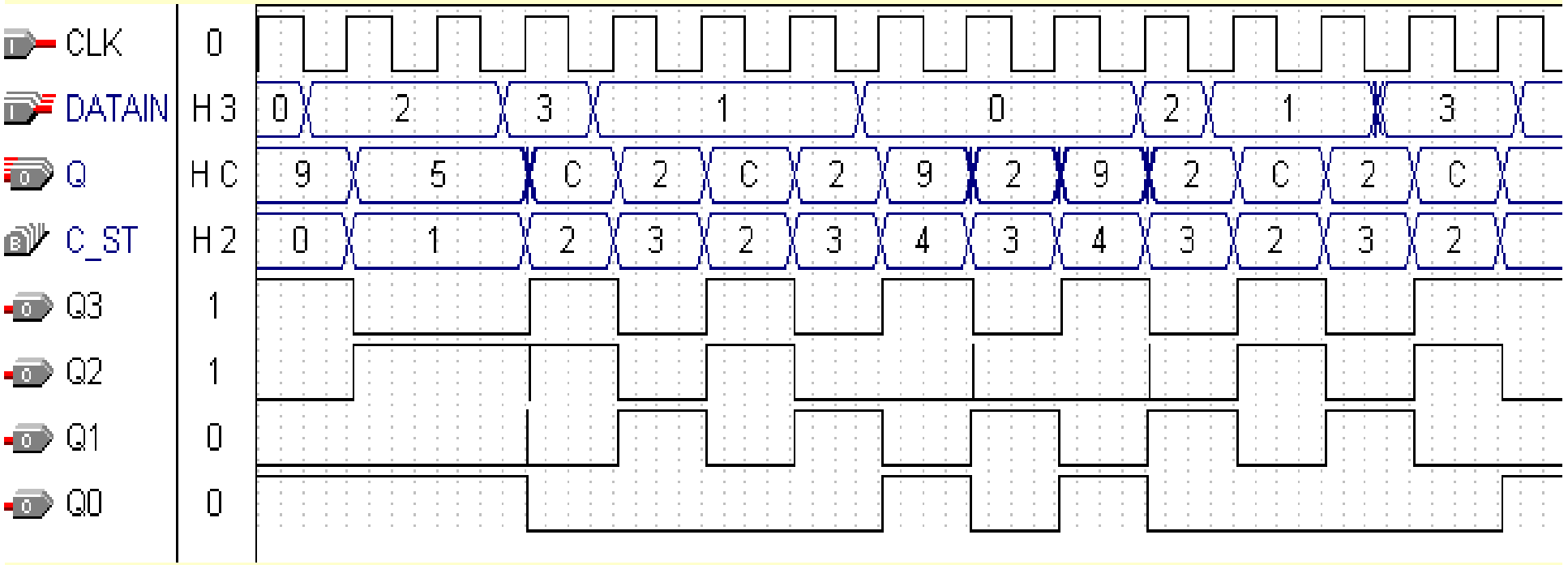


图8-9 对应于例8-4的二进程状态机工作时序图

康芯科技

8.3 Mealy型有限状态机设计

【例8-5】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY1 IS
PORT ( CLK ,DATAIN,RESET  : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END MEALY1;
ARCHITECTURE behav OF MEALY1 IS
  TYPE states IS (st0, st1, st2, st3,st4);
  SIGNAL STX : states  ;
BEGIN
 COMREG : PROCESS(CLK,RESET)  BEGIN --决定转换状态的进程
    IF RESET ='1' THEN    STX <= ST0;
    ELSIF CLK'EVENT AND CLK = '1' THEN    CASE STX IS
        WHEN st0 => IF DATAIN = '1' THEN  STX <= st1; END IF;
        WHEN st1 => IF DATAIN = '0' THEN  STX <= st2; END IF;
```

(接下页)

```
WHEN st2 => IF DATAIN = '1' THEN  STX <= st3; END IF;
        WHEN st3=>  IF DATAIN = '0' THEN  STX <= st4; END IF;
        WHEN st4=>  IF DATAIN = '1' THEN  STX <= st0; END IF;
        WHEN OTHERS => STX <= st0;
      END CASE ;
   END IF;
 END PROCESS COMREG ;
COM1: PROCESS(STX,DATAIN) BEGIN --输出控制信号的进程
   CASE STX IS
     WHEN st0 => IF DATAIN = '1' THEN Q <= "10000" ;
                   ELSE Q<="01010" ; END IF ;
    WHEN st1 => IF DATAIN = '0' THEN Q <= "10111" ;
                   ELSE Q<="10100" ; END IF ;
    WHEN st2 => IF DATAIN = '1' THEN Q <= "10101" ;
                   ELSE Q<="10011" ; END IF ;
    WHEN st3=>  IF DATAIN = '0' THEN Q <= "11011" ;
                   ELSE Q<="01001" ; END IF ;
    WHEN st4=>  IF DATAIN = '1' THEN Q <= "11101" ;
                   ELSE Q<="01101" ; END IF ;
    WHEN OTHERS =>  Q<="00000" ;
  END CASE ;
  END PROCESS COM1 ;
END behav;
```

8.3 Mealy型有限状态机设计

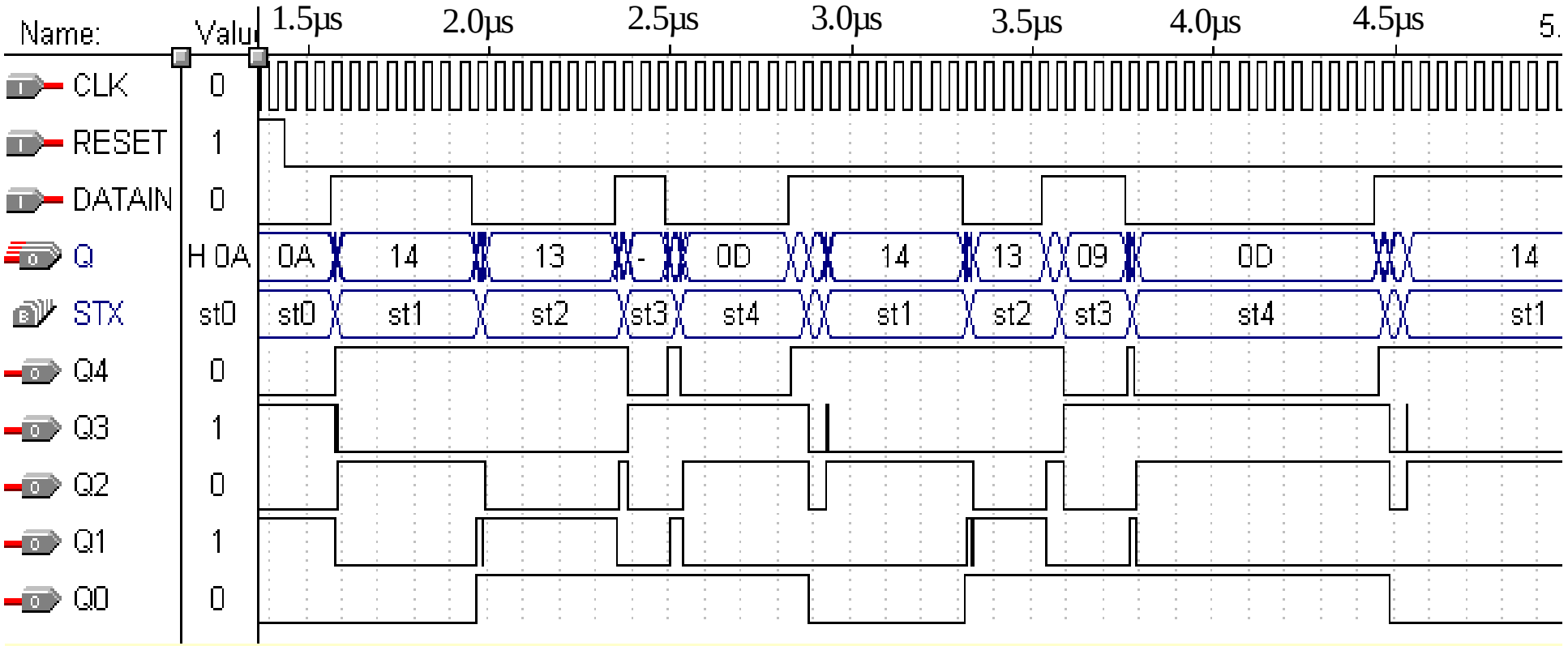


图8-10 例8-5状态机工作时序图

KX 康芯科技

8.3 Mealy型有限状态机设计

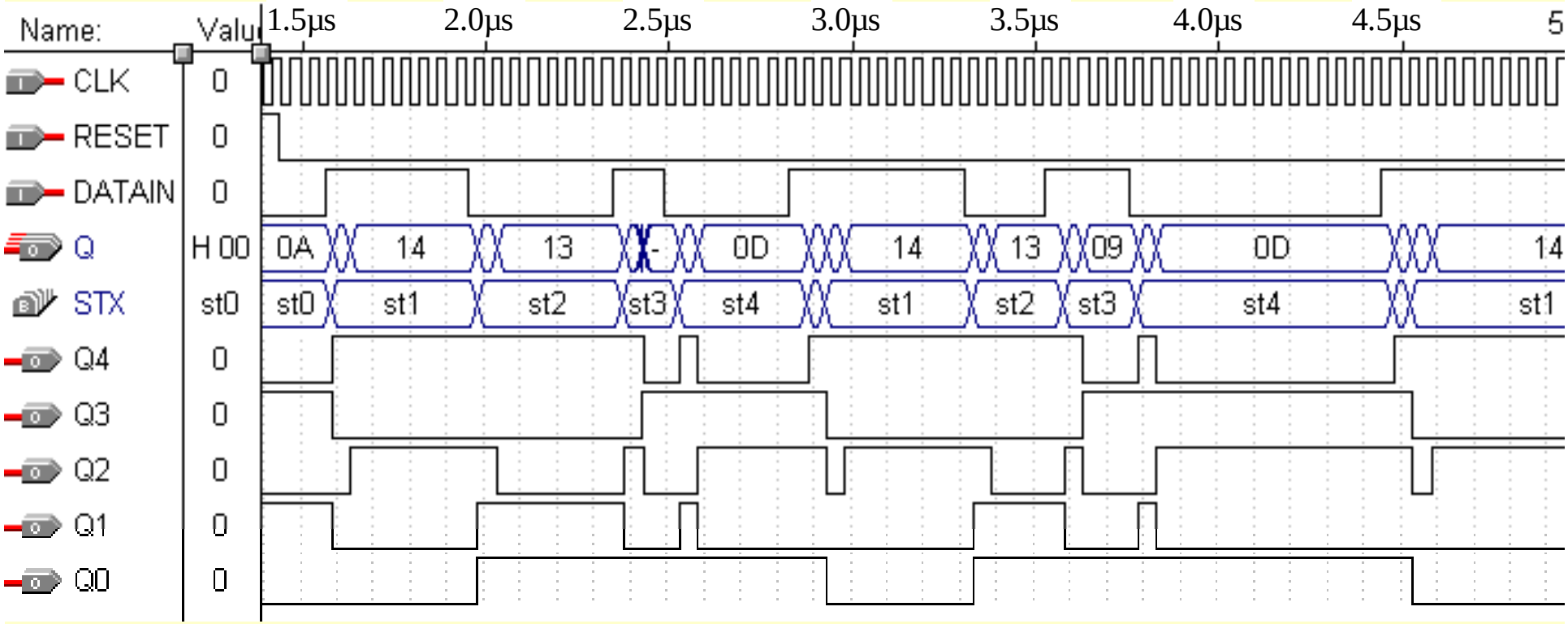


图8-11 例8-6状态机工作时序图

KX 康芯科技

8.3 Mealy型有限状态机设计

【例8-6】

```
LIBRARY IEEE;  --MEALY FSM
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY2 IS
  PORT ( CLK ,DATAIN,RESET  : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END MEALY2;
ARCHITECTURE behav OF MEALY2 IS
   TYPE states IS (st0, st1, st2, st3,st4);
   SIGNAL STX : states  ;
   SIGNAL Q1 : STD_LOGIC_VECTOR(4 DOWNTO 0);
 BEGIN
 COMREG : PROCESS(CLK,RESET)    --决定转换状态的进程
   BEGIN
     IF RESET ='1' THEN    STX <= ST0;
     ELSIF CLK'EVENT AND CLK = '1' THEN
       CASE STX IS
```

（接下页）

```
WHEN st0 => IF DATAIN = '1' THEN  STX <= st1; END IF;
WHEN st1 => IF DATAIN = '0' THEN  STX <= st2; END IF;
         WHEN st2 => IF DATAIN = '1' THEN  STX <= st3; END IF;
         WHEN st3=>  IF DATAIN = '0' THEN   STX <= st4; END IF;
         WHEN st4=>  IF DATAIN = '1' THEN   STX <= st0; END IF;
         WHEN OTHERS => STX <= st0;
        END CASE ;
      END IF;
   END PROCESS COMREG ;
COM1: PROCESS(STX,DATAIN,CLK)  --输出控制信号的进程
    VARIABLE Q2 : STD_LOGIC_VECTOR(4 DOWNTO 0);
  BEGIN
  CASE STX IS
 WHEN st0=> IF DATAIN='1' THEN Q2 :="10000"; ELSE Q2:="01010"; END IF;
  WHEN st1=> IF DATAIN='0' THEN Q2 :="10111"; ELSE Q2:="10100"; END IF;
  WHEN st2=> IF DATAIN='1' THEN Q2 :="10101"; ELSE Q2:="10011"; END IF;
  WHEN st3=> IF DATAIN='0' THEN Q2 :="11011"; ELSE Q2:="01001"; END IF;
  WHEN st4=> IF DATAIN='1' THEN Q2 :="11101"; ELSE Q2:="01101"; END IF;
  WHEN OTHERS =>  Q2:="00000" ;
  END CASE ;
  IF CLK'EVENT AND CLK = '1' THEN   Q1<=Q2;  END IF;
   END PROCESS COM1 ;
 Q <= Q1 ;
END behav;
```

8.4 状态编码

8.4.1 状态位直接输出型编码

表8-1 控制信号状态编码表

状态	状 态 编 码					
	START	ALE	OE	LOCK	B	功 能 说 明
ST0	0	0	0	0	0	初始态
ST1	1	1	0	0	0	启动转换
ST2	0	0	0	0	1	若测得EOC=1时，转下一状态ST3
ST3	0	0	1	0	0	输出转换好的数据
ST4	0	0	1	1	0	利用LOCK的上升沿将转换好的数据锁存

8.4 状态编码

【例8-7】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY AD0809 IS
...   PORT (D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        CLK ,EOC : IN STD_LOGIC;
      ALE, START, OE, ADDA   : OUT STD_LOGIC;
     c_state  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
        Q  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END AD0809;
ARCHITECTURE behav OF AD0809 IS
SIGNAL  current_state, next_state: STD_LOGIC_VECTOR(4 DOWNTO 0 );
  CONSTANT st0 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000" ;
  CONSTANT st1 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "11000" ;
  CONSTANT st2 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00001" ;
  CONSTANT st3 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00100" ;
  CONSTANT st4 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00110" ;
  SIGNAL REGL   : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL REGL   : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL LOCK   : STD_LOGIC;
```

（接下页）

```
   BEGIN
     ADDA <= '1';  Q <= REGL;  START<=current_state(4);
ALE<=current_state(3);
     OE<=current_state(2); LOCK<=current_state(1);c_state
<=current_state;
    COM: PROCESS(current_state,EOC)  BEGIN  --规定各状态转换方式
     CASE current_state IS
     WHEN st0=> next_state <= st1; --0809初始化
     WHEN st1=> next_state <= st2; --启动采样
     WHEN st2=> IF (EOC='1') THEN next_state <= st3; --EOC=1表明转换结束
                   ELSE next_state <= st2;          --转换未结束，继续等待
      END IF ;
     WHEN st3=> next_state <= st4;--开启OE,输出转换好的数据
     WHEN st4=> next_state <= st0;
     WHEN OTHERS => next_state <= st0;
     END CASE ;
   END PROCESS COM ;
    REG: PROCESS (CLK)
       BEGIN
         IF (CLK'EVENT AND CLK='1') THEN current_state<=next_state;
             END IF;
     END PROCESS REG ;       -- 由信号current_state将当前状态值带出此进程:REG
    LATCH1: PROCESS (LOCK) -- 此进程中，在LOCK的上升沿，将转换好的数据锁入
          BEGIN
          IF LOCK='1' AND LOCK'EVENT THEN   REGL <= D ;
          END IF;
         END PROCESS LATCH1 ;
   END behav;
```

8.4 状态编码

8.4.1 状态位直接输出型编码

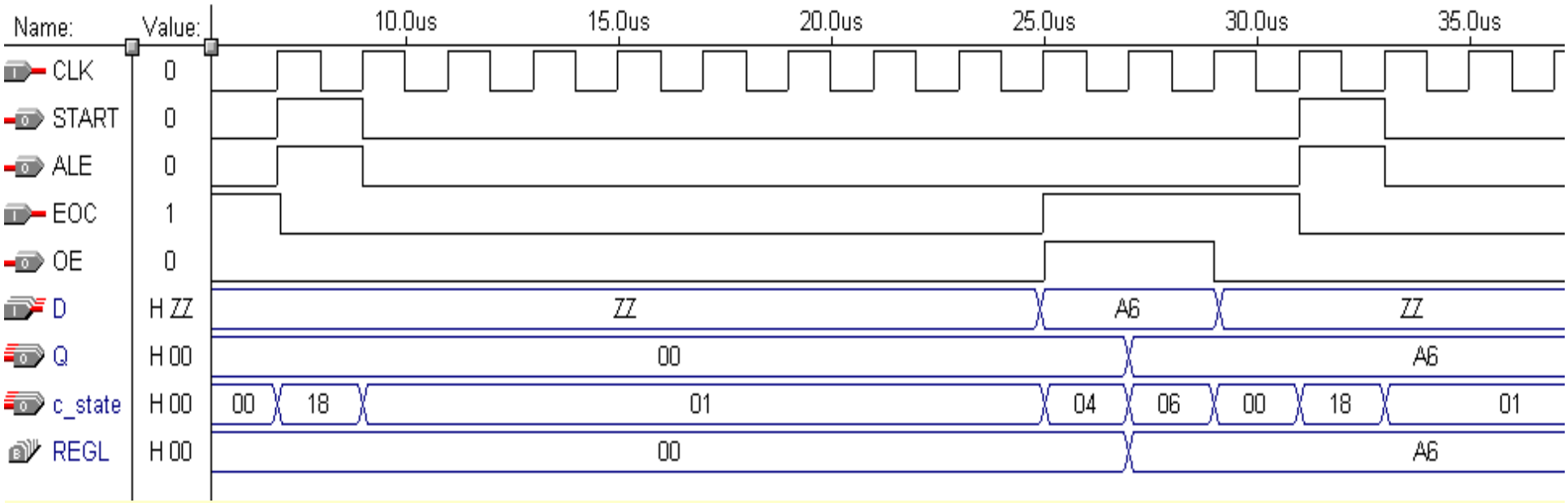


图8-12 例8-7状态机工作时序图

康芯科技

8.4 状态编码

8.4.2 顺序编码

表8-2 编码方式

状 态	顺序编码	一位热码编码
STATE0	**000**	**100000**
STATE1	**001**	**010000**
STATE2	**010**	**001000**
STATE3	**011**	**000100**
STATE4	**100**	**000010**
STATE5	**101**	**000001**

8.4 状态编码

8.4.2 顺序编码

【例8-8】

```
...
SIGNAL CRURRENT_STATE,NEXT_STATE: STD_LOGIC_VECTOR(2 DOWNTO 0 );
CONSTANT ST0 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000" ;
CONSTANT ST1 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001" ;
CONSTANT ST2 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010" ;
CONSTANT ST3 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011" ;
CONSTANT ST4 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100" ;
...
```

8.4.3 一位热码编码

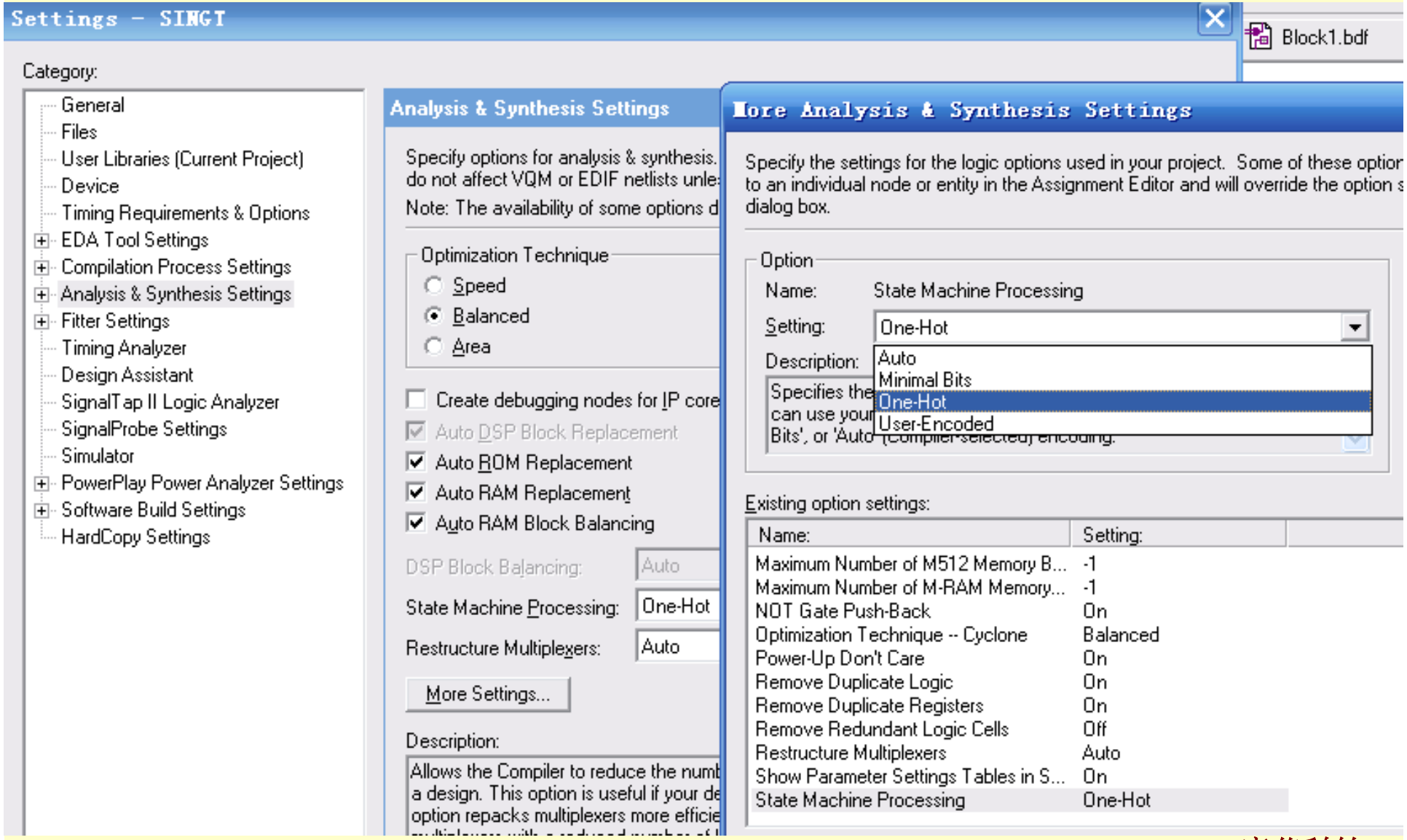


图8-13 一位热码编码方式选择对话框

KX 康芯科技

8.5 非法状态处理

表8-3 剩余状态

状 态	st0	St1	St2	St3	St4	st_ilg1	st_ilg2	st_ilg3
顺序编码	000	001	010	011	100	101	110	111

(1) 在语句中对每一个非法状态都作出明确的状态转换指示，如在原来的CASE语句中增加诸如以下语句：

```
WHEN st_ilg1 =>   next_state <= st0;
WHEN st_ilg2 =>   next_state <= st0;
   ...
```

(2) 如例8-9那样，利用OTHERS语句中对未提到的状态作统一处理。

8.5 非法状态处理

【例8-9】

```
  ...
  TYPE states IS (st0, st1, st2, st3，st4，  st_ilg1，
st_ilg2 ，st_ilg3);
  SIGNAL current_state, next_state: states;
  ...
  COM：PROCESS(current_state, state_Inputs)  -- 组合逻辑进程
  BEGIN
    CASE current_state IS                  -- 确定当前状态的状态值
        ...
        WHEN OTHERS =>   next_state <= st0;
    END case;
```

8.5 非法状态处理

【例8-10】

```
...
alarm <= (st0 AND (st1 OR st2 OR st3 OR st4 OR st5)) OR
          (st1 AND (st0 OR st2 OR st3 OR st4 OR st5)) OR
          (st2 AND (st0 OR st1 OR st3 OR st4 OR st5)) OR
          (st3 AND (st0 OR st1 OR st2 OR st4 OR st5)) OR
          (st4 AND (st0 OR st1 OR st2 OR st3 OR st5)) OR
     (st5 AND (st0 OR st1 OR st2 OR st3 OR st4)) ;
```

习 题

8-1. 仿照例**8-1**，将例**8-4**用两个进程，即一个时序进程，一个组合进程表达出来。

8-2. 为确保例**8-5**的状态机输出信号没有毛刺，试用例**8-4**的方式构成一个单进程状态，使输出信号得到可靠锁存，在相同输入信号条件下，给出两程序的仿真波形。

8-3. 序列检测器可用于检测一组或多组由二进制码组成的脉冲序列信号，当序列检测器连续收到一组串行二进制码后，如果这组码与检测器中预先设置的码相同，则输出**1**，否则输出**0**。由于这种检测的关键在于正确码的收到必须是连续的，这就要求检测器必须记住前一次的正确码及正确序列，直到在连续的检测中所收到的每一位码都与预置数的对应码相同。在检测过程中，任何一位不相等都将回到初始状态重新开始检测。例**8-11**描述的电路完成对序列数“**11100101**”的检测，当这一串序列数高位在前(左移)串行进入检测器后，若此数与预置的密码数相同，则输出“**A**”，否则仍然输出“**B**”。

习 题

【例8-11】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SCHK IS
  PORT(DIN，CLK，CLR  : IN STD_LOGIC; --串行输入数据位/工作时钟/复位信号
       AB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)); --检测结果输出
END SCHK;
ARCHITECTURE behav OF SCHK IS
    SIGNAL Q : INTEGER RANGE 0 TO 8 ;
    SIGNAL D : STD_LOGIC_VECTOR(7 DOWNTO 0); --8位待检测预置数(密码=E5H)
BEGIN
    D <= "11100101 " ; --8位待检测预置数
  PROCESS( CLK, CLR )
  BEGIN
  IF CLR = '1' THEN    Q <= 0 ;
  ELSIF  CLK'EVENT AND CLK='1' THEN  --时钟到来时，判断并处理当前输入的位
 CASE Q IS
  WHEN 0=>  IF DIN = D(7) THEN Q <= 1 ; ELSE Q <= 0 ; END IF ;
```

（接下页）

习 题

```
    WHEN 1=>  IF DIN = D(6) THEN Q <= 2 ; ELSE Q <= 0 ; END IF ;
    WHEN 2=>  IF DIN = D(5) THEN Q <= 3 ; ELSE Q <= 0 ; END IF ;
    WHEN 3=>  IF DIN = D(4) THEN Q <= 4 ; ELSE Q <= 0 ; END IF ;
    WHEN 4=>  IF DIN = D(3) THEN Q <= 5 ; ELSE Q <= 0 ; END IF ;
    WHEN 5=>  IF DIN = D(2) THEN Q <= 6 ; ELSE Q <= 0 ; END IF ;
    WHEN 6=>  IF DIN = D(1) THEN Q <= 7 ; ELSE Q <= 0 ; END IF ;
    WHEN 7=>  IF DIN = D(0) THEN Q <= 8 ; ELSE Q <= 0 ; END IF ;
    WHEN OTHERS =>  Q <= 0 ;
          END CASE ;
      END IF ;
    END PROCESS ;
    PROCESS( Q )                                    --检测结果判断输出
    BEGIN
        IF Q = 8  THEN  AB <= "1010" ;      --序列数检测正确，输出 “A”
        ELSE              AB <= "1011" ;      --序列数检测错误，输出 “B”
        END IF ;
    END PROCESS ;
END behav ;
```

习 题

要求1：说明例8-11的代码表达的是什么类型的状态机，它的优点是什么？详述其功能和对序列数检测的逻辑过程。要求2：根据例8-11写出由两个主控进程构成的相同功能的符号化Moore型有限状态机，画出状态图，并给出其仿真测试波形。要求3：将8位待检测预置数作为外部输入信号，即可以随时改变序列检测器中的比较数据。写出此程序的符号化单进程有限状态机。

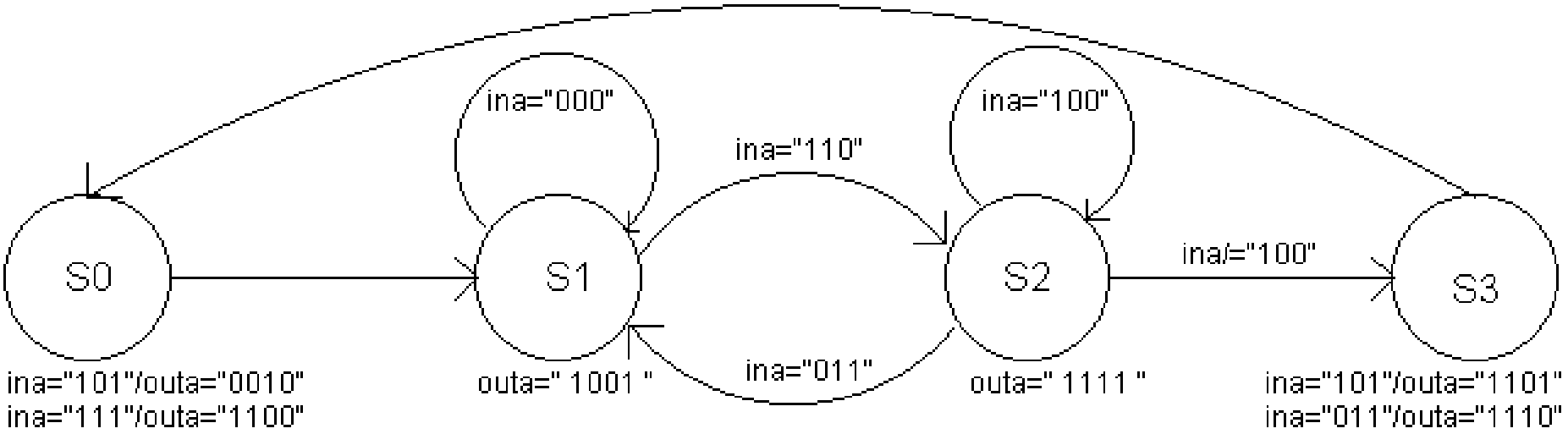
提示：对于D<=“11100101”，电路需分别不间断记忆：初始状态、1、11、111 、 1110 、11100、111001、1110010、11100101 共9种状态。

习 题

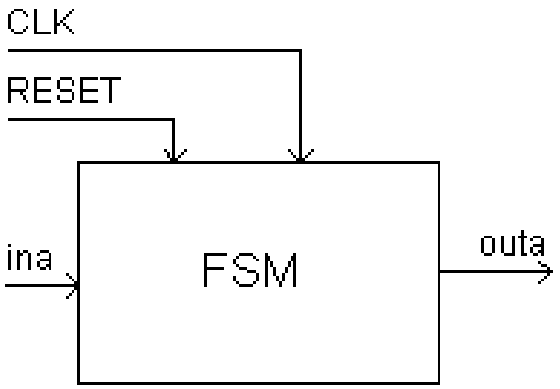
8-4. 根据图8-14(a)所示的状态图，分别按照图8-4(b)和图8-14(c)写出对应结构的VHDL状态机。

8-5. 在不改变原代码功能的条件下用两种方法改写例8-2，使其输出的控制信号(ALE、START、OE、LOCK)没有毛刺。方法1：将输出信号锁存后输出；方法2：使用状态码直接输出型状态机，并比较这三种状态机的特点。

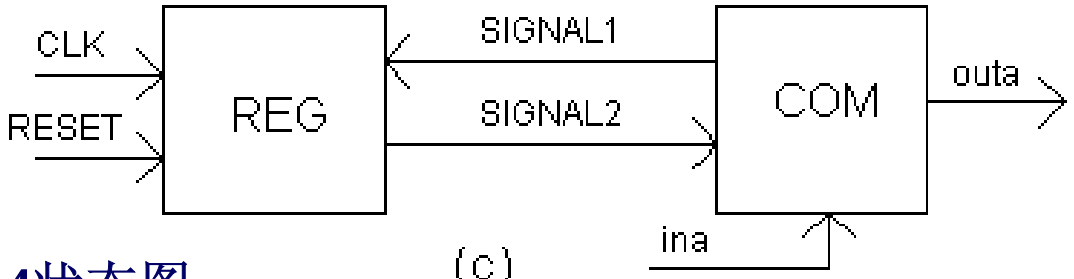
习 题



(a)



(b)



(c)

图8-14 习题8-4状态图

实 验 与 设 计

8-1. 序列检测器设计

(1) 实验目的：用状态机实现序列检测器的设计，了解一般状态机的设计与应用。

(2) 实验原理：序列检测器的工作原理已在习题**8-3**中作了说明。

(3) 实验内容1：仔细完成习题**8-3**的全部内容，利用**QuartusII**对例**8-11**进行文本编辑输入、仿真测试并给出仿真波形，了解控制信号的时序，最后进行引脚锁定并完成硬件测试实验。

建议选择电路模式**No.8**（附录图**9**），用键**7(PIO11)**控制复位信号**CLR**；键**6(PIO9)**控制状态机工作时钟**CLK**；待检测串行序列数输入**DIN**接**PIO10(**左移，最高位在前**)**；指示输出**AB**接**PIO39～PIO36(**显示于数码管**6)**。下载后：①按实验板“系统复位”键；②用键**2**和键**1**输入**2**位十六进制待测序列数“**1100101**”；③按键**7**复位**(**平时数码**6**指示显“**B**”**)**；④按键**6(CLK) 8**次，这时若串行输入的**8**位二进制序列码**(**显示于数码**2/1**和发光管**D8～D0)**与预置码“**1100101**”相同，则数码**6**应从原来的**B**变成**A**，表示序列检测正确，否则仍为**B**。

实 验 与 设 计

8-1. 序列检测器设计

(4) 实验内容2：根据习题8-3中的要求3提出的设计方案，重复以上实验内容(将8位待检测预置数由键4/键3作为外部输入，从而可随时改变检测密码)。

(5) 实验思考题：如果待检测预置数必须以右移方式进入序列检测器，写出该检测器的VHDL代码(两进程符号化有限状态机)，并提出测试该序列检测器的实验方案。

(6) 实验报告：根据以上的实验内容写出实验报告，包括设计原理、程序设计、程序分析、仿真分析、硬件测试和详细实验过程。

实 验 与 设 计

8-2. ADC0809采样控制电路实现

(1) 实验目的：学习用状态机对A/D转换器ADC0809的采样控制电路的实现。

(2) 实验原理：ADC0809的采样控制原理已在8.2.1节中作了详细说明(实验程序用例8-2)。

ADC0809是CMOS的8位A/D转换器，片内有8路模拟开关，可控制8个模拟量中的一个进入转换器中。转换时间约100μs，含锁存控制的8路多路开关，输出有三态缓冲器控制，单5V电源供电。

主要控制信号如图8-3所示：START是转换启动信号，高电平有效；ALE是3位通道选择地址(ADDC、ADDB、ADDA)信号的锁存信号。当模拟量送至某一输入端(如IN1或IN2等)，由3位地址信号选择，而地址信号由ALE锁存；EOC是转换情况状态信号，当启动转换约100μs后，EOC产生一个负脉冲，以示转换结束；在EOC的上升沿后，若使输出使能信号OE为高电平，则控制打开三态缓冲器，把转换好的8位数据结果输至数据总线，至此ADC0809的一次转换结束。实验过程。

实 验 与 设 计

8-2. ADC0809采样控制电路实现

(3) 实验内容：利用**QuartusII**对例**8-2**进行文本编辑输入和仿真测试；给出仿真波形。最后进行引脚锁定并进行测试，硬件验证例**8-2**电路对**ADC0809**的控制功能。

测试步骤：建议选择电路模式**No.5**，由对应的电路图可见，**ADC0809**的转换时钟**CLK**已经事先接有**750kHz**的频率，引脚锁定为：**START**接**PIO34**，**OE**（**ENABLE**）接**PIO35**，**EOC**接**PIO8**，**ALE**接**PIO33**，状态机时钟**CLK**接**clock0**，**ADDA**接**PIO32(ADDB**和**ADDC**都接**GND)**，**ADC0809**的**8**位输出数据线接**PIO23**～**PIO16**，锁存输出**Q**显示于数码**8/**数码**7(PIO47**～**PIO40)**。

(4) 实验思考题：在不改变原代码功能的条件下将例**8-2**表达成用状态码直接输出型的状态机。

(5) 实验报告：根据以上的实验要求、实验内容和实验思考题写出实验报告。

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(1) 实验目的：掌握**LPM RAM**模块**VHDL**元件定制、调用和使用方法；熟悉**A/D**和**D/A**与**FPGA**接口电路设计；了解**HDL**文本描述与原理图混合设计方法。

(2) 实验原理：主要内容参考本章和**7.2**节。本设计项目是利用**FPGA**直接控制**0809**对模拟信号进行采样，然后将转换好的**8**位二进制数据迅速存储到存储器中，在完成对模拟信号一个或数个周期的采样后，由外部电路系统**(**如单片机**)**将存储器中的采样数据读出处理。**1.** 元件**ADCINT**。**ADCINT**是控制**0809**的采样状态机，其**VHDL**描述以及其输入输出信号的含义与例**8-2**完全相同，工作方式可参考**8.2.1**节；。

(3) 实验内容1：设**ADDA='1'**；即模拟信号来自**0809**的**IN1**口（可用实验系统右下角的电位器产生被测模拟信号）完成此项设计，给出仿真波形及其分析，将设计结果在**Cyclone**中硬件实现，用**QuartusII**的在系统**RAM/ROM**数据编辑器了解采入**RAM**中的数据。

(4) 实验内容2：优化设计。仿真设计电路图**8-15**，检查此项设计得**START**信号是否有毛刺，如果有，改进**ADCINT**的设计（也可用其他方法），排除**START**的毛刺。

实 验 与 设 计

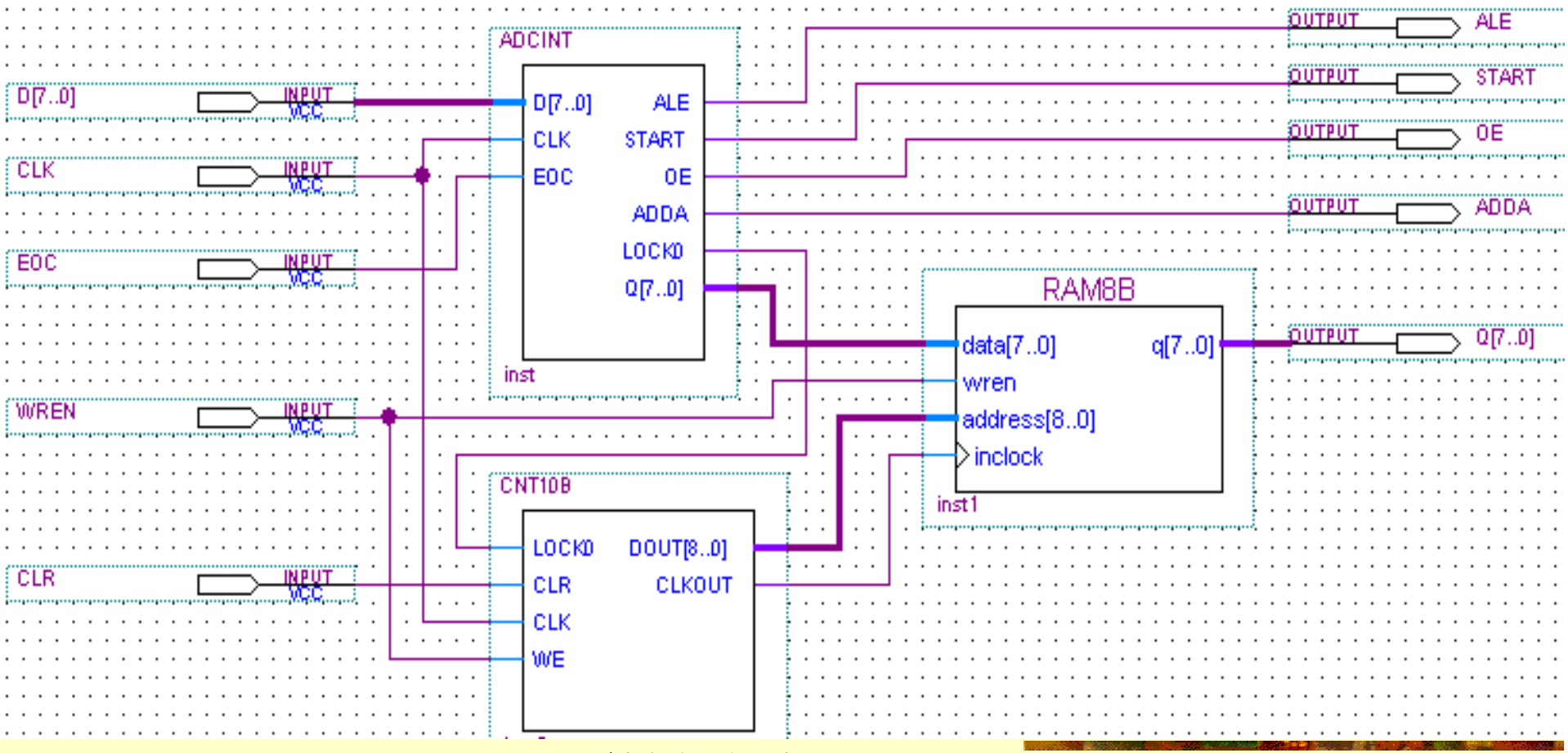


图8-15 ADC0809采样电路系统：RSV.bdf

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(5) 实验内容3：对电路图**8-15**完成设计和仿真后锁定引脚，进行硬件测试。参考实验**8-2**和**7-1**对**0809**和**0832**的引脚锁定：元件**“ADCINT”**引脚锁定参考实验**8-2**。**WE**用键**1**控制；为了实验方便，**CLK**接**clock0**，频率先选择**64Hz**（选择较慢的采样时钟），作状态机工作时钟。硬件实验中，建议选择电路模式**No.5**，打开**+/-12V**电源，首先使**WE=’1’**，即键**1**置高电平，允许采样，由于这时的程序中设置**ADDA <= '1'**，模拟信号来自**AIN1**，即可通过调协实验板上的电位器（此时的模拟信号是手动产生的），将转换好的数据采入**RAM**中；然后按键**1**，使**WE=’0’**，**clock0**的频率选择**16384Hz**（选择较高时钟），即能从示波器中看见被存于**RAM**中的数据（可以首先通过**QuartusII**的**RAM**在系统读写器观察已采入**RAM**中的数据）。

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(6) 实验内容4：程序中设置**ADDA <= '0'**，模拟信号将由**AIN0**进入，即**AIN0**的输入信号来自外部信号源的模拟连续信号。外部模拟信号可来自实验箱，方法如下：

首先打开**+/-12V**电源，将**GW48**主系统板右侧的**“JL11”**跳线座短路**“L_F”**端；跳线座**“JP18”**的**“INPUT”**端与系统右下角的时钟**64Hz**相接；并用一插线将插座**“JP17”**的**“OUTPUT”**端与实验箱最左侧的**“JL10”**坐的**“AIN0”**端相接，这样就将**64Hz**待采样的模拟信号接入了**0809**的**IN0**端（注意，这时例**8-2/12**程序中设置**ADDA <= '0'**）。试调节**“JP18”**上方的电位器，使得主系统右侧的**“WAVE OUT”**端输出正常信号波形（用示波器监视，峰值调在**4V**以下）。注意，如果要将采入（用**CLK=64Hz**采样）**RAM**中的数据扫描显示到示波器上观察，必须用高频率时钟才行（**clock0**接**16384Hz**）。

可以使键**1**高电平是对模拟信号采样，低电平时示波器显示已存入**RAM**的波形数据。

实 验 与 设 计

8-3. 数据采集电路和简易存储示波器设计

(6) 实验内容5：仅按照以上方法，会发现示波器显示的波形并不理想，原因是从**RAM**中扫出的数据都不是一个完整的波形周期。试设计一个状态机，结合被锁入**RAM**中的某些数据，改进元件**CNT10B**，使之存入**RAM**中的数据和通过**D/A**在示波器上扫出的数据都是一个或数个完整波形数据。

(7) 实验内容6：在图**8-15**的电路中增加一个锯齿波发生器，扫描时钟与地址发生器的时钟一致。锯齿波数据通过另一个**D/A**输出，控制示波器的**X**端（不用示波器内的锯齿波信号），而**Y**端由原来的**D/A**给出**RAM**中的采样信息，由此完成一个比较完整的存储示波器的显示控制。

8-3. 数据采集电路和简易存储示波器设计

【例8-12】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY CNT10B IS
      PORT (LOCK0,CLR : IN STD_LOGIC;
                  CLK : IN STD_LOGIC;
                   WE : IN STD_LOGIC;
                 DOUT : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
               CLKOUT : OUT STD_LOGIC );
     END CNT10B;
ARCHITECTURE behav OF CNT10B IS
      SIGNAL CQI  :  STD_LOGIC_VECTOR(8 DOWNTO 0);
      SIGNAL CLK0 :  STD_LOGIC;
BEGIN
CLK0 <= LOCK0 WHEN WE='1' ELSE
        CLK;
  PROCESS(CLK0,CLR,CQI)
   BEGIN
    IF CLR = '1' THEN  CQI <= "000000000";
    ELSIF CLK0'EVENT AND CLK0 = '1' THEN  CQI <= CQI + 1; END IF;
  END PROCESS;
        DOUT <= CQI; CLKOUT <= CLK0;
END behav;
```

实 验 与 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

(1) 实验目的：学习较复杂状态机的设计。

(2) 实验原理：图8-19是一个用比较器LM311和DAC0832构成的8位A/D转换器的电路框图。其工作原理是：当被测模拟信号电压vi接于LM311的“+”输入端时，由FPGA产生自小到大的搜索数据加于DAC0832后，LM311的“-”端将得到一个比较电压vc；当vc<vi时，LM311的“1”脚输出高电平’1’，而当vc>vi时，LM311输出低电平。在LM311输出由’1’到’0’的转折点处，FPGA输向0832数据必定与待测信号电压vi成正比。由此数即可算得vi的大小。

(3) 实验内容1：例8-13是图8-16中FPGA的一个简单的示例性程序。

实 验 与 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

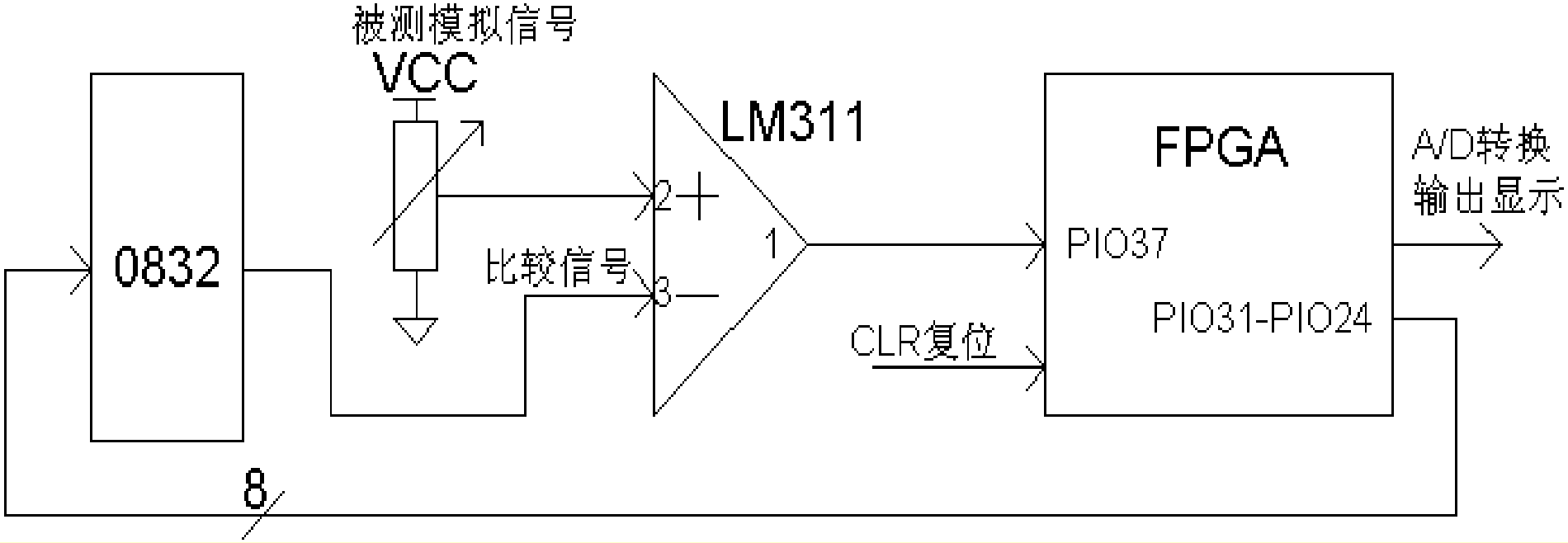


图8-16 比较器和D/A构成A/D电路框图。

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

【例8-13】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DAC2ADC IS
    PORT ( CLK     :  IN STD_LOGIC; --计数器时钟
           LM311   :  IN STD_LOGIC;  --LM311输出，由PIO37口进入FPGA
           CLR     :  IN STD_LOGIC;  --计数器复位
            DD     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;--输向0832的数据
         DISPDATA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );--转换数据显示
END;
ARCHITECTURE DACC OF DAC2ADC IS
 SIGNAL CQI  : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
 BEGIN
 DD <= CQI ;
PROCESS(CLK, CLR, LM311)
    BEGIN
      IF CLR = '1' THEN   CQI <= "00000000";
       ELSIF CLK'EVENT AND CLK = '1' THEN
        IF LM311 = '1' THEN CQI <= CQI + 1;  END IF;--如果是高电平，继续搜索
            END IF;         --如果出现低电平，即可停止搜索，保存计数值于CQI中
    END PROCESS;
 DISPDATA <= CQI  WHEN LM311='0' ELSE "00000000" ;--将保存于CQI中的数输出
 END;
```

实 验 与 设 计

8-4. 比较器和D/A器件实现A/D转换功能的电路设计

(3) 实验内容2： 例**8-14**的缺点有**2**个：**1**、无法自动搜索被测信号，每次测试都必须复位一次；**2**、由于每次搜索都是从**0**开始，从而**“A/D**转换**”**速度太慢。

试设计一个控制搜索的状态机，克服这两个缺点。且尽量提高“转换”速度，如安排一个特定的算法（如黄金分割法）进行快速搜索。

实 验 与 设 计

8-5. 通用异步收发器设计

(1) 实验目的：学习利用状态机设计通用异步RS232硬件通信模块，并根据图8-22设计信号采集、分析与通信模块。

(2) 实验原理：UART即Universal Asynchronous Receiver Transmitter通用异步收发器，是一种应用广泛的短距离串行传输接口。往往用于短距离、低速、低成本的微机与下位机的通讯中。8250、8251、NS16450等芯片都是常见的UART器件。这类芯片有些已经做得相当复杂，含有许多辅助的模块，比如FIFO。图8-17是常见的UART连接通信图。注意，图8-17中两边的TXD、RXD信号是交错的。TXD是UART发送端，为输出；RXD是UART接收端，为输入。在TXD、RXD信号线上的电平也不是普通的TTL 5V电平，而是RS232的接口电平。

实 验 与 设 计

8-5. 通用异步收发器设计

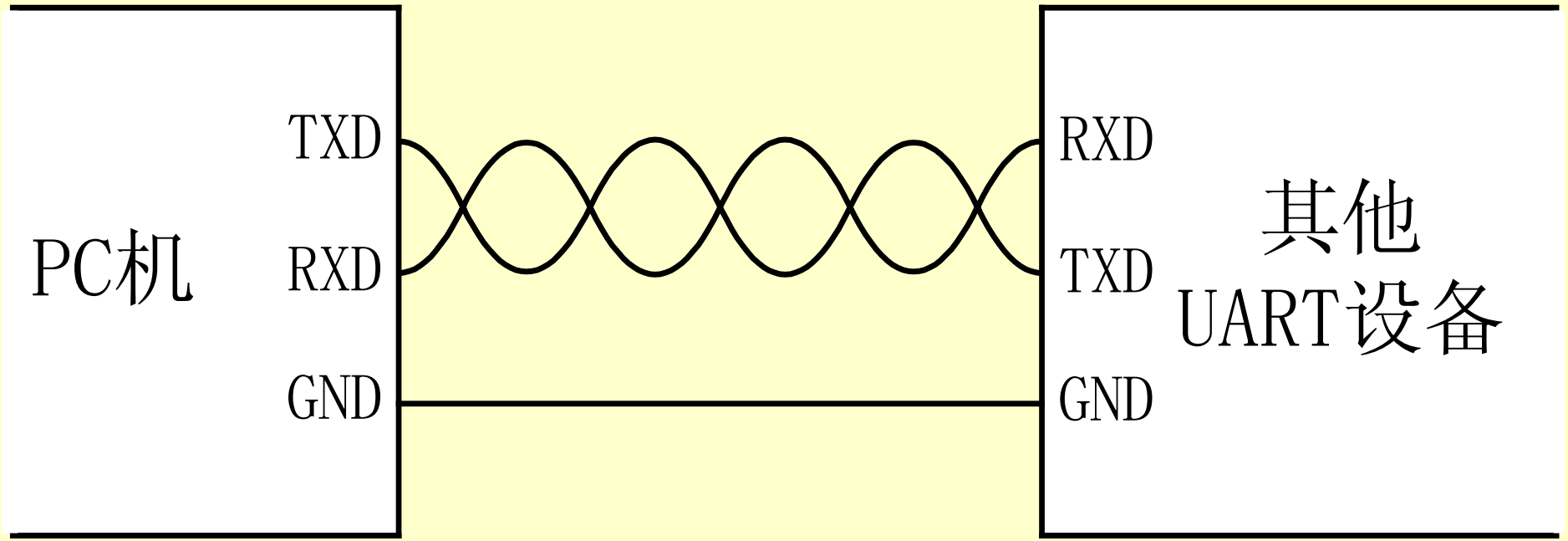


图8-17 UART三线连接通信示意

实 验 与 设 计

8-5. 通用异步收发器设计

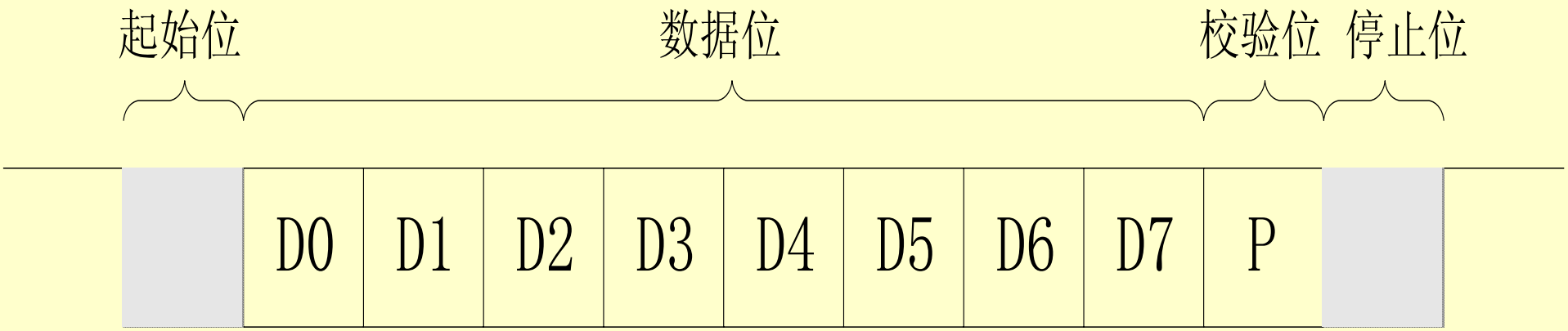


图8-18 基本UART帧格式

实 验 与 设 计

8-5. 通用异步收发器设计

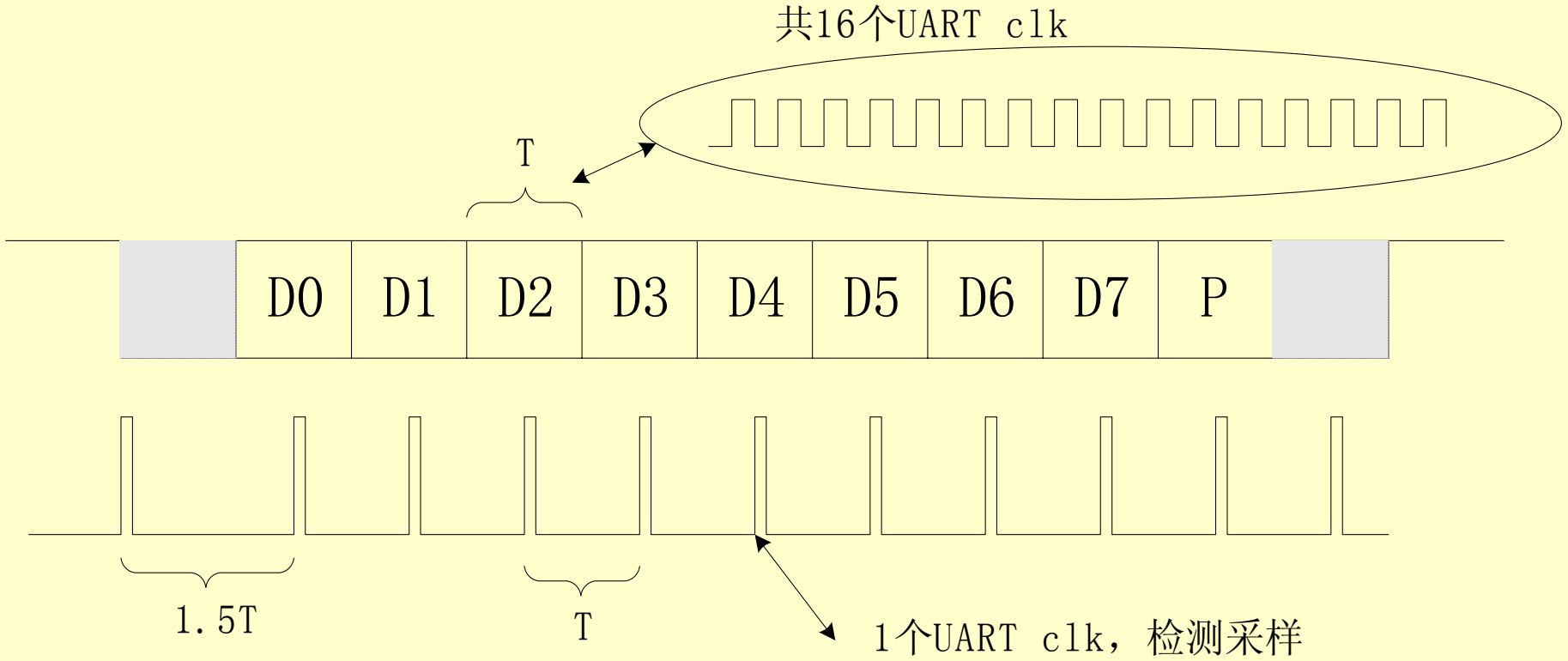


图8-19 基本UART帧时序

8-5. 通用异步收发器设计

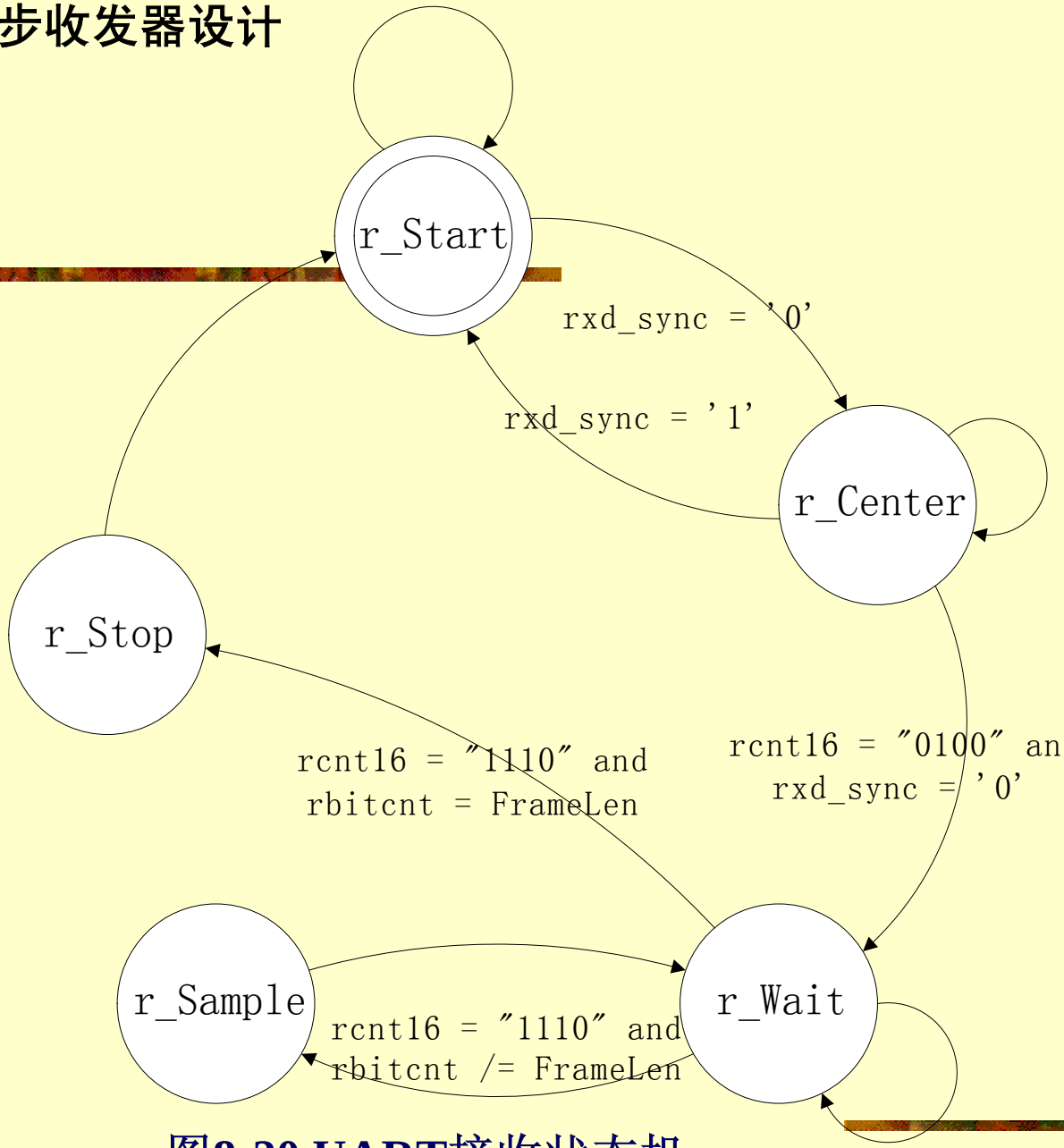


图8-20 UART接收状态机

8-5. 通用异步收发器设计

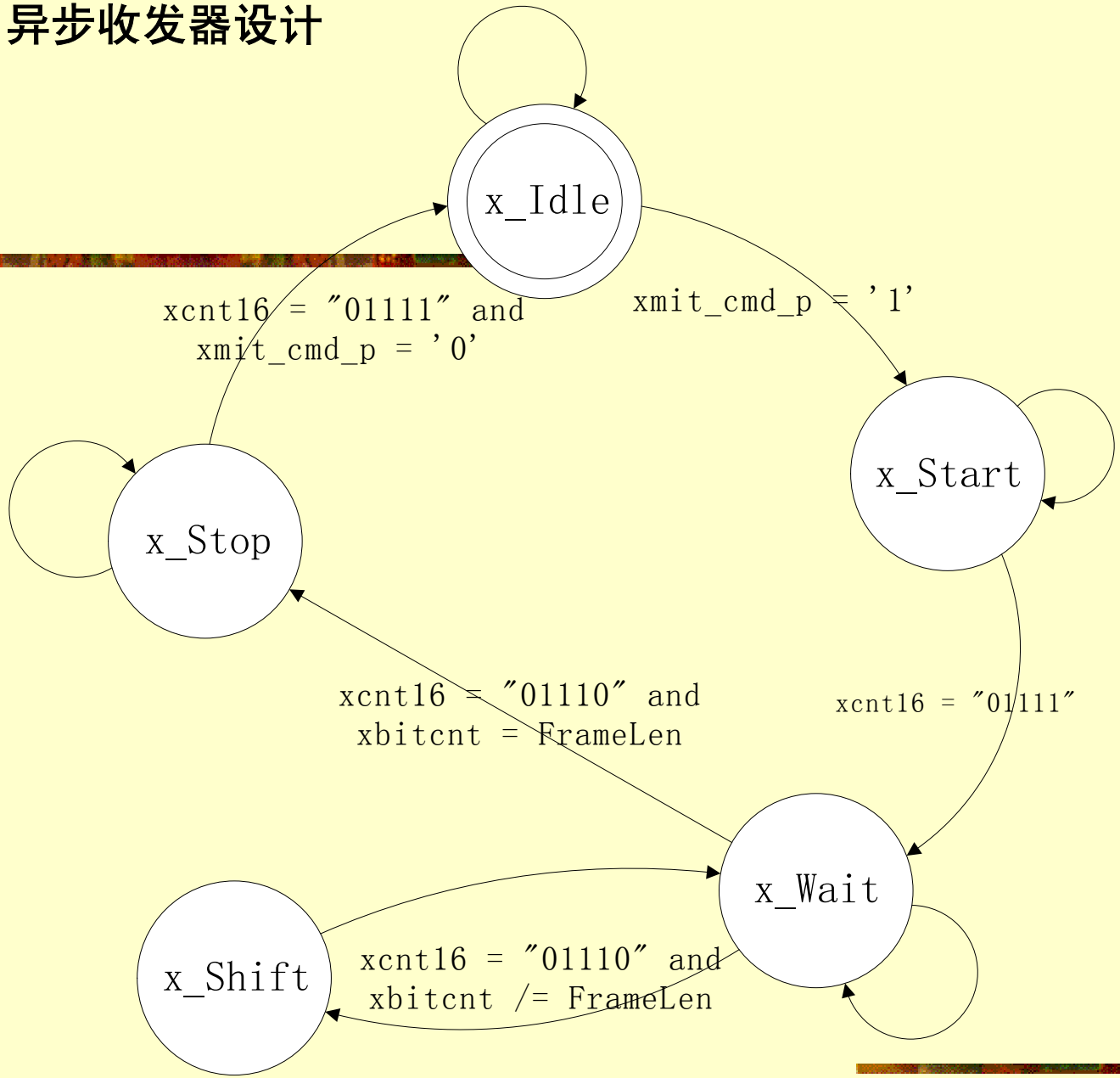


图8-21 UART发送状态机

实 验 与 设 计

8-5. 通用异步收发器设计

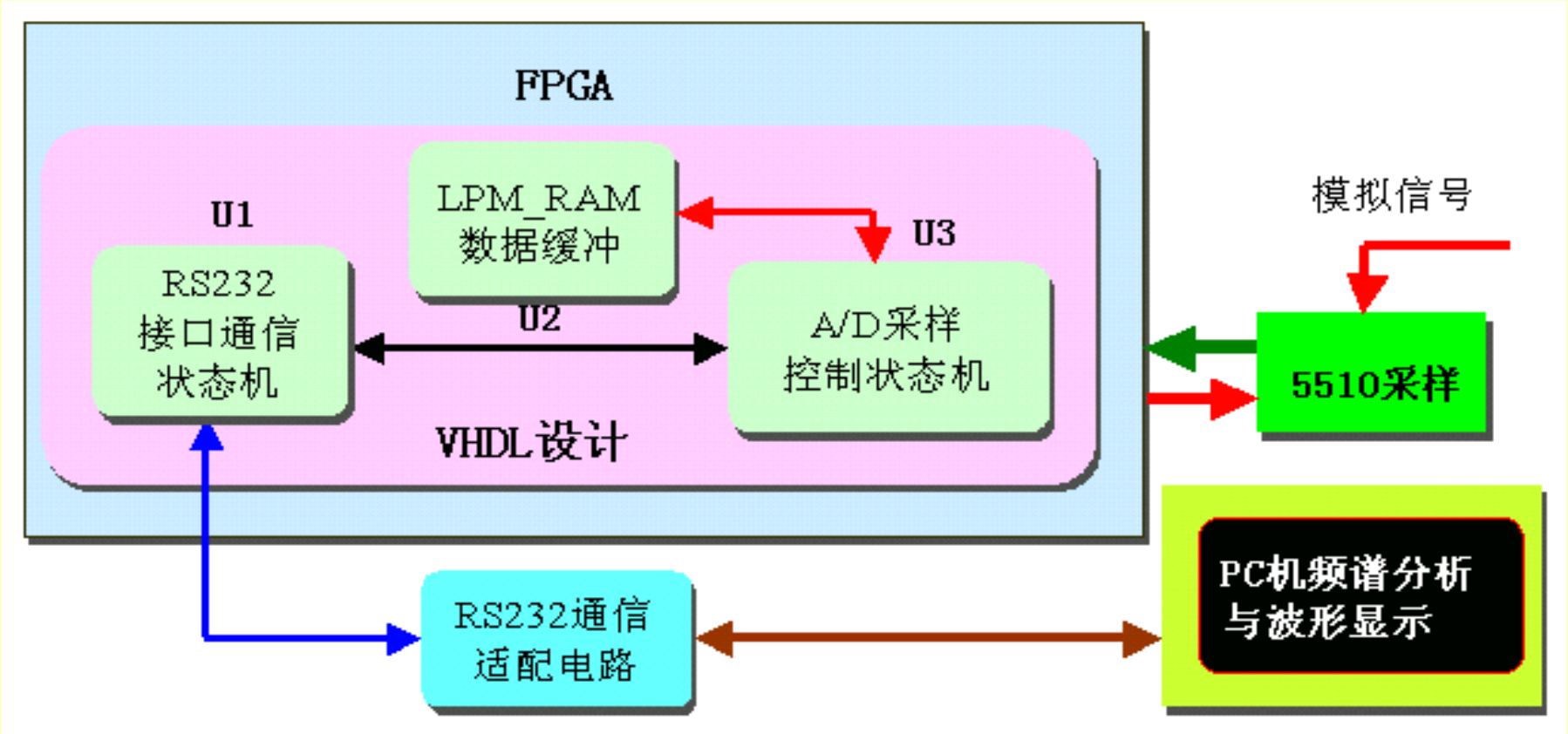


图8-22信号采集与频谱分析电路模块图

8-5. 通用异步收发器设计

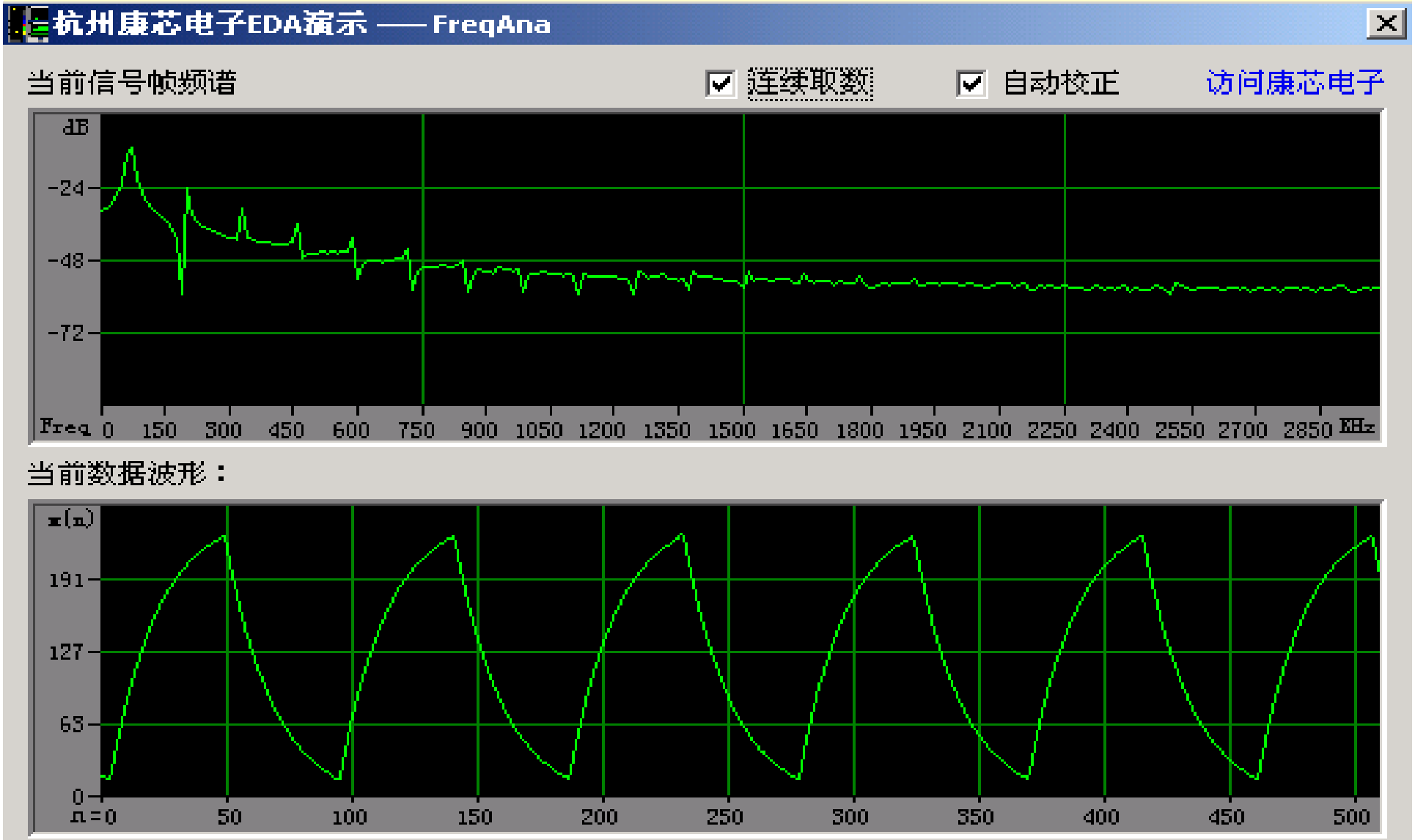


图8-23 采集的65536Hz模拟信号PC机显示

8-5. 通用异步收发器设计

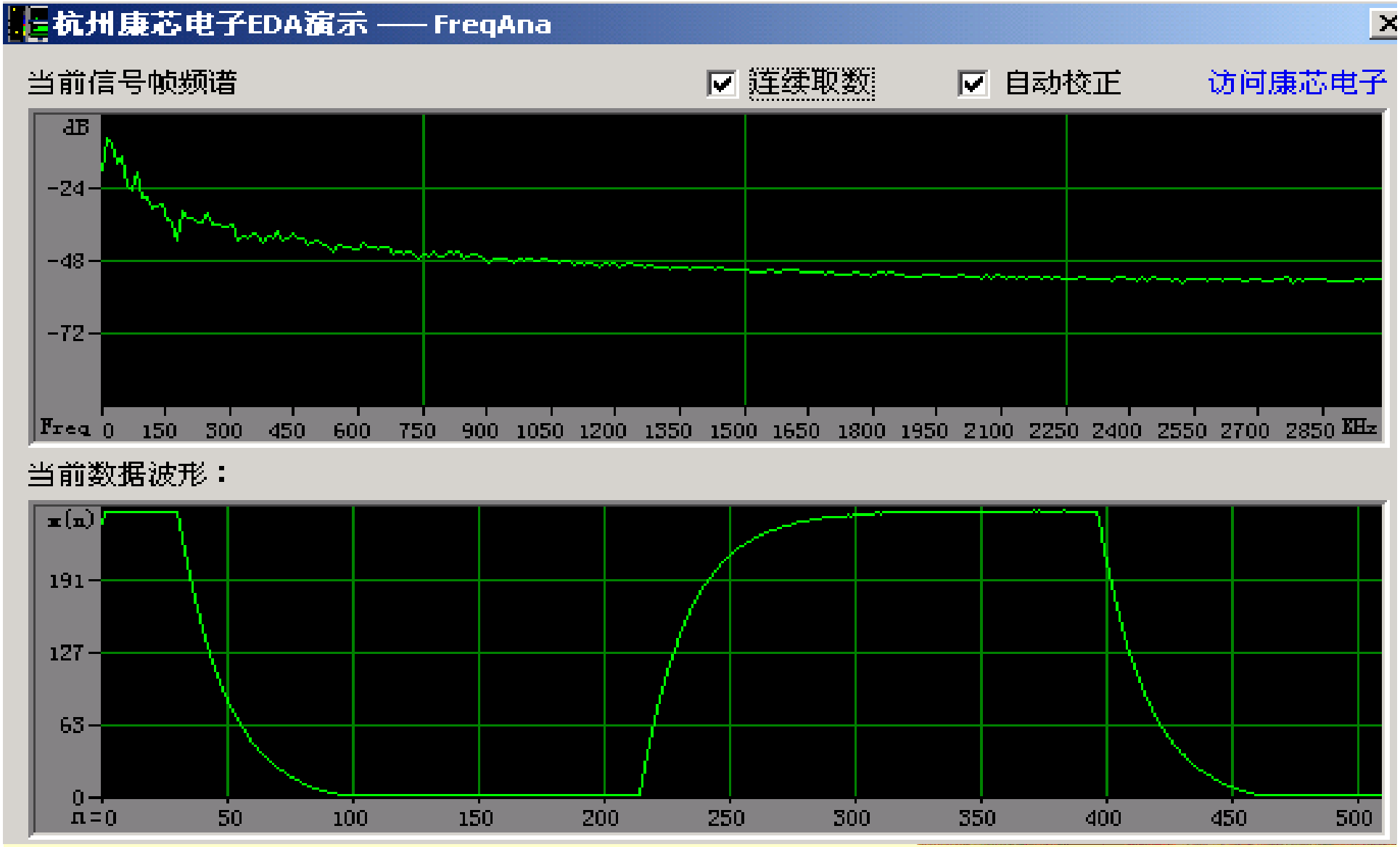


图8-24 采集的16384Hz模拟信号PC机显示

8-5. 通用异步收发器设计

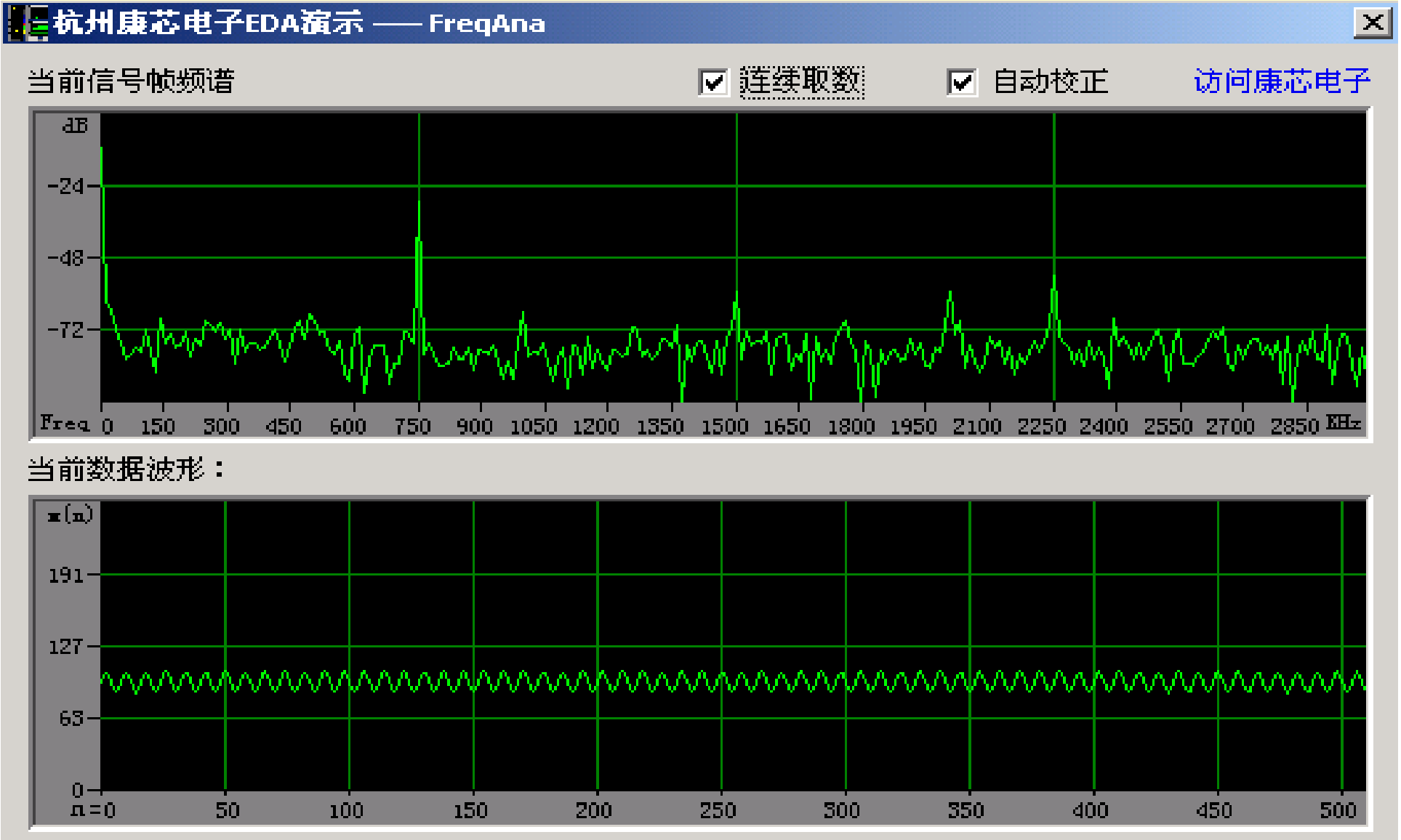


图8-23 采集的65536Hz模拟信号PC机显示

实 验 与 设 计

8-5. 通用异步收发器设计

(3) 实验内容：完成已有设计的验证性实验：将RS232通信线的一头接GW48系统，另一头接PC机的串行1口（COM1口）；将GW48系统右侧的开关向左拨“TO FPGA”，以便使FPGA直接与PC机的RS232通信接口相连；使clock0输入13MHz频率；下载SPECTREM中的ADSUART.SOF，到FPGA中；用模式键选模式“5”，再按一次右侧的复位键，按键1，键2，使其输出高电平；进入“README”目录，运行（双击）并安装分析软件：FASetup.exe ；将模拟信号输入ADDA适配板上的模拟信号输入端口“AIN”。

8-5. 通用异步收发器设计

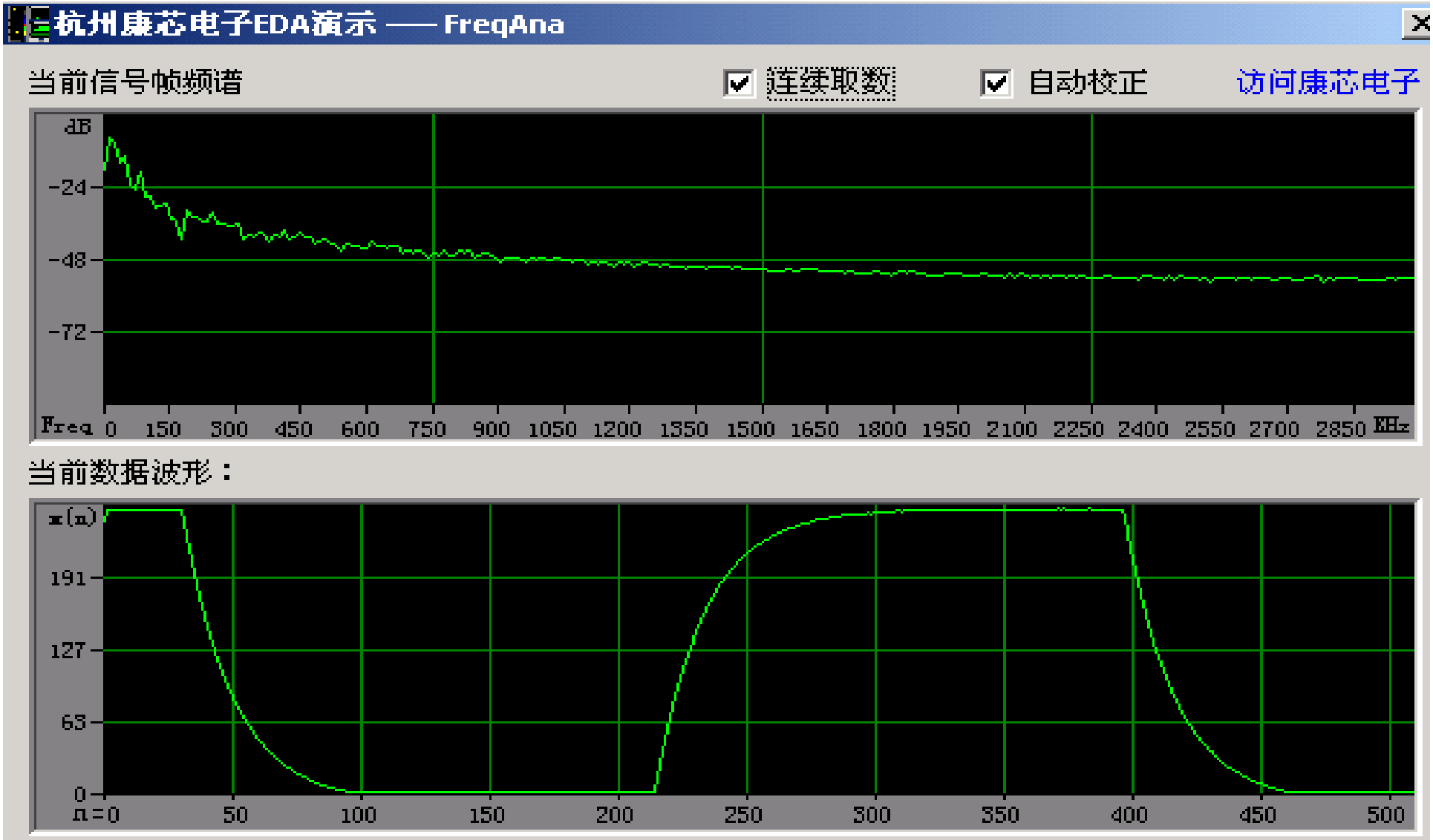


图8-24 采集的16384Hz模拟信号PC机显示

8-5. 通用异步收发器设计

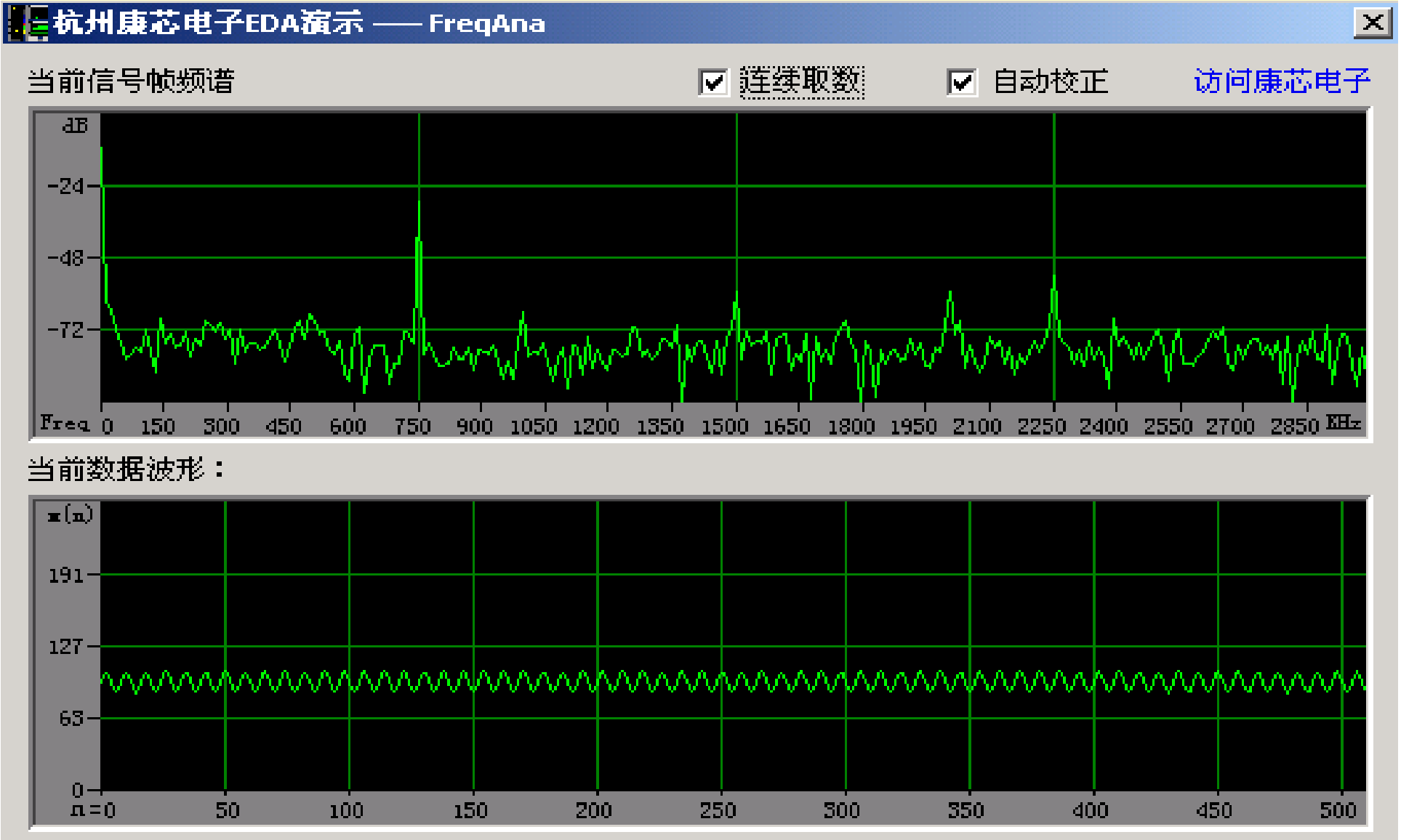


图8-25 采集的750KHz模拟信号PC机显示

EDA 技术实用教程

第 7 章

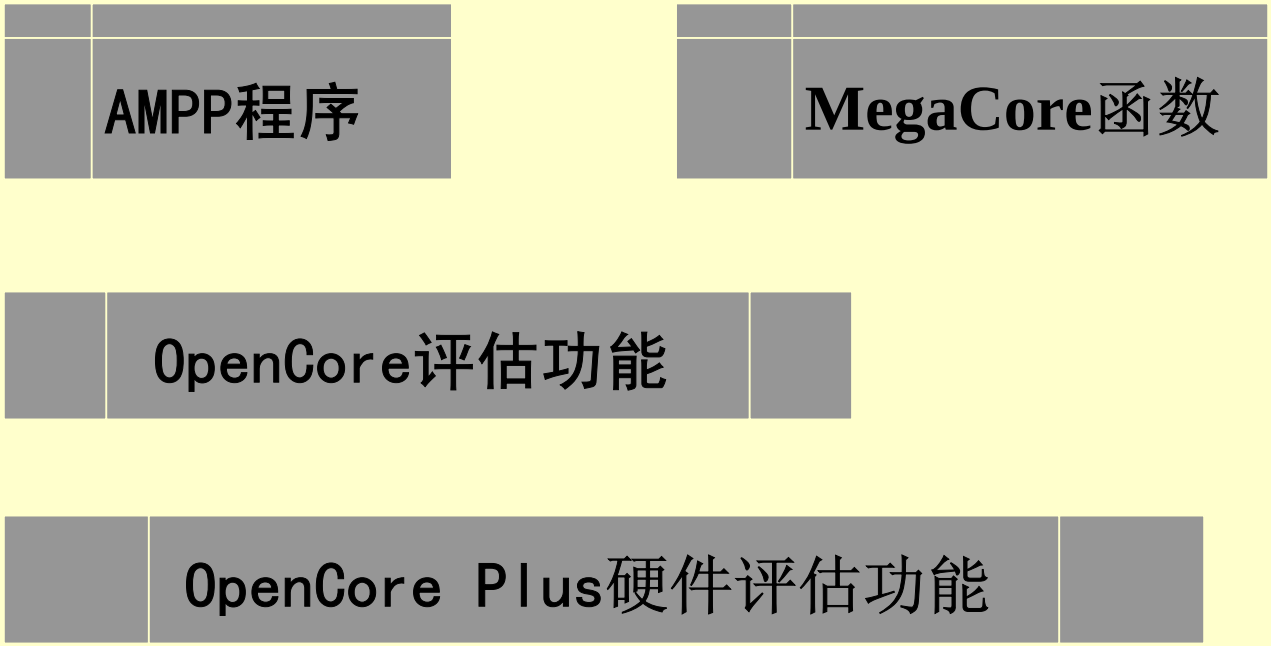
宏功能模块与IP应用

7.1 宏功能模块概述

算术组件	累加器、加法器、乘法器和LPM算术函数
门电路	多路复用器和LPM门函数
I/O组件	时钟数据恢复(CDR)、锁相环(PLL)、双数据速率(DDR)、千兆位收发器块(GXB)、LVDS接收器和发送器、PLL重新配置和远程更新宏功能模块
存储器编译器	FIFO Partitioner、RAM和ROM宏功能模块
存储组件	存储器、移位寄存器宏模块和LPM存储器函数

7.1 宏功能模块概述

7.1.1 知识产权核的应用



7.1 宏功能模块概述

7.1.2 使用MegaWizard Plug-In Manager

<输出文件>.bsf ： Block Editor中使用的宏功能模块的符号（元件）。

<输出文件>.cmp： 组件申明文件。

<输出文件>.inc ： 宏功能模块包装文件中模块的AHDL包含文件。

<输出文件>.tdf： 要在AHDL设计中实例化的宏功能模块包装文件。

<输出文件>.vhd： 要在VHDL设计中实例化的宏功能模块包装文件。

<输出文件>.v ： 要在VerilogHDL设计中实例化的宏功能模块包装文件。

<输出文件>_bb.v： VerilogHDL设计所用宏功能模块包装文件中模块的空体或black-box申明，用于在使用EDA 综合工具时指定端口方向。

<输出文件>_inst.tdf： 宏功能模块包装文件中子设计的AHDL例化示例。

<输出文件>_inst.vhd： 宏功能模块包装文件中实体的VHDL例化示例。

<输出文件>_inst.v ： 宏功能模块包装文件中模块的VerilogHDL例化示例。

7.1 宏功能模块概述

7.1.3 在QuartusII中对宏功能模块进行例化

1、在VerilogHDL和VHDL中例化

2、使用端口和参数定义

3、使用端口和参数定义生成宏功能模块

计数器	乘-累加器和乘-加法器
加法/减法器	RAM
乘法器	移位寄存器

7.2 宏模块应用实例

7.2.1 工作原理

$$f = f_0 / 64$$

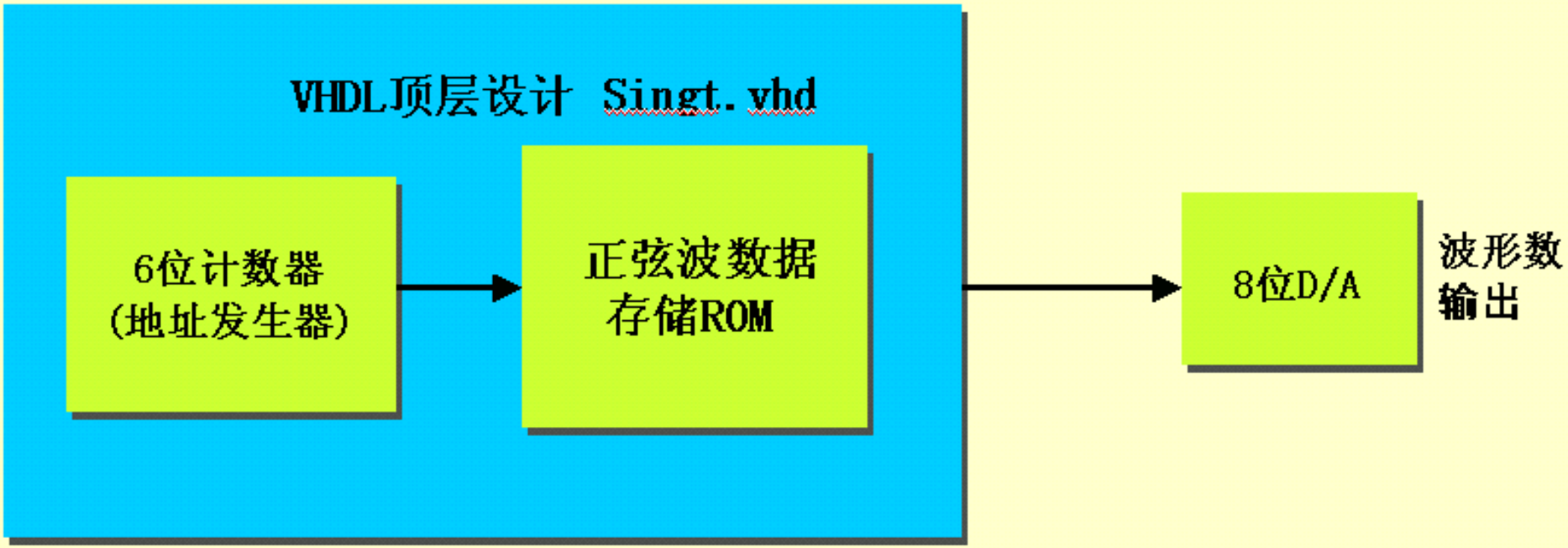


图7-1 正弦信号发生器结构框图

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

1. 建立.mif格式文件

【例7-1】

```
WIDTH = 8;
DEPTH = 64;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
0        :       FF;
1        :       FE;
2        :       FC;
3        :       F9;
4        :       F5;
...（数据略去）
3D        :        FC;
3E        :        FE;
3F        :        FF;
END;
```

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

1. 建立.mif格式文件

【例7-2】

```
#include <stdio.h>
#include "math.h"
main()
{int i;float s;
for(i=0;i<1024;i++)
    { s = sin(atan(1)*8*i/1024);
      printf("%d : %d;\n",i,(int)((s+1)*1023/2));
    }
}
```

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

2. 建立.hex格式文件

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	255	254	252	249	245	239	233	225
8	217	207	197	186	174	162	150	137
16	124	112	99	87	75	64	53	43
24	34	26	19	13	8	4	1	0
32	0	1	4	8	13	19	26	34
40	43	53	64	75	87	99	112	124
48	137	150	162	174	186	197	207	217
56	225	233	239	245	249	252	254	255

图7-2 将波形数据填入mif文件表中

2. 建立.hex格式文件

图7-3 ASM格式建hex文件

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

2. 建立.hex格式文件

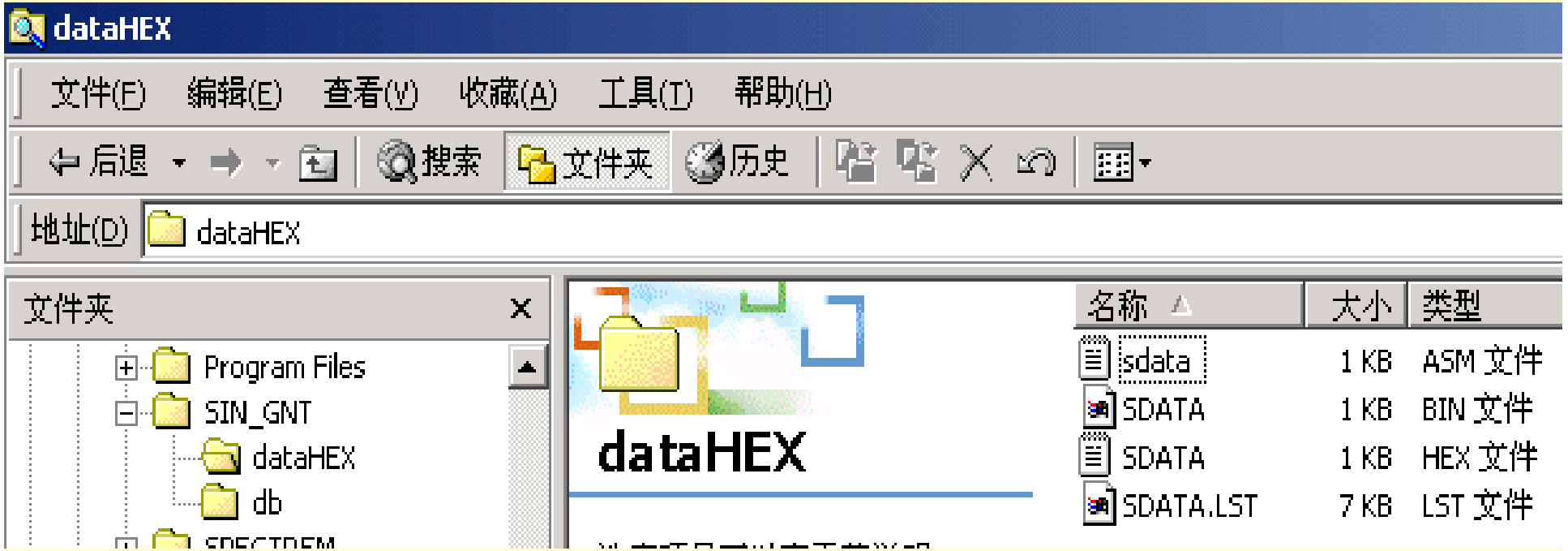


图7-4 sdata.hex文件的放置路径

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.3 定制LPM_ROM元件

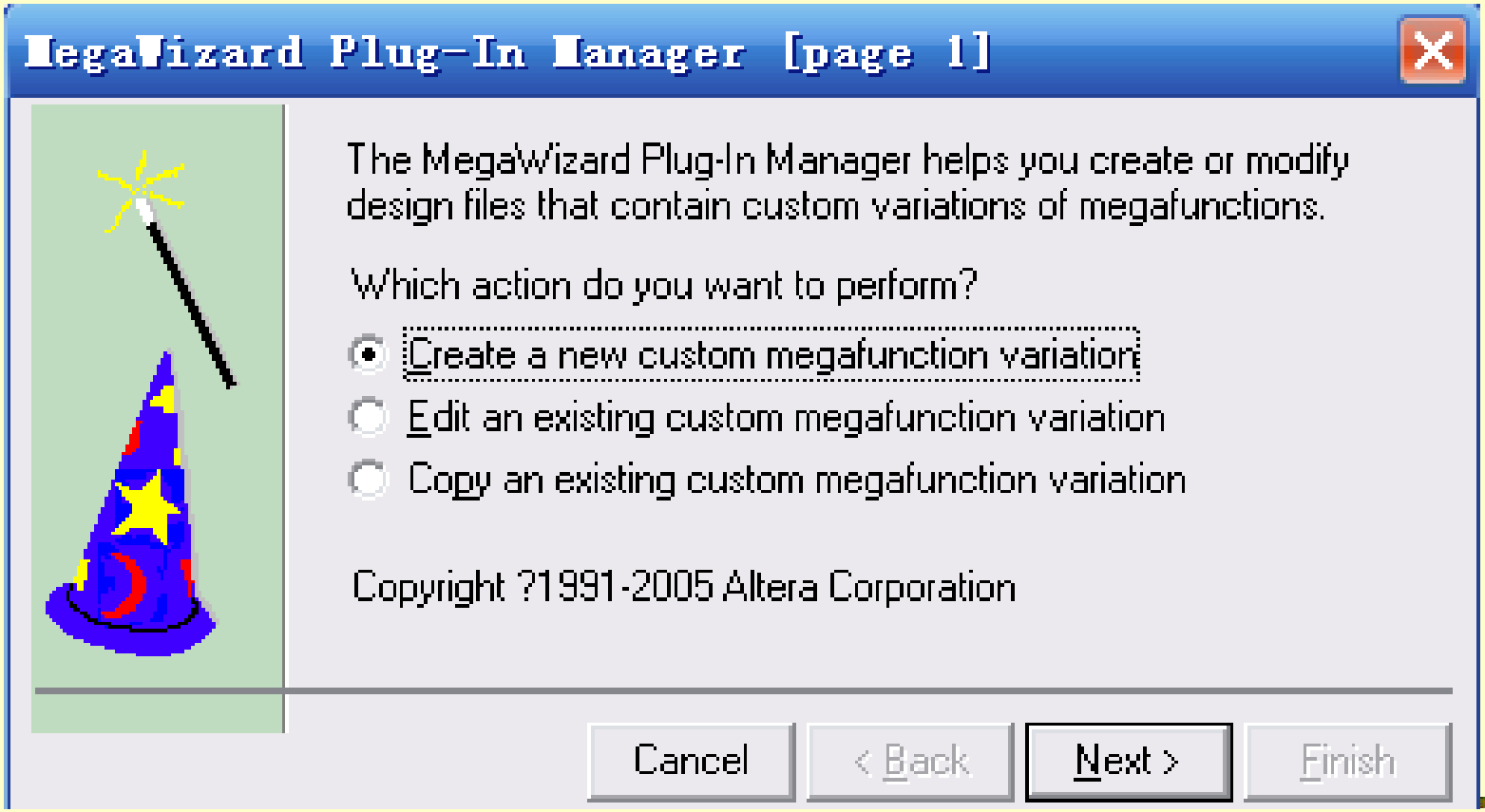


图7-5 定制新的宏功能块

康芯科技

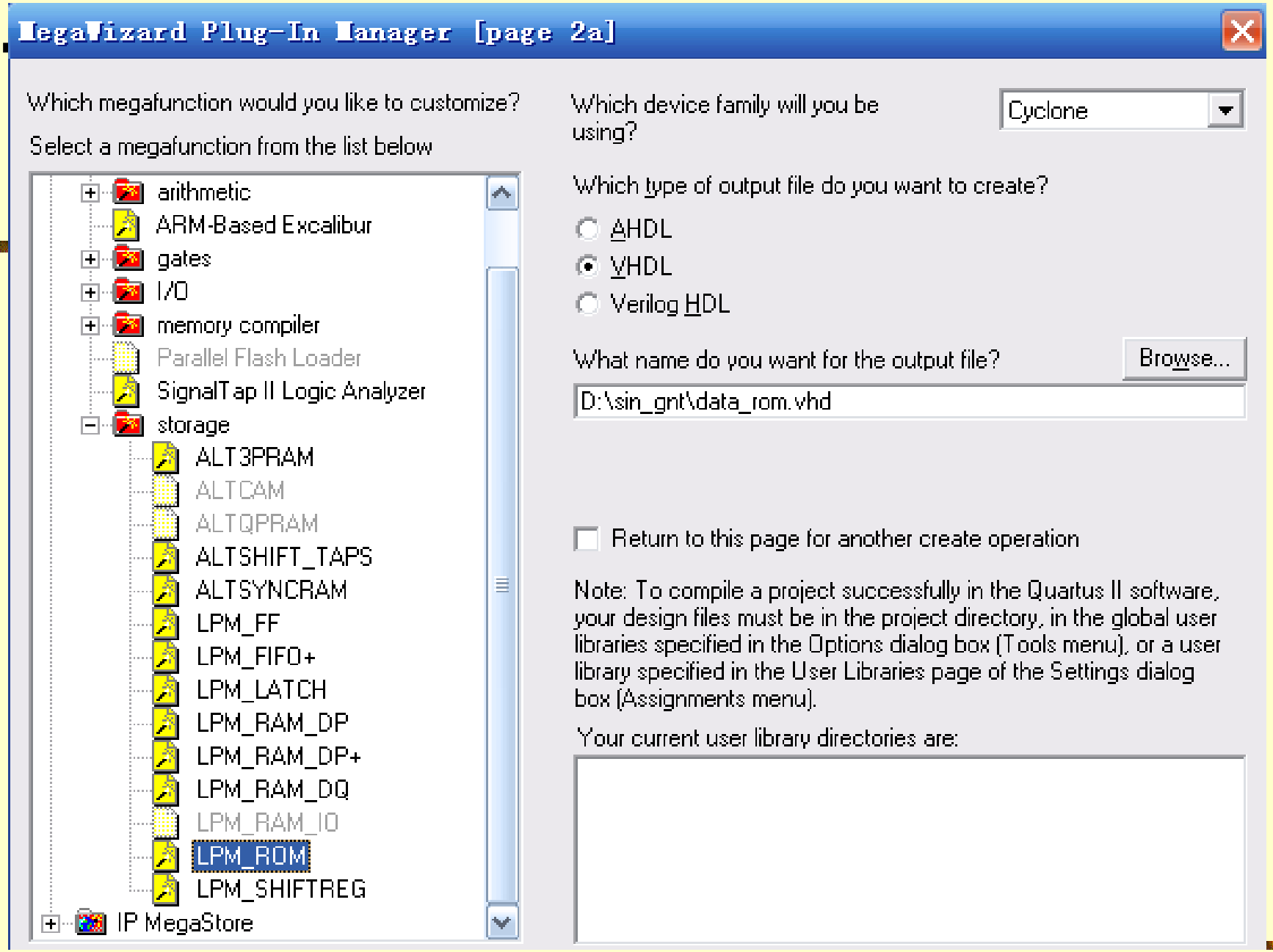


图7-6 LPM宏功能块设定

KX 康芯科技

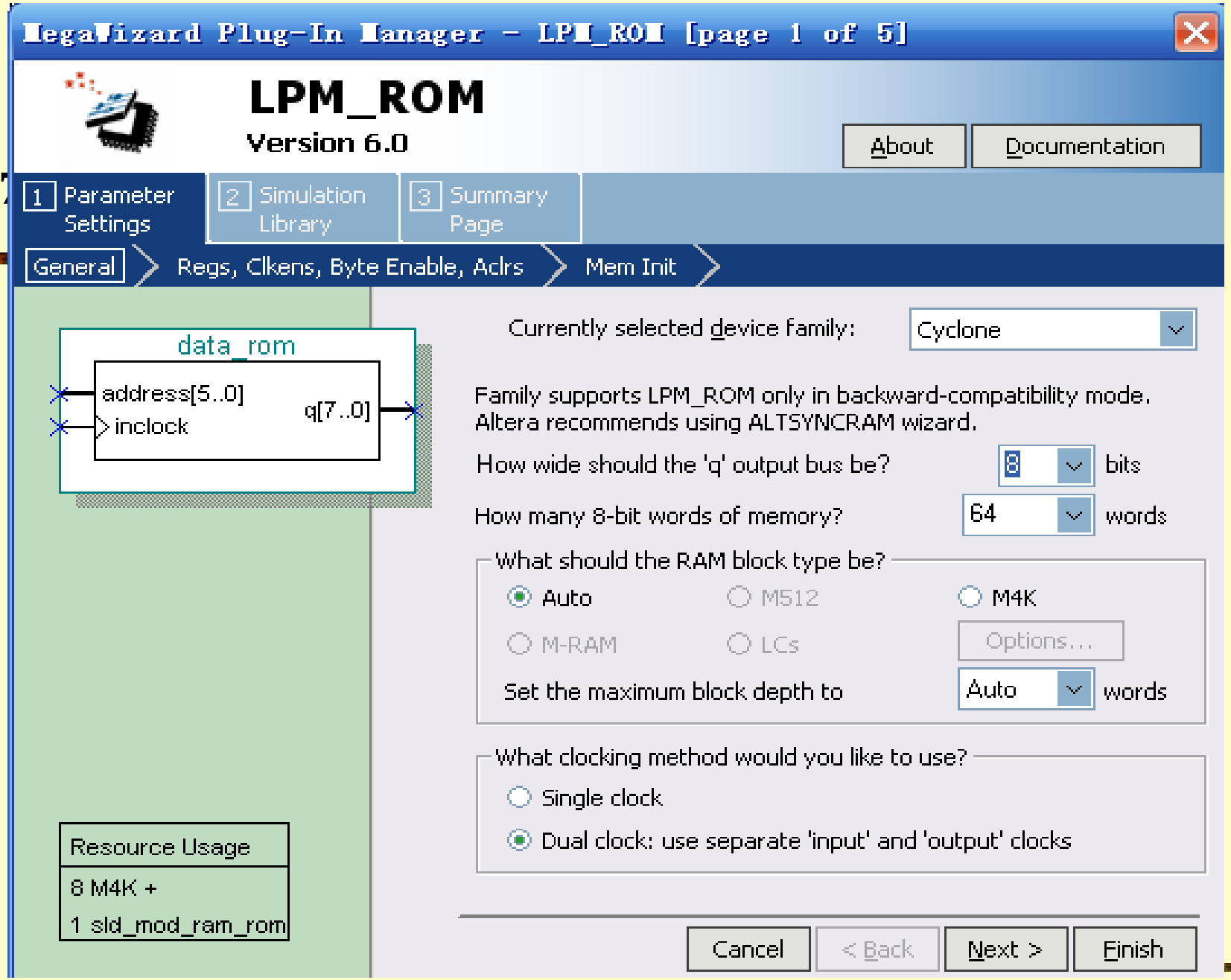


图7-7 选择data_rom模块数据线和地址线宽度

KX 康芯科技

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.3 定制LPM_ROM元件

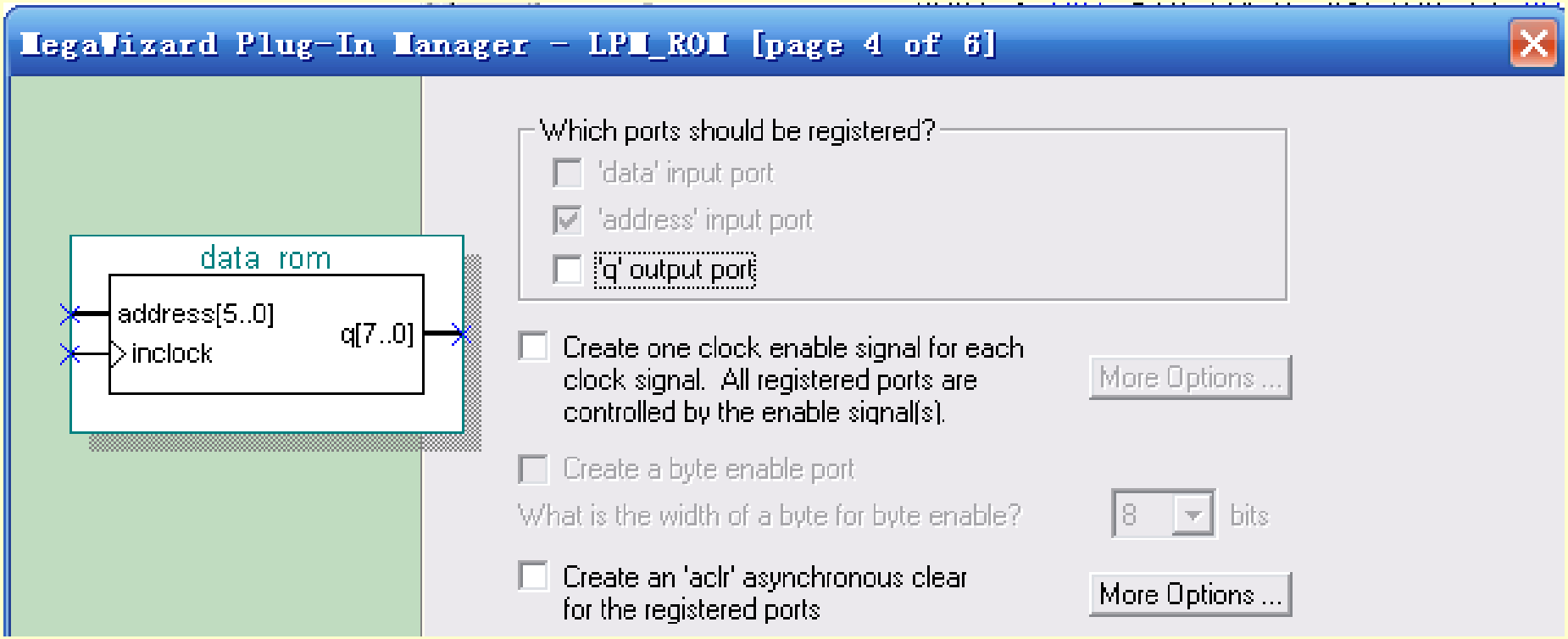


图7-8 选择地址锁存信号inclock

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.3 定制LPM_ROM元件

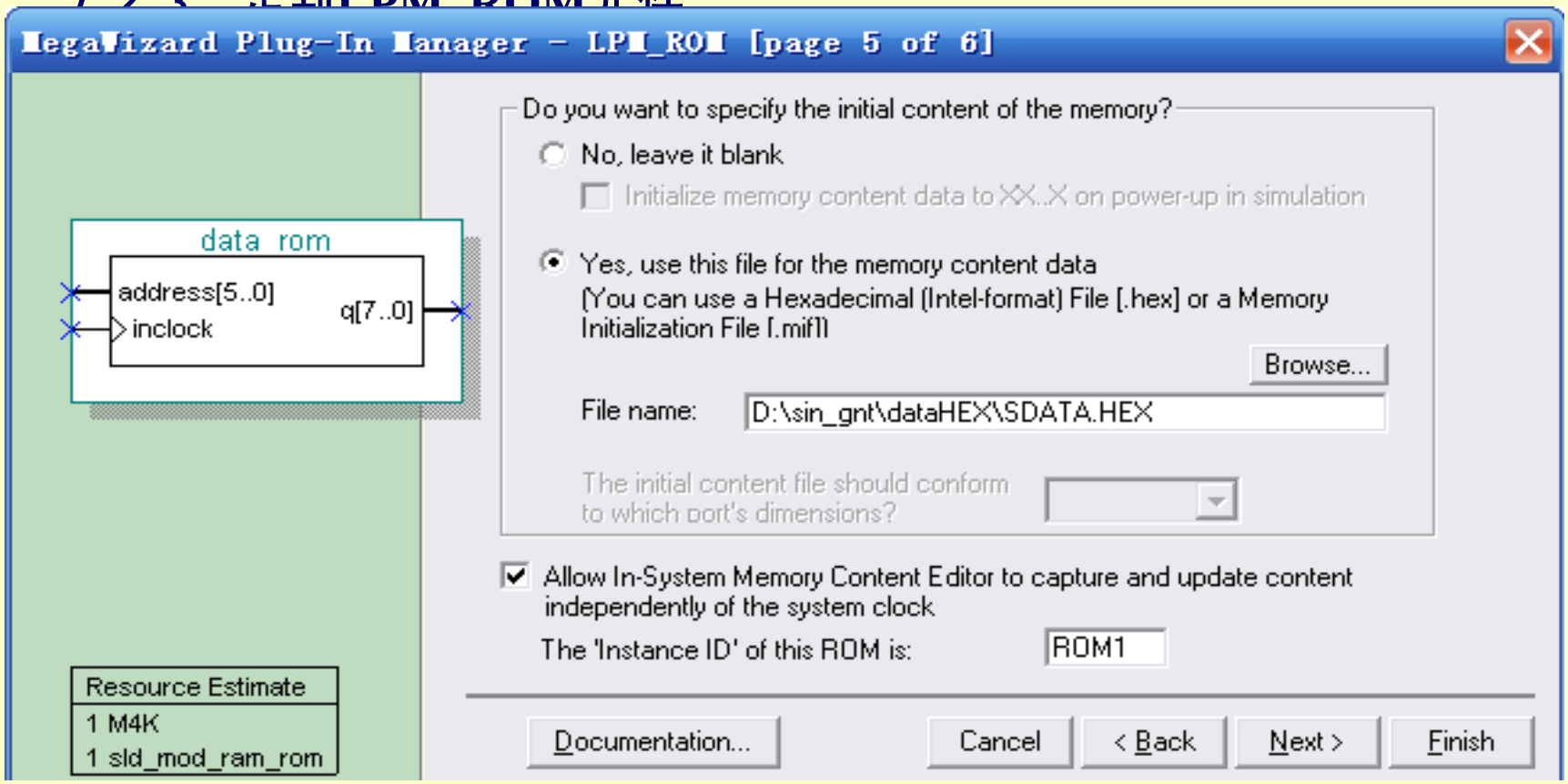


图7-9 调入ROM初始化数据文件并选择在系统读写功能

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.3 定制LPM_ROM元件

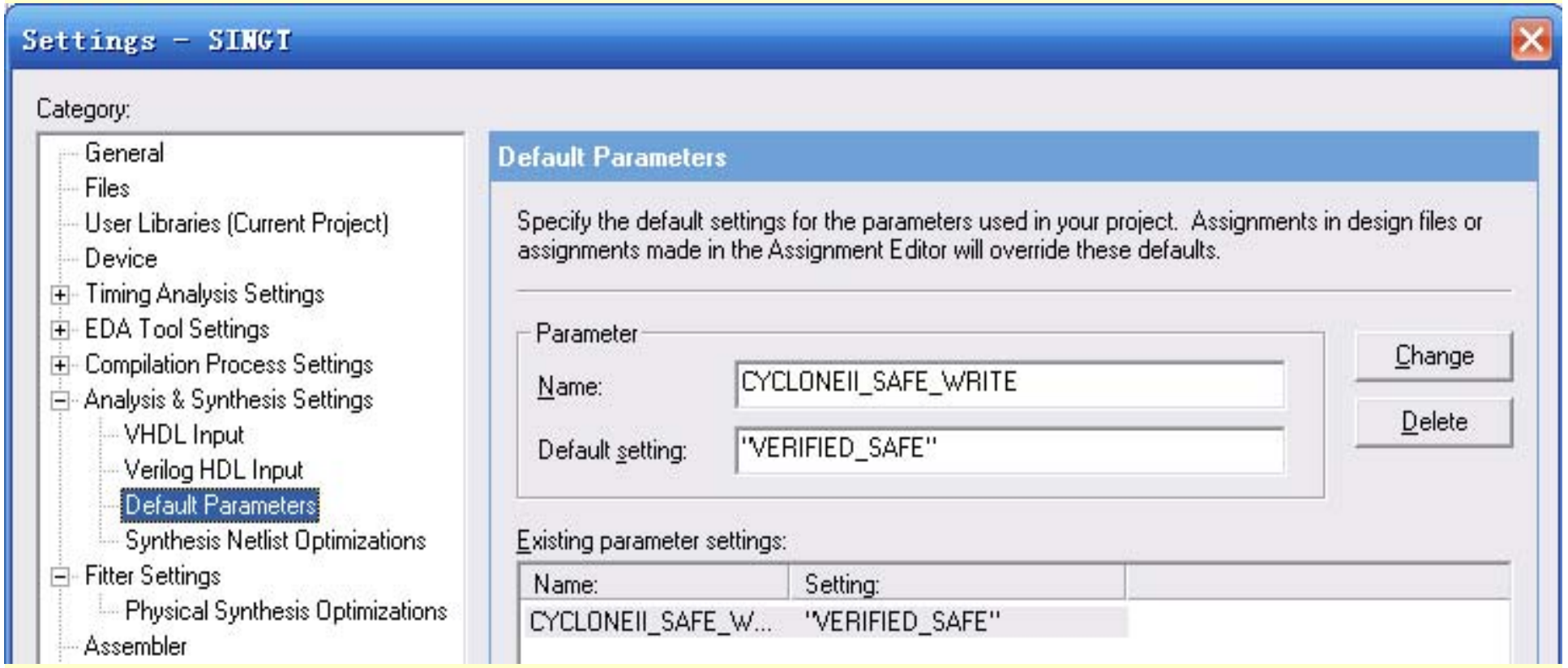


图7-10 LPM_ROM设计完成

【例7-3】

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;    --使用宏功能库中的所有元件
ENTITY data_rom IS
       PORT (address    : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
               inclock   : IN STD_LOGIC ;
              q          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
END data_rom;
ARCHITECTURE SYN OF data_rom IS
       SIGNAL sub_wire0     : STD_LOGIC_VECTOR (7 DOWNTO 0);
       COMPONENT altsyncram      --例化altsyncram元件，调用了LPM模块
altsyncram
       GENERIC (                                  --参数传递语句
              intended_device_family     : STRING;   --类属参量数据类
型定义
              width_a                : NATURAL;
              widthad_a              : NATURAL;
              numwords_a             : NATURAL;
              operation_mode               : STRING;
              outdata_reg_a          : STRING;
              address_aclr_a               : STRING;
```

接下页

```
                outdata_aclr_a                  : STRING;
                width_byteena_a                 : NATURAL;
                init_file               : STRING;
                lpm_hint                : STRING;
                lpm_type                : STRING      );
        PORT ( clock0 : IN STD_LOGIC ;        --altsyncram元件接口声明
                address_a     : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                    q_a       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
        END COMPONENT;
BEGIN
        q    <= sub_wire0(7 DOWNTO 0);
        altsyncram_component : altsyncram
        GENERIC MAP ( intended_device_family => "Cyclone",  --参数
传递映射
                width_a => 8,                           --数据线宽度8
                widthad_a => 6,                         --地址线宽度6
                numwords_a => 64,                       --数据数量64
                operation_mode => "ROM",                --LPM模式ROM
                outdata_reg_a => "UNREGISTERED",  --输出无锁存
                address_aclr_a => "NONE",               --无异步地址清0
                outdata_aclr_a => "NONE",               --无输出锁存异步清0
                width_byteena_a => 1,             -- byteena_a输入口宽度1
                init_file => "./dataHEX/SDATA.hex",  --ROM初始化数据文
件，此处已修改过
```

接下页

接上页

```
                    lpm_hint => "ENABLE_RUNTIME_MOD=YES,
INSTANCE_NAME=NONE",
              lpm_type => "altsyncram" )             --LPM类型
       PORT MAP ( clock0 => inclock, address_a => address,q_a =>
sub_wire0 );
```

7.2.4 完成顶层设计

【例7-4】正弦信号发生器顶层设计

```
LIBRARY IEEE;  --正弦信号发生器源文件
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SINGT IS
    PORT ( CLK  : IN STD_LOGIC;                          --信号源时钟
           DOUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );--8位波形数据输出
END;
ARCHITECTURE DACC OF SINGT IS
COMPONENT data_rom    --调用波形数据存储器LPM_ROM文件：data_rom.vhd声明
   PORT(address : IN STD_LOGIC_VECTOR (5 DOWNTO 0);--6位地址信号
        inclock : IN STD_LOGIC ;--地址锁存时钟
              q : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)    );
END COMPONENT;
     SIGNAL Q1 : STD_LOGIC_VECTOR (5 DOWNTO 0); --设定内部节点作为地址计数器
    BEGIN
PROCESS(CLK )                              --LPM_ROM地址发生器进程
    BEGIN
IF CLK'EVENT AND CLK = '1' THEN  Q1<=Q1+1; --Q1作为地址发生器计数器
END IF;
END PROCESS;
u1 : data_rom PORT MAP(address=>Q1, q => DOUT,inclock=>CLK);--例化
END;
```

KX 康芯科技

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.4 完成顶层设计

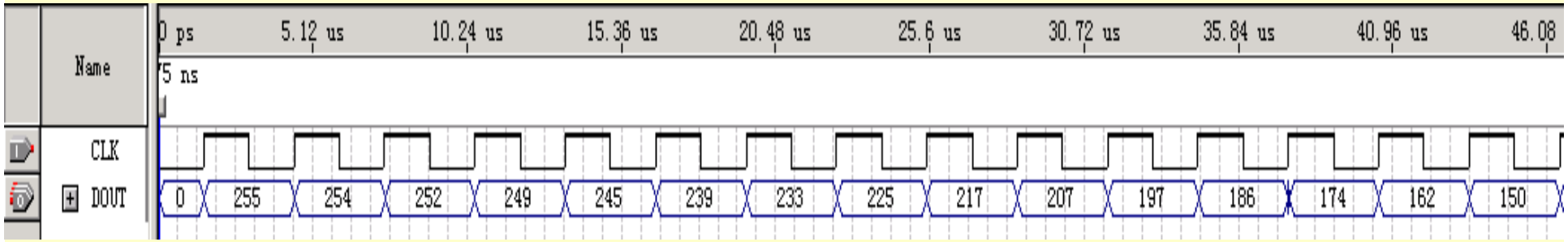


图7-11 仿真波形输出

7.2 宏模块应用实例

7.2.2 定制初始化数据文件

7.2.4 完成顶层设计

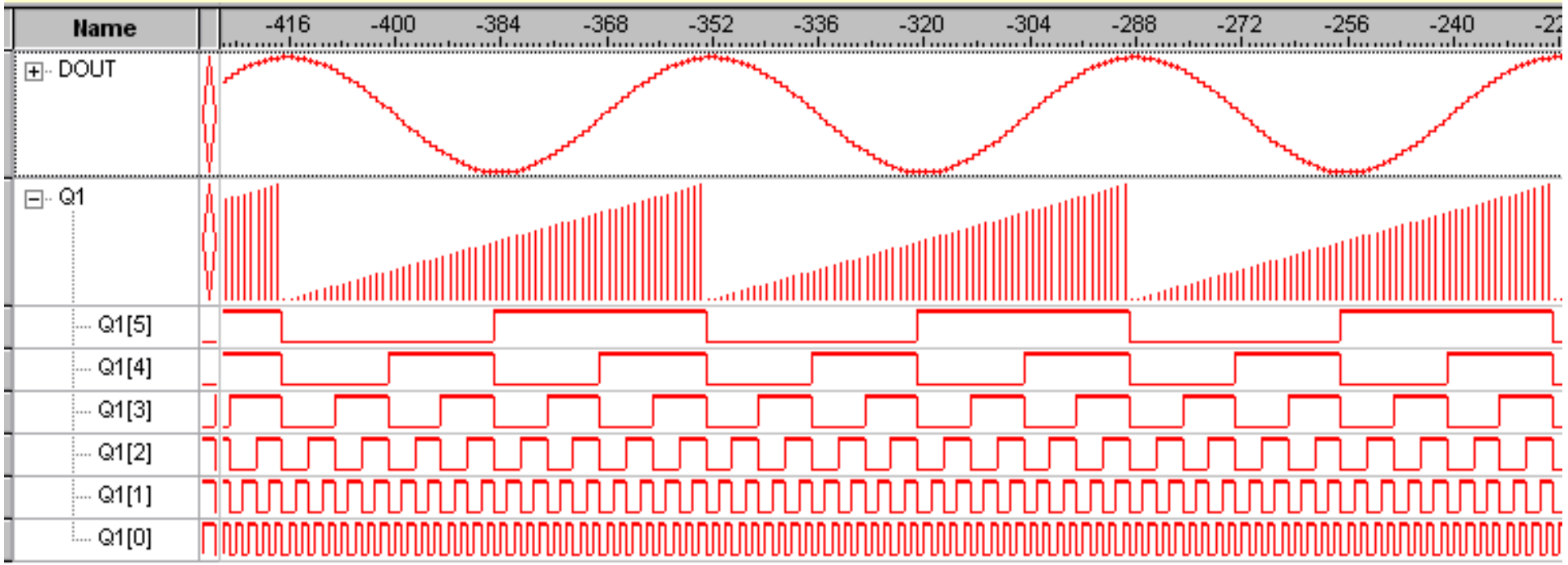


图7-12 嵌入式逻辑分析仪获得的波形

7.3 在系统存储器数据读写编辑器应用

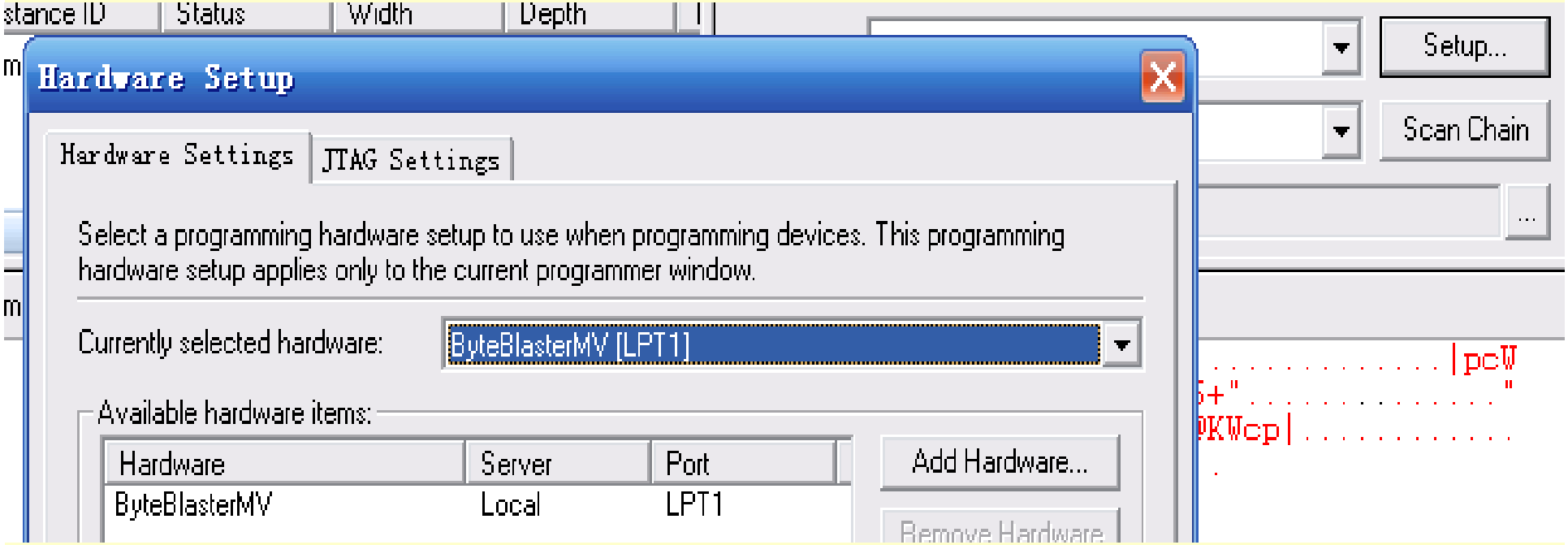


图7-13 In-System Memory Content Editor编辑窗

7.3 在系统存储器数据读写编辑器应用

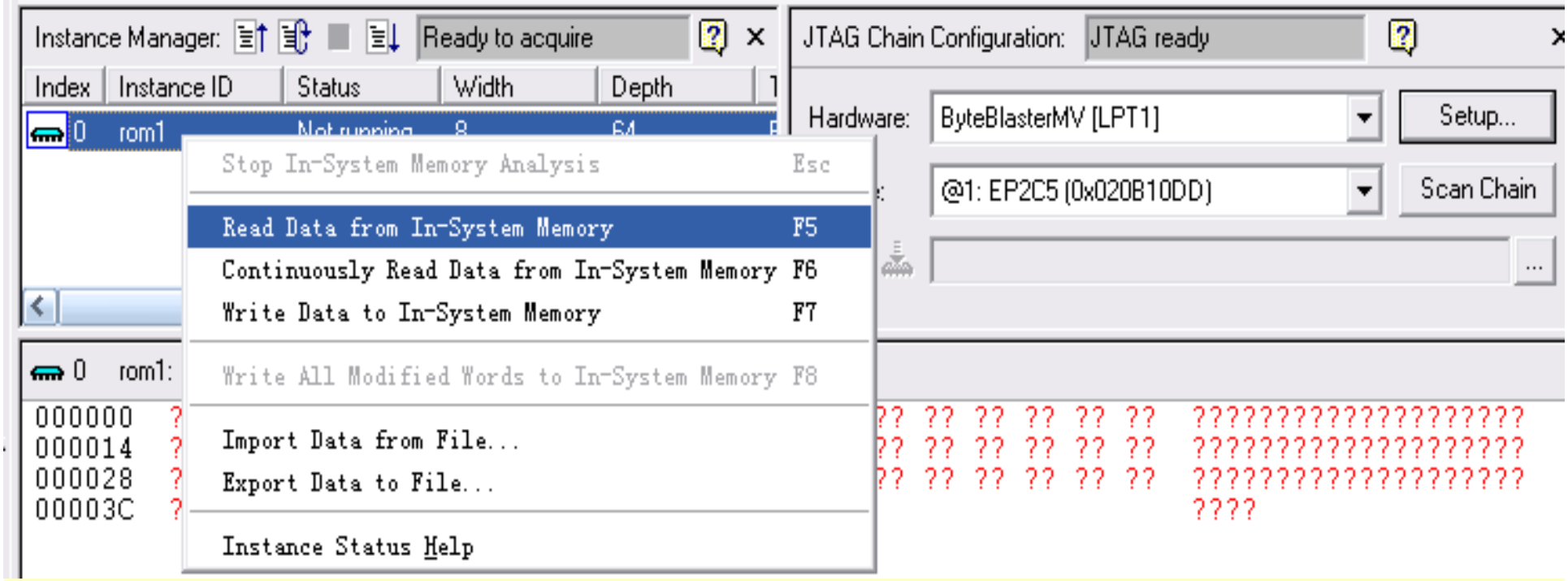


图7-14 与实验系统上的FPGA通信正常情况下的编辑窗界面

7.3 在系统存储器数据读写编辑器应用

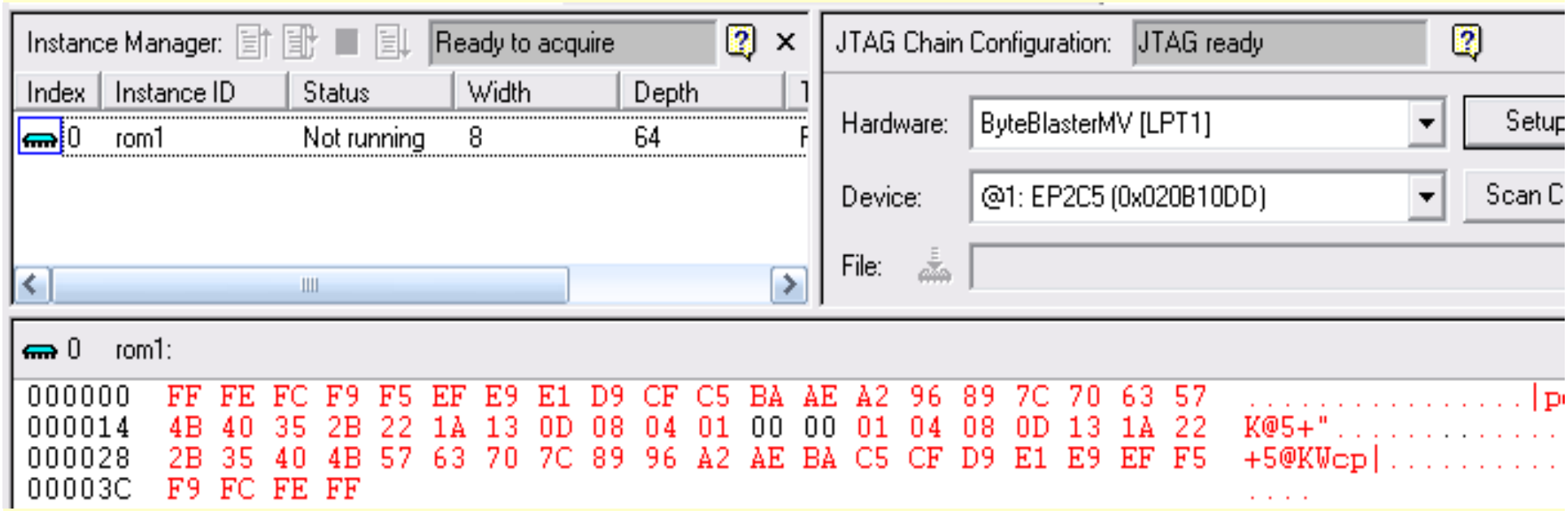


图7-15 从FPGA中的ROM读取波形数据

7.3 在系统存储器数据读写编辑器应用

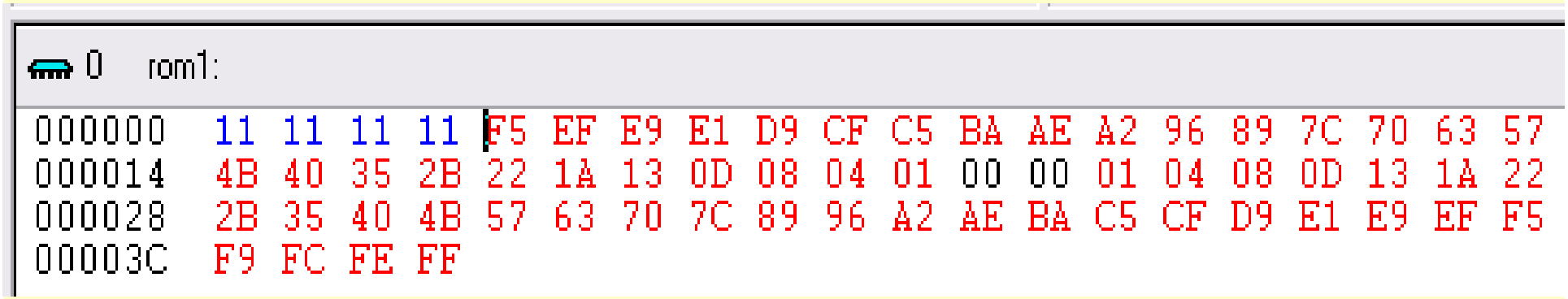


图7-16 编辑波形数据

7.3 在系统存储器数据读写编辑器应用

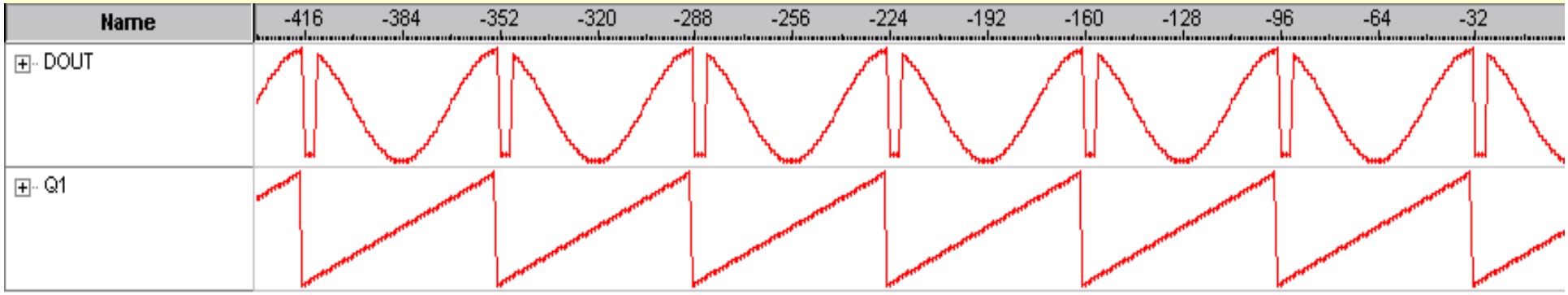


图7-16下载编辑数据后的SignalTap II采样波形

7.4 编辑SignalTapII的触发信号

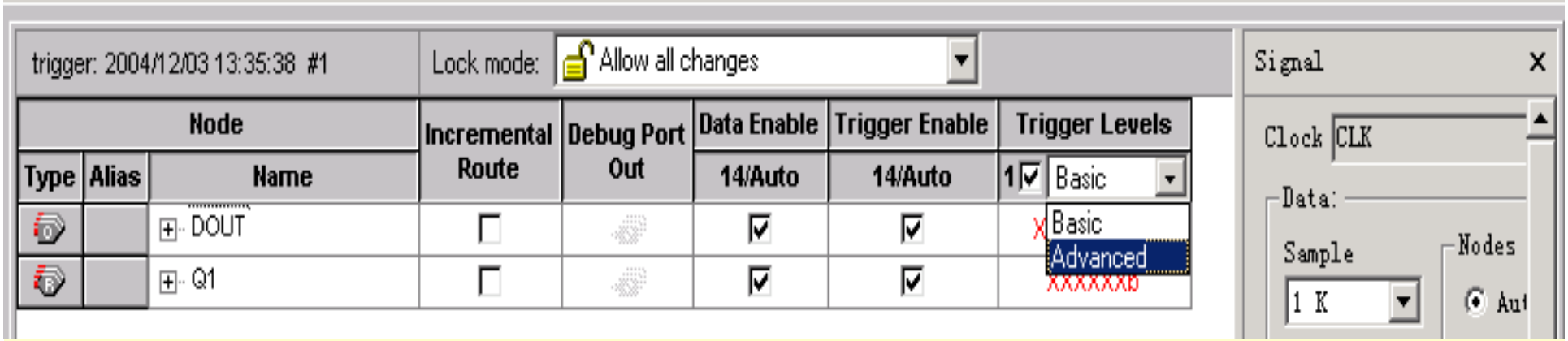


图7-17 选择高级触发条件

7.4 编辑SignalTapII的触发信号

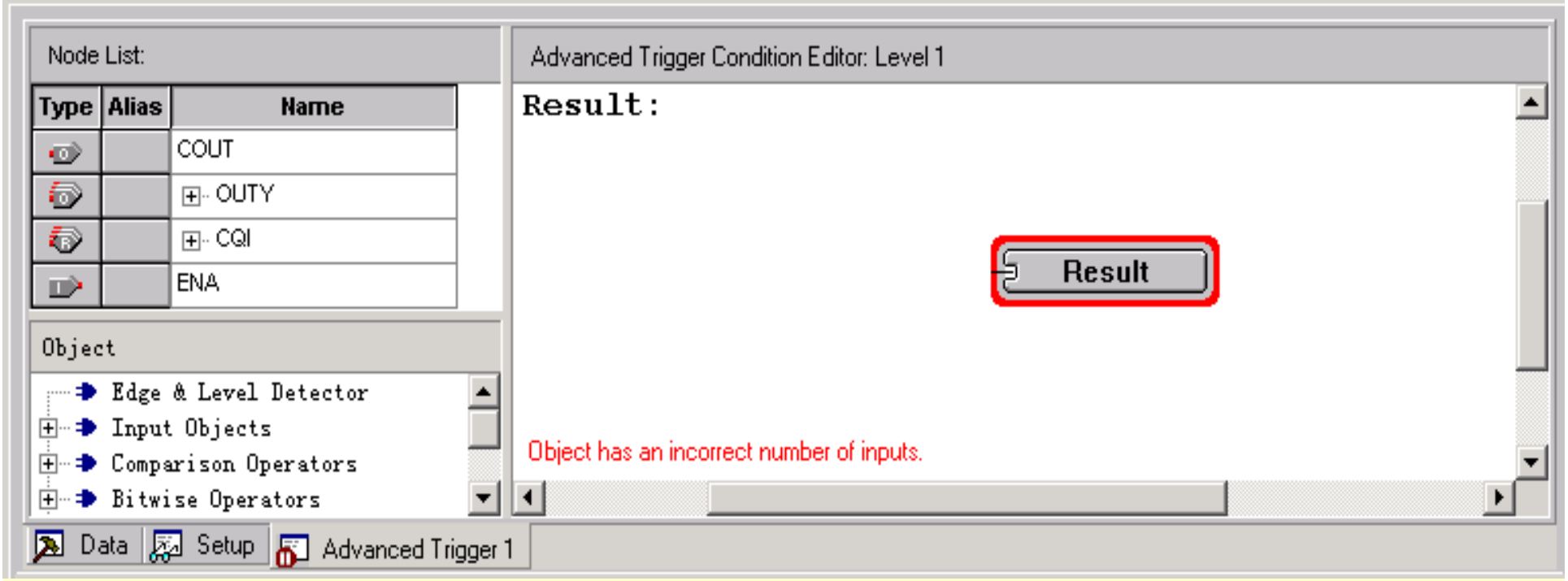


图7-18 进入“触发条件函数编辑”窗口

7.4 编辑SignalTapII的触发信号

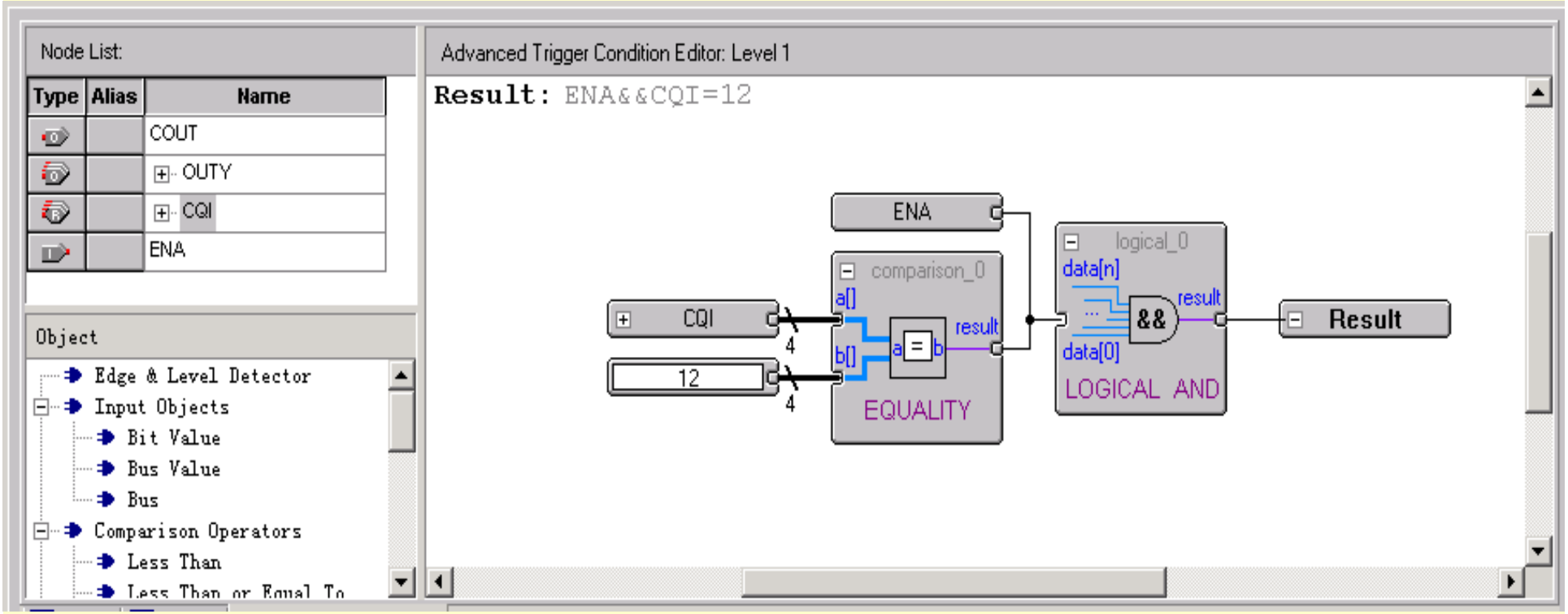


图7-19 编辑触发函数

7.5 其它存储器模块的定制与应用

7.5.1 RAM定制

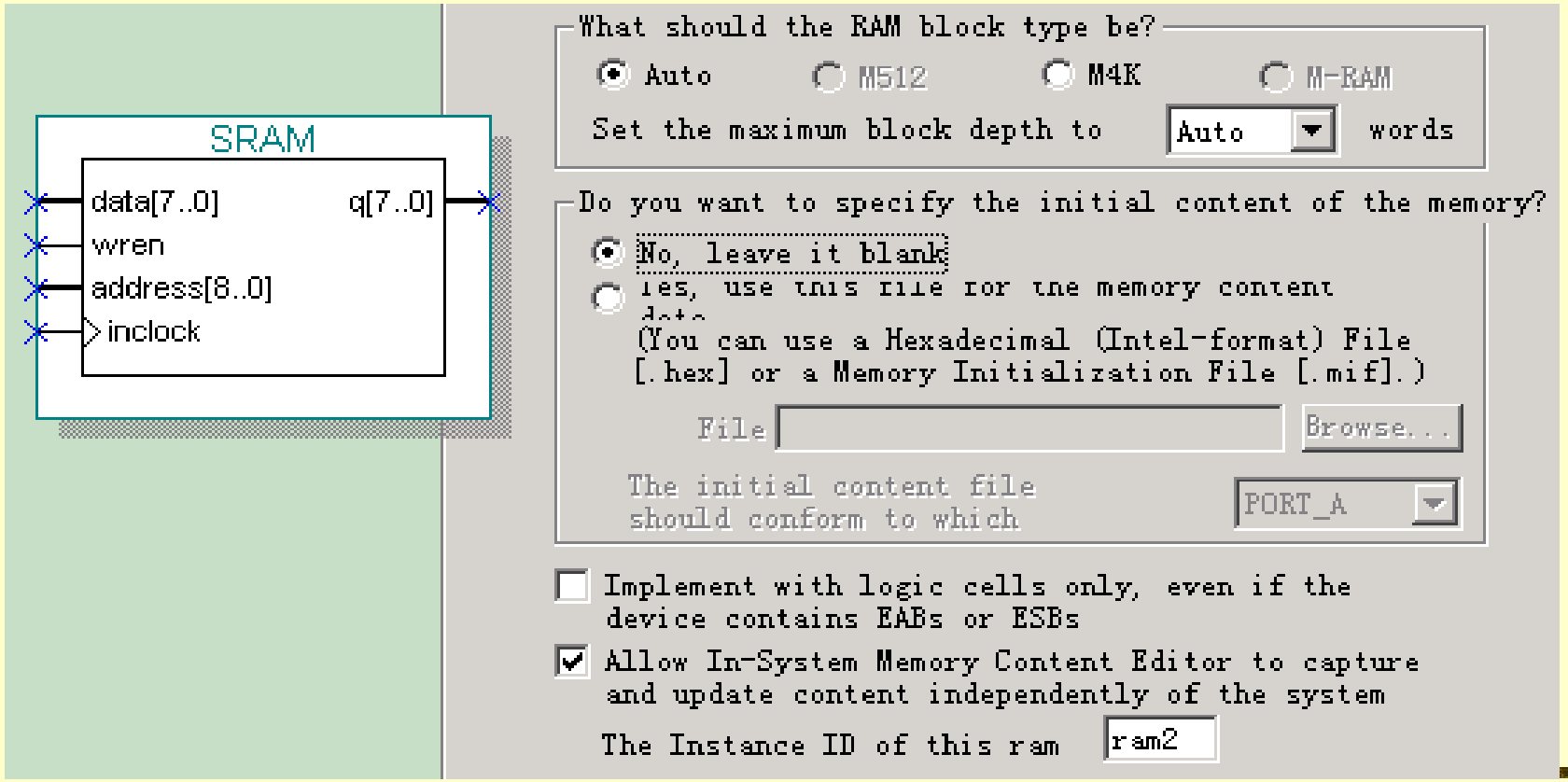


图7-20 编辑定制RAM

7.5 其它存储器模块的定制与应用

7.5.1 RAM定制

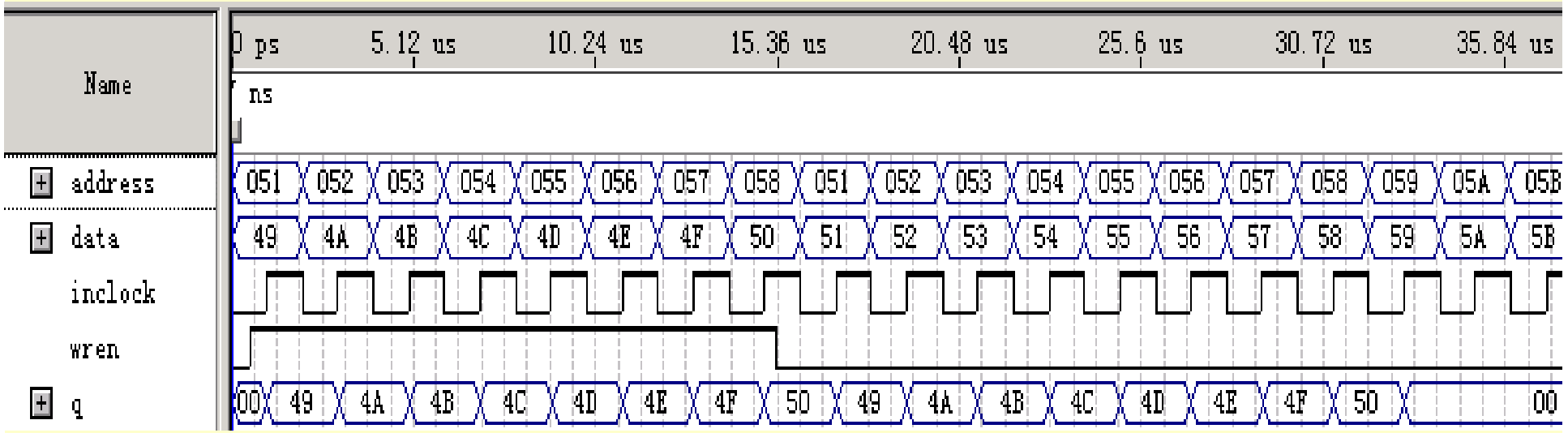


图7-21 LPM_RAM的仿真波形

7.5 其它存储器模块的定制与应用

7.5.2 FIFO定制

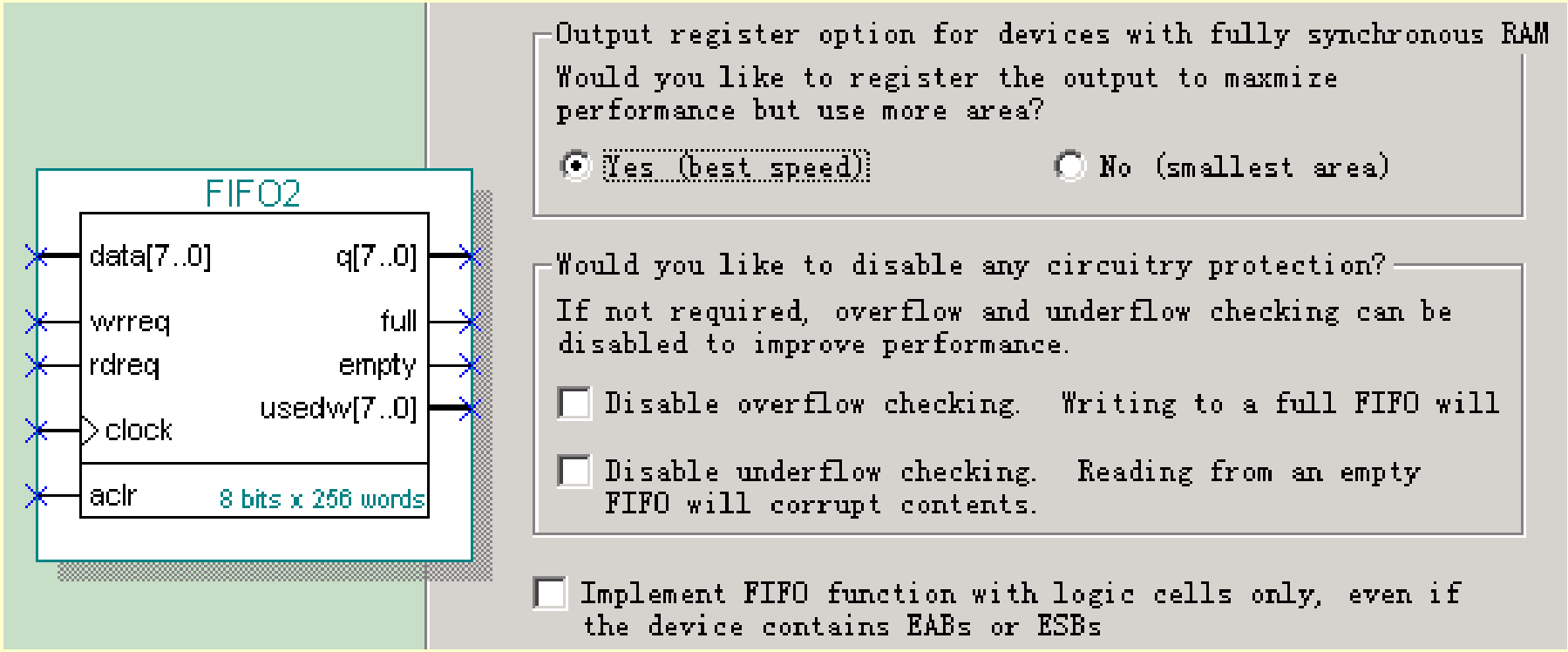


图7-22 FIFO编辑窗

7.5 其它存储器模块的定制与应用

7.5.2 FIFO定制

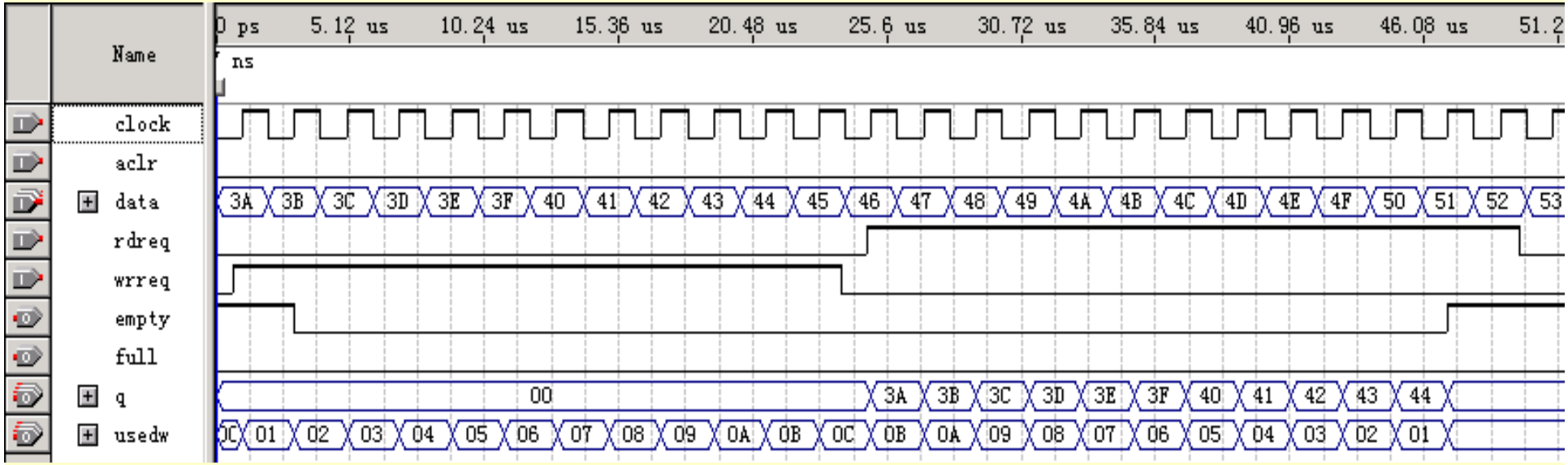


图7-23 FIFO的仿真波形

7.6流水线乘法累加器的混合输入设计

（1）用VHDL设计16位加法器。

【例7-5】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY ADDER16B IS
    PORT ( CIN : IN STD_LOGIC;
             A, B : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
             S : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
          COUT : OUT STD_LOGIC    );
END ADDER16B;
ARCHITECTURE behav OF ADDER16B IS
    SIGNAL SINT : STD_LOGIC_VECTOR(16 DOWNTO 0);
       SIGNAL AA,BB : STD_LOGIC_VECTOR(16 DOWNTO 0);
BEGIN
       AA<='0'&A;     BB<='0'& B;
    SINT <= AA + BB + CIN;   S <= SINT(15 DOWNTO 0);   COUT <= SINT(4);
END behav;
```

7.6流水线乘法累加器的混合输入设计

（2）顶层原理图文件设计。

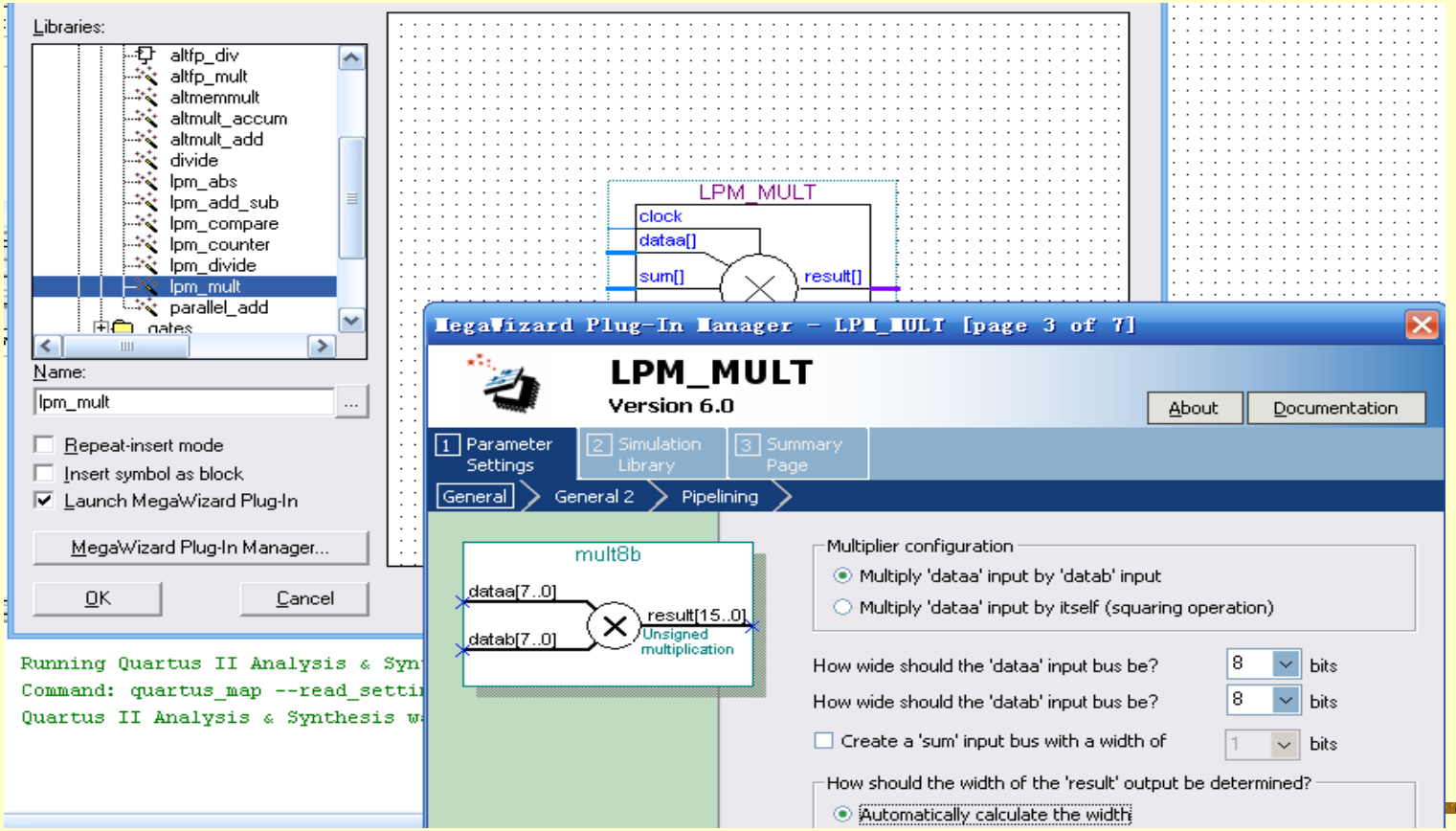


图7-24 在原理图编辑窗加入LPM元件

康芯科技

7.6流水线乘法累加器的混合输入设计

（2）顶层原理图文件设计。

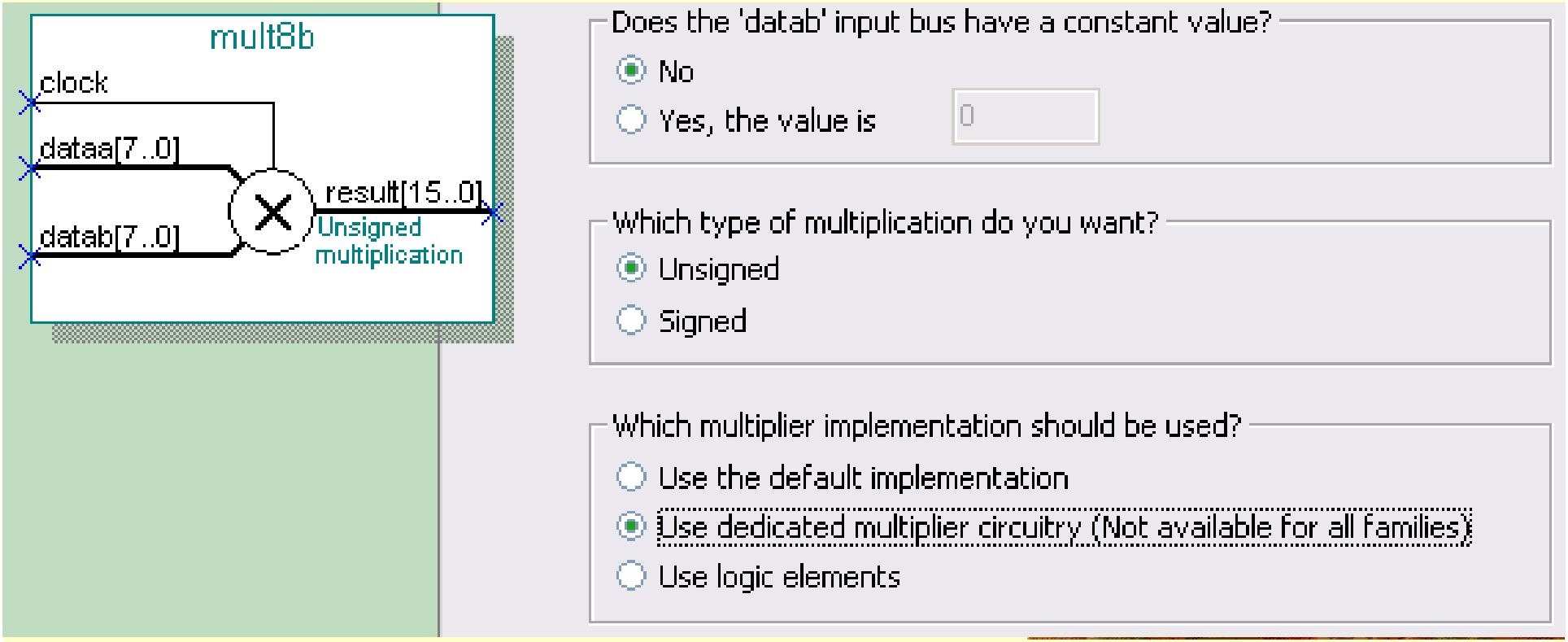


图7-25 将LPM乘法器设置为流水线工作方式

KX 康芯科技

7.6流水线乘法累加器的混合输入设计

（2）顶层原理图文件设计。

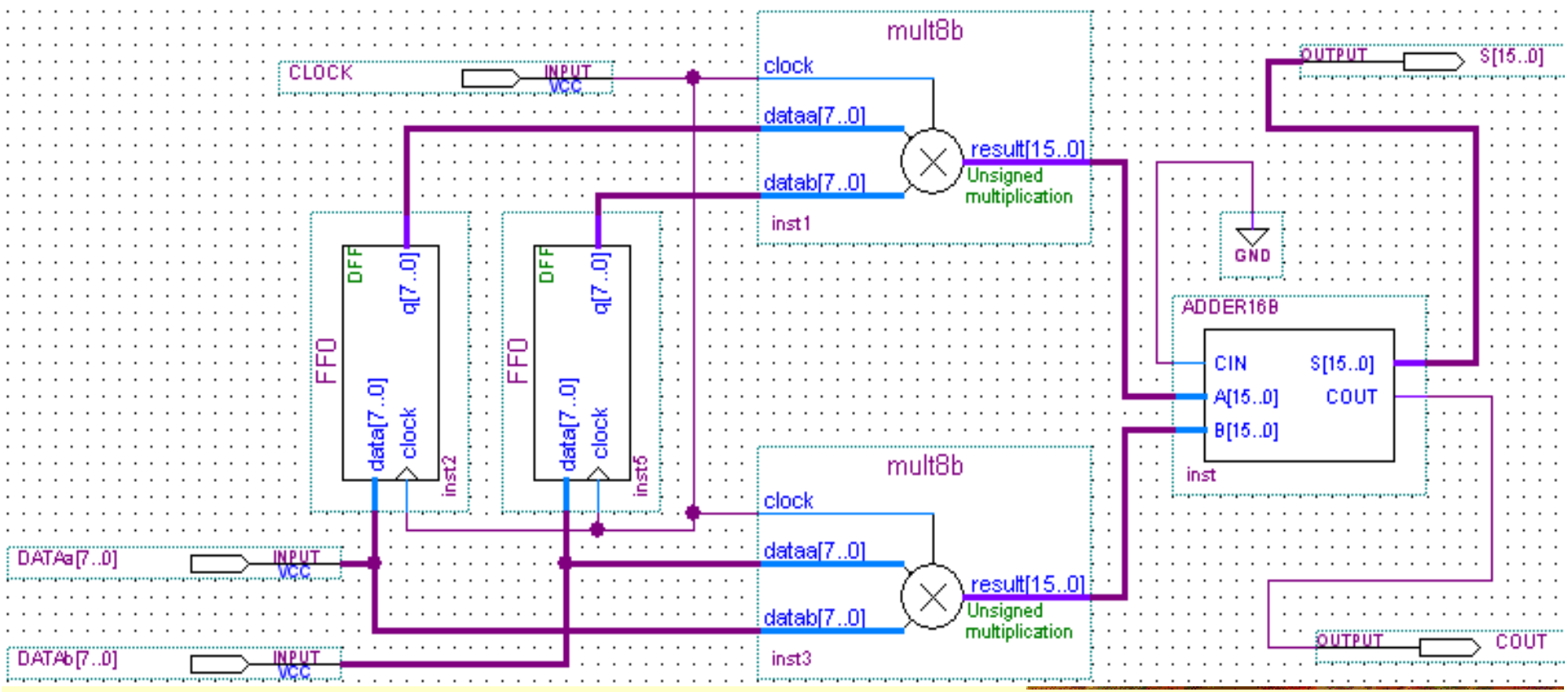


图7-26 乘法累加器电路

康芯科技

7.6流水线乘法累加器的混合输入设计

（3）仿真。

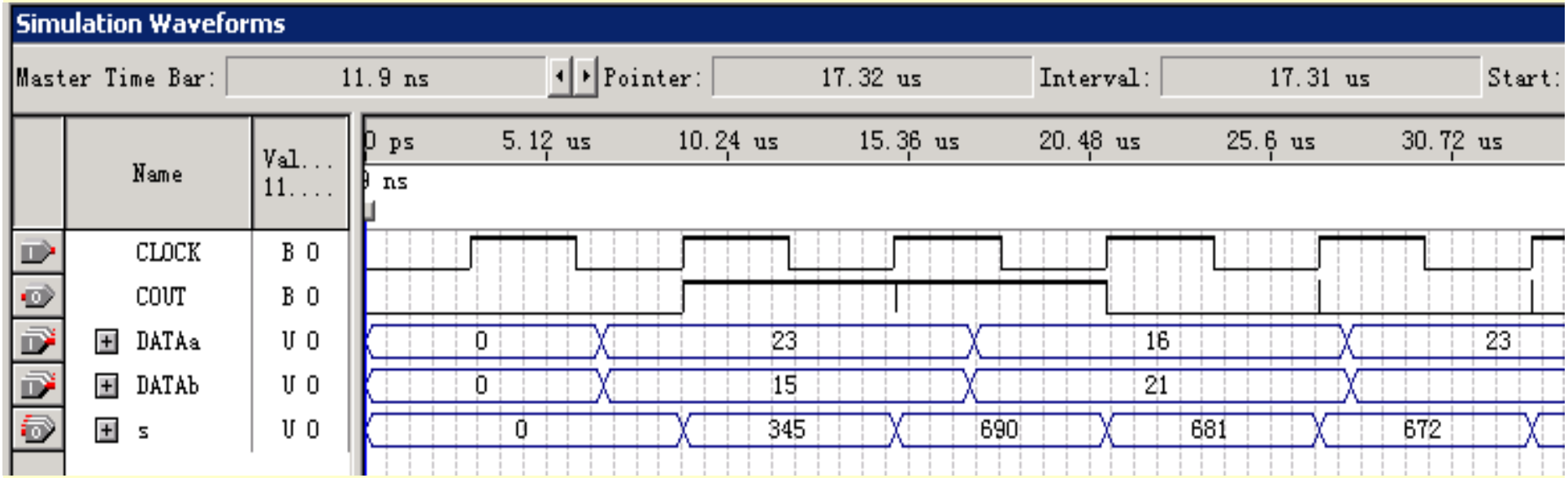


图7-27 muladd工程仿真波形

KX 康芯科技

7.6流水线乘法累加器的混合输入设计

（4）图7-28是对于图7-25在进行不同项目的选择后，编译报告给出的不同资源利用情况。

Total logic elements	224 / 4,608 (5 %
Total registers	110
Total pins	34 / 89 (38 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)

Total logic elements	17 / 4,608 (< 1 %)
Total registers	0
Total pins	34 / 89 (38 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	2 / 26 (8 %)
Total PLLs	0 / 2 (0 %)

图7-28 对乘法器选择不同设置后的编译报告

7.7 LPM嵌入式锁相环调用

7.7.1 建立嵌入式锁相环元件

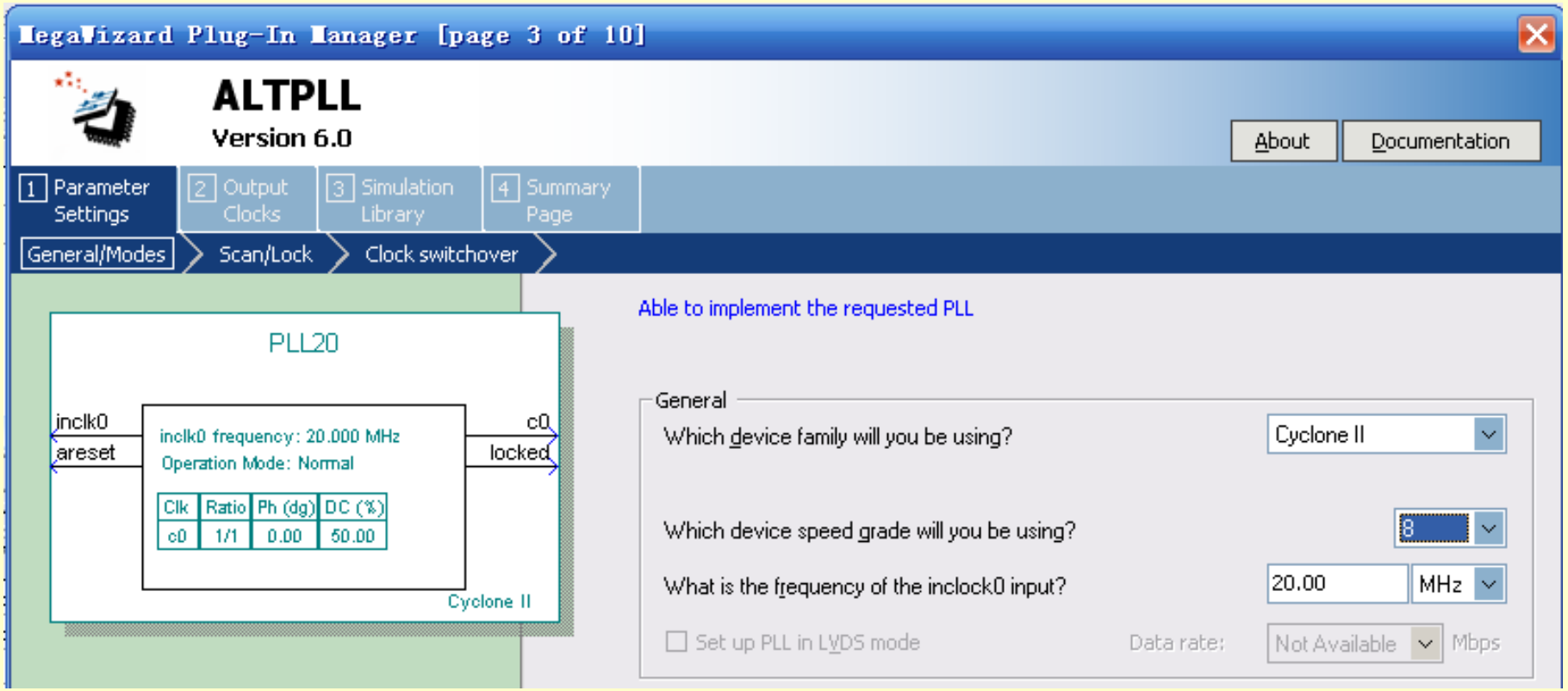


图7-29 选择参考时钟为20MHz

KX 康芯科技

7.7 LPM嵌入式锁相环调用

7.7.1 建立嵌入式锁相环元件

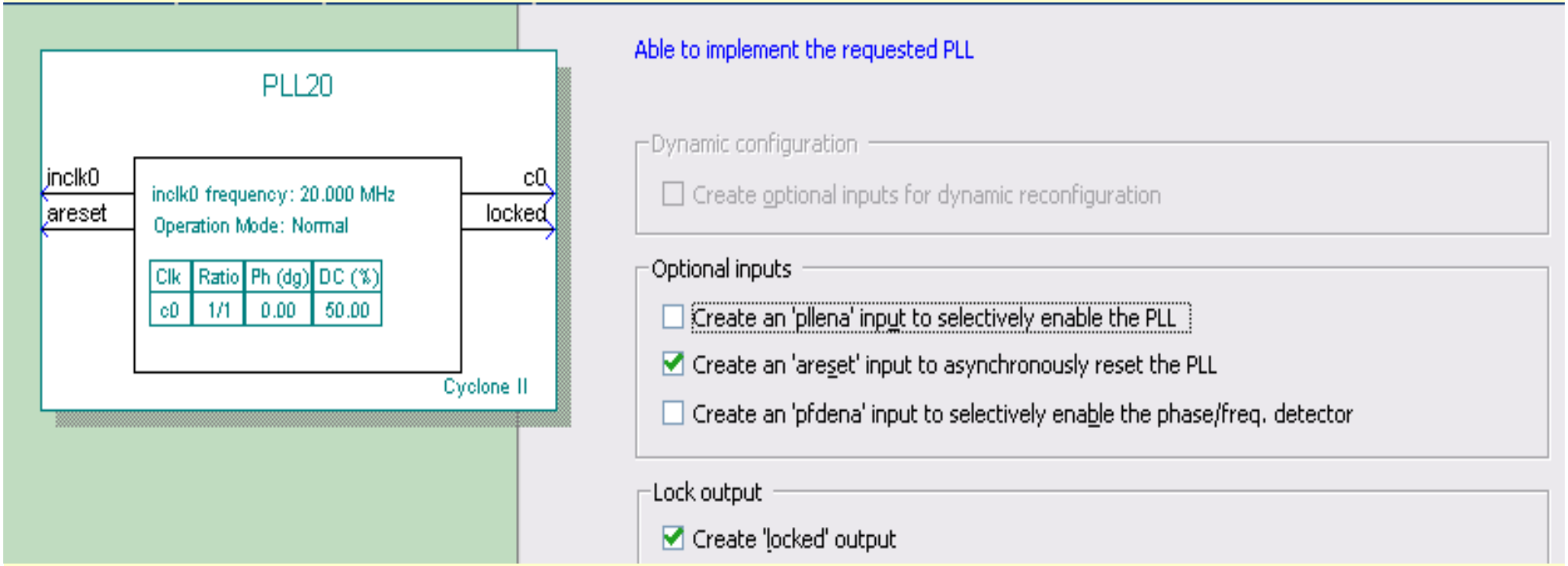


图7-30 选择控制信号

7.7 LPM嵌入式锁相环调用

7.7.1 建立嵌入式锁相环元件

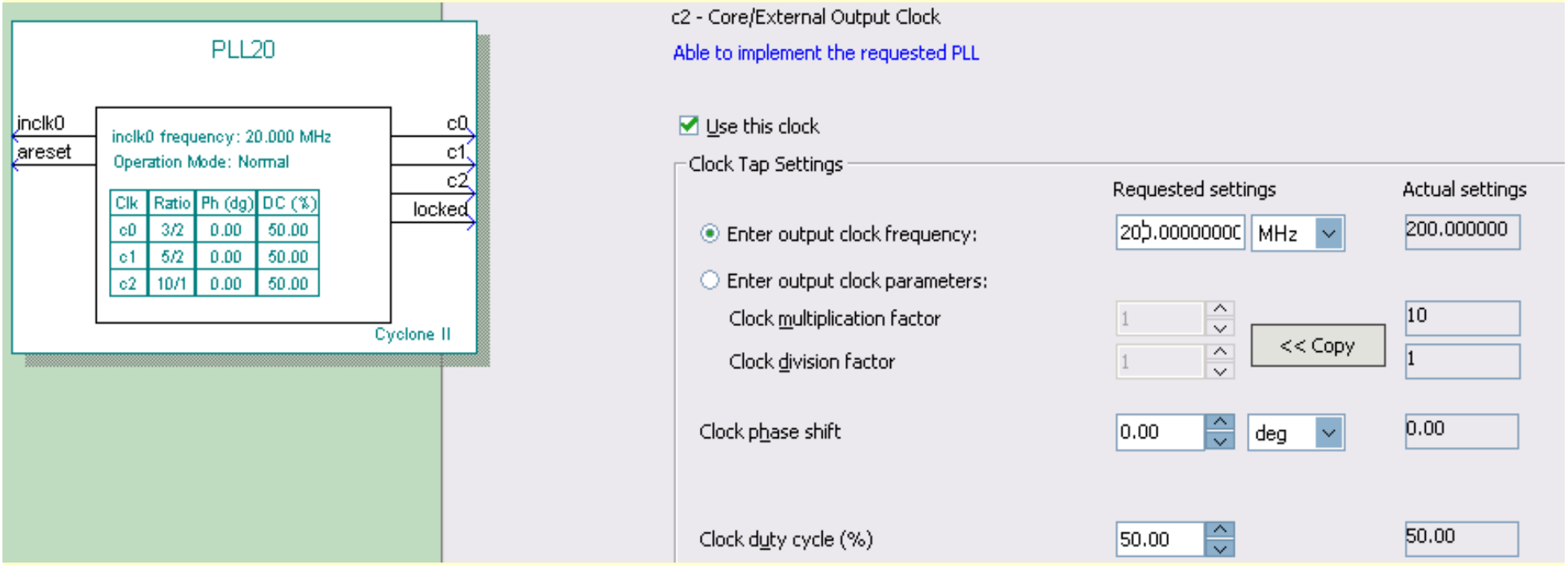


图7-31 选择e0的输出频率为210MHz

KX 康芯科技

7.7 LPM嵌入式锁相环调用

7.7.2 测试锁相环

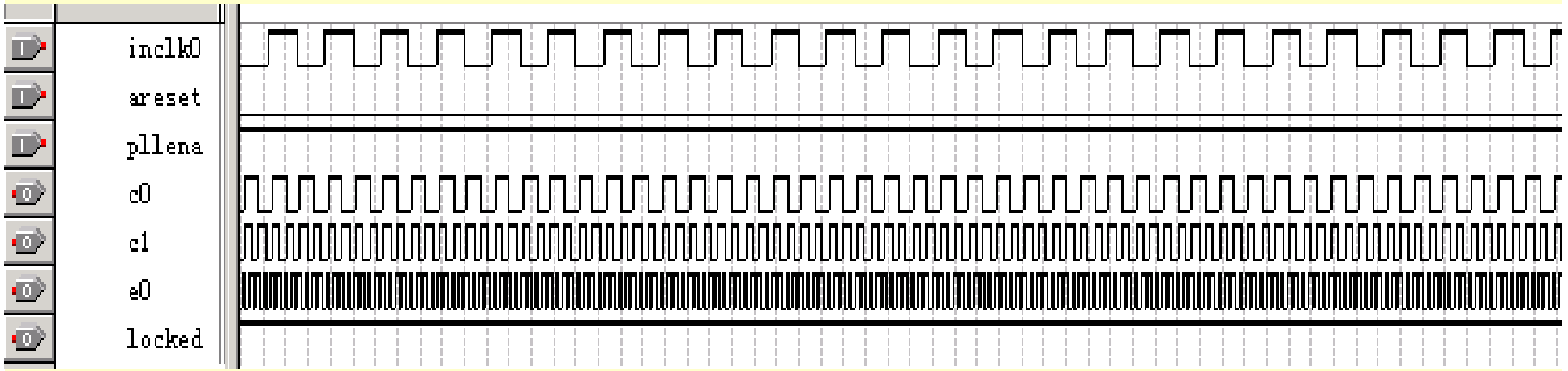


图7-32 PLL元件的仿真波形

7.7.2 测试锁相环

单频率输出的应用PLL的示例：

```
...;
ENTITY DDS_VHDL IS
    PORT (     CLKK : IN  STD_LOGIC;  --此时钟进入锁相环
               FWORD : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
...;
ARCHITECTURE one OF DDS_VHDL IS
   COMPONENT PLLU                              --调入PLL声明
        PORT (    inclk0 : IN STD_LOGIC  := '0';
                    c0 : OUT STD_LOGIC );
    END COMPONENT;
     COMPONENT REG32B
...;
BEGIN
...;
  u6 :  SIN_ROM PORT MAP( address=>D32B(31 DOWNTO 22), q=>POUT,
inclock=>CLK );
  u7 :    PLL20  PORT MAP( inclk0=> CLKK,c0=>CLK);  --例化
END;
```

7.8 IP核NCO数控振荡器使用方法

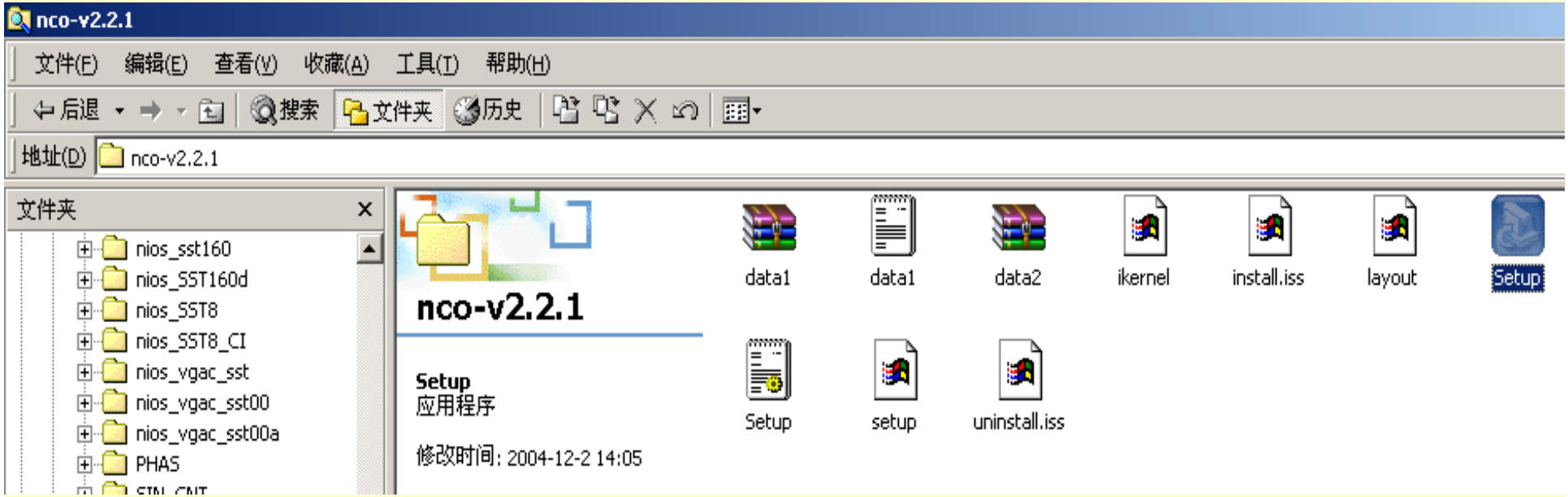


图7-33 安装NCO核

7.8 IP核NCO数控振荡器使用方法

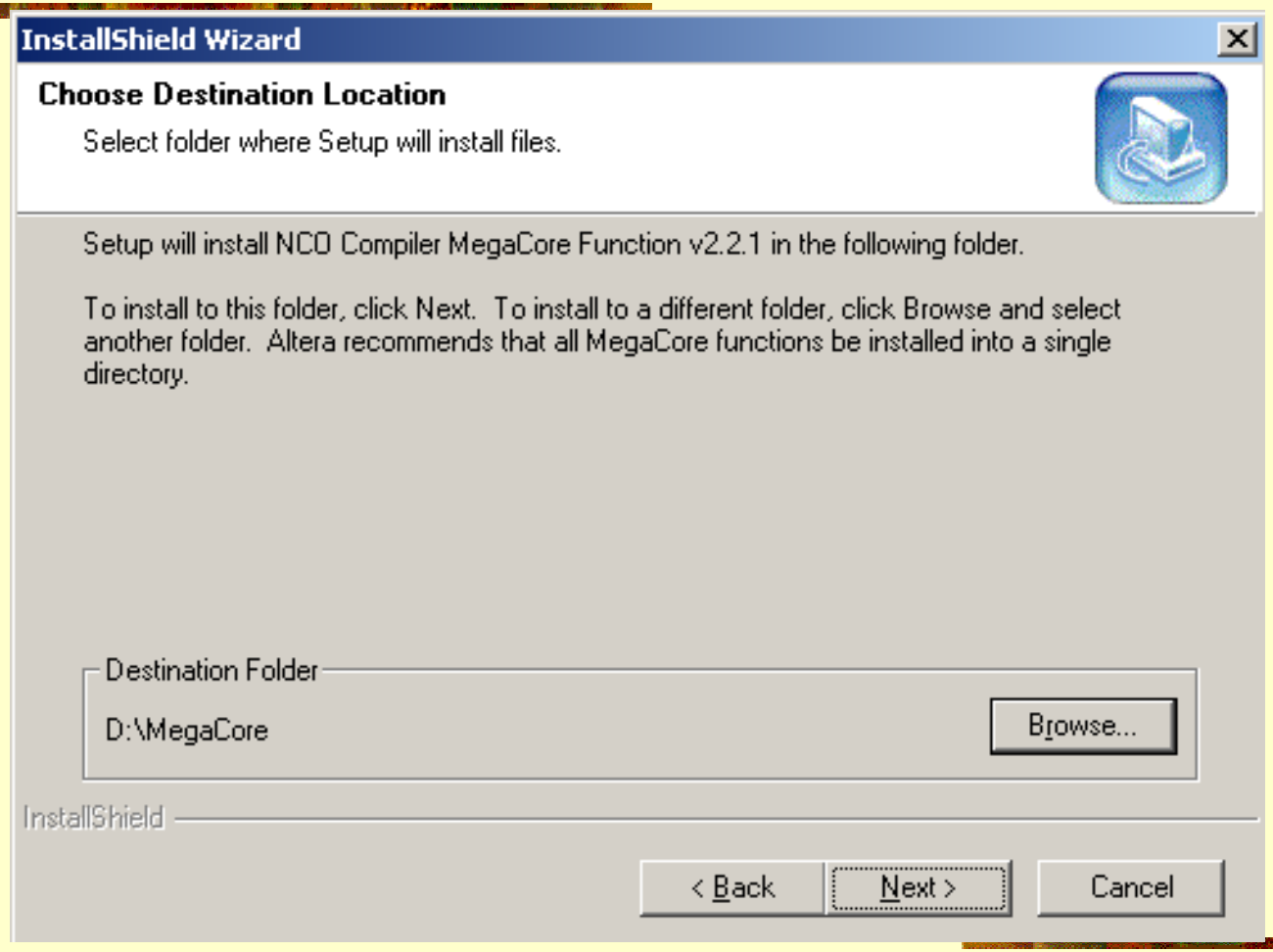


图7-34 确定安装路径

7.8 IP核NCO数控振荡器使用方法

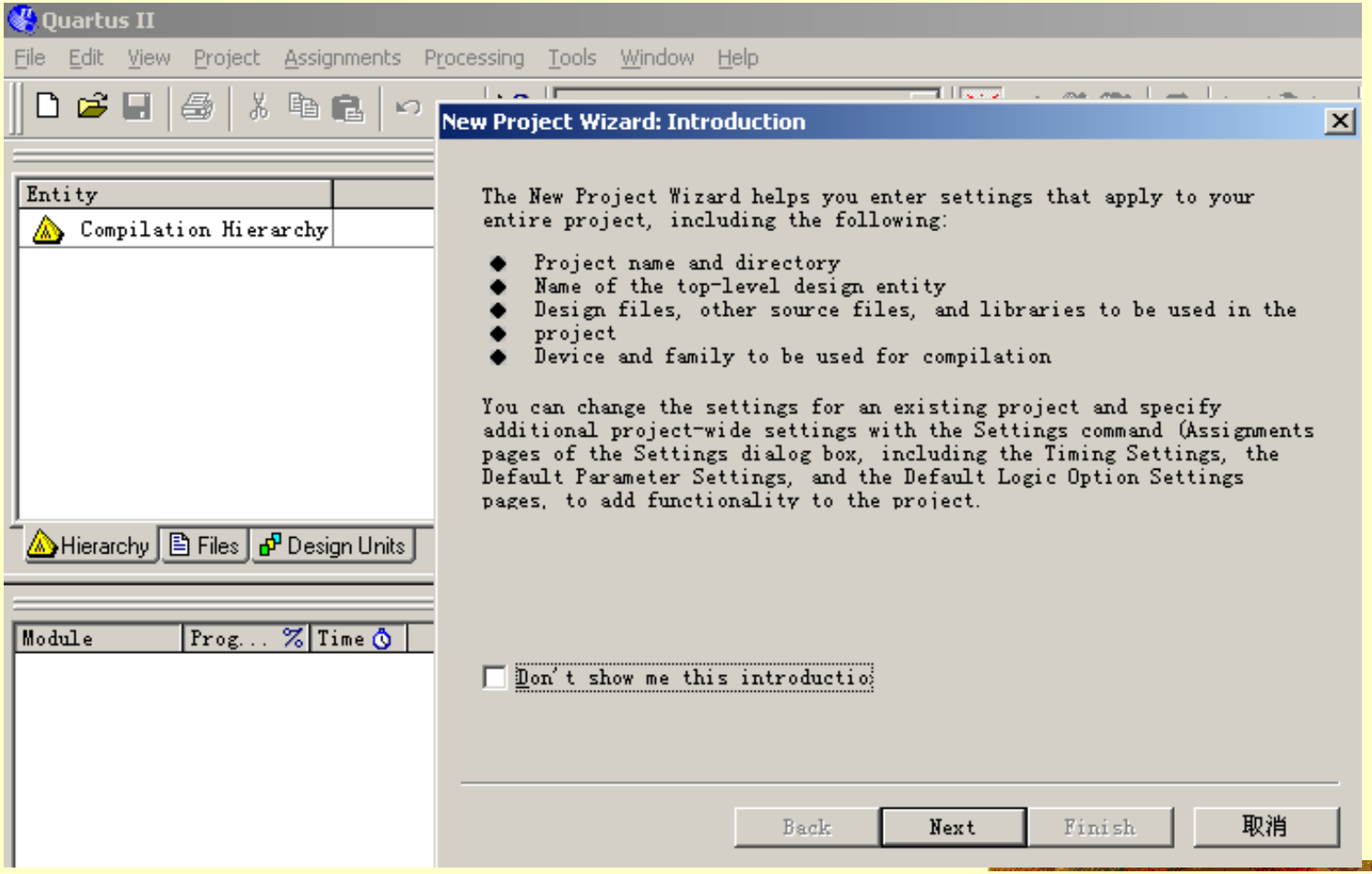


图7-35 开始Core的工程路径

7.8 IP核NCO数控振荡器使用方法

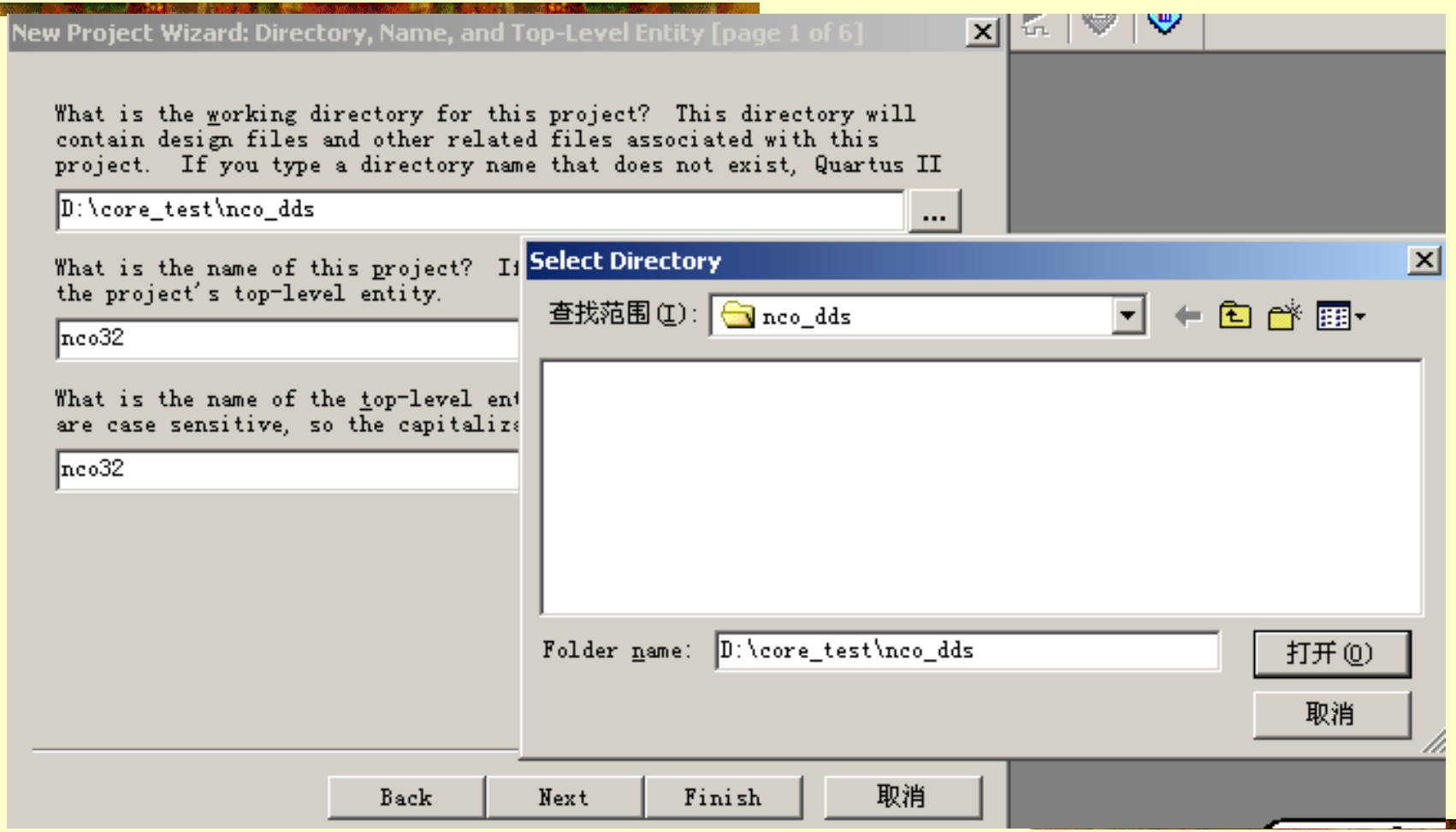


图7-36 确定工程路径和工程名

7.8 IP核NCO数控振荡器使用方法

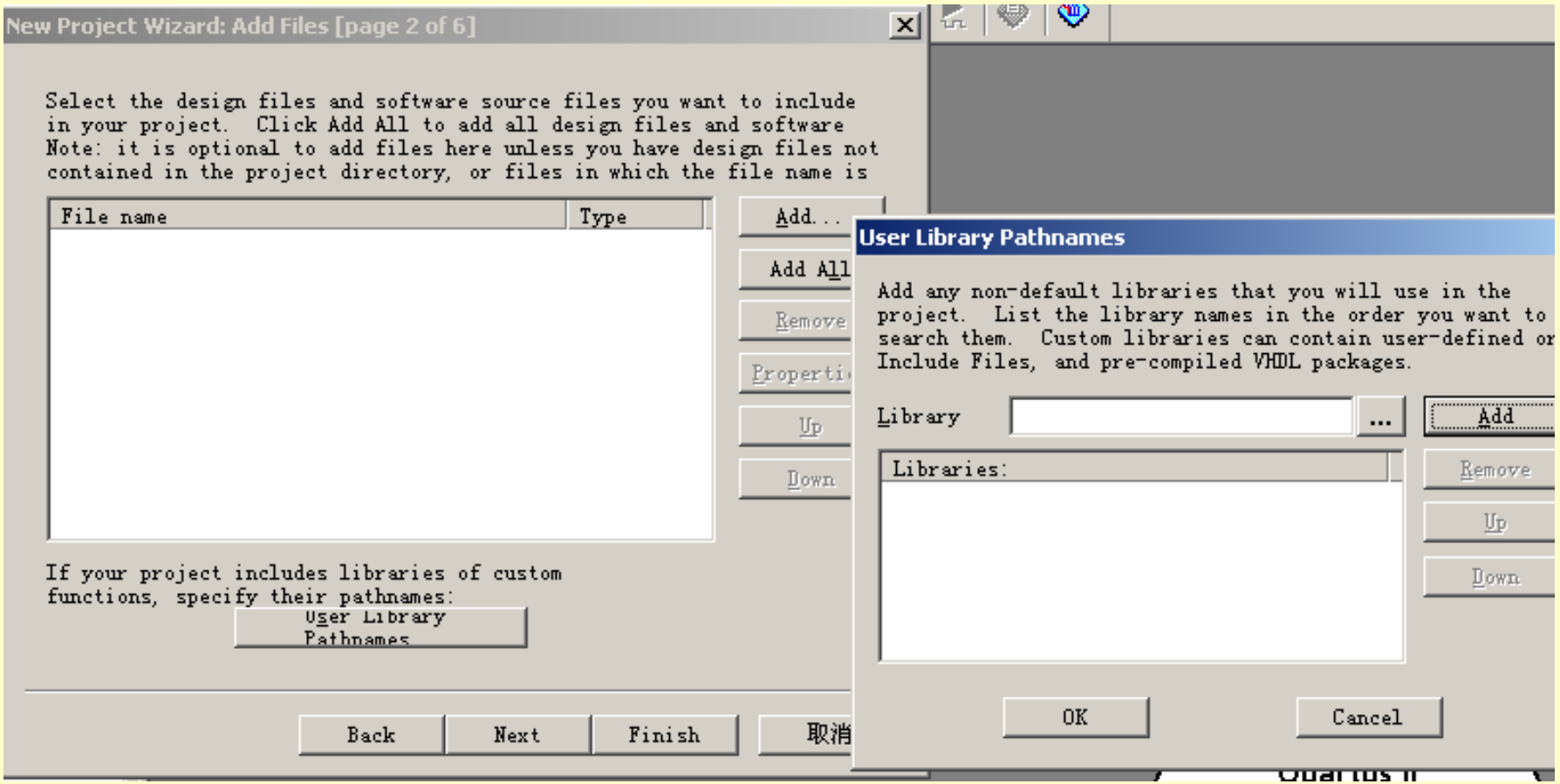


图7-37 打开Core用户库设置窗

7.8 IP核NCO数控振荡器使用方法

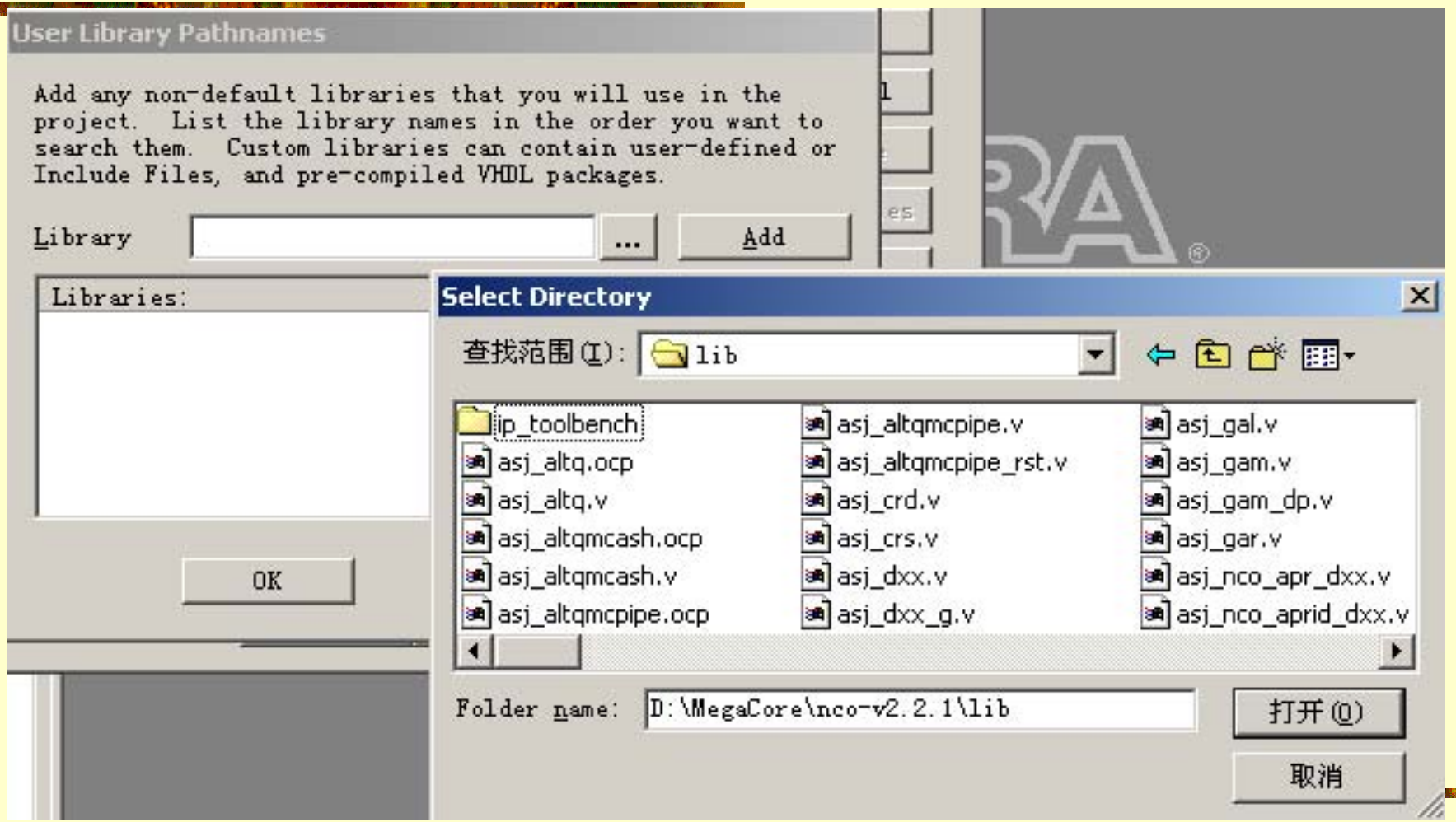


图7-38 选中确定路径上的NCO库

7.8 IP核NCO数控振荡器使用方法

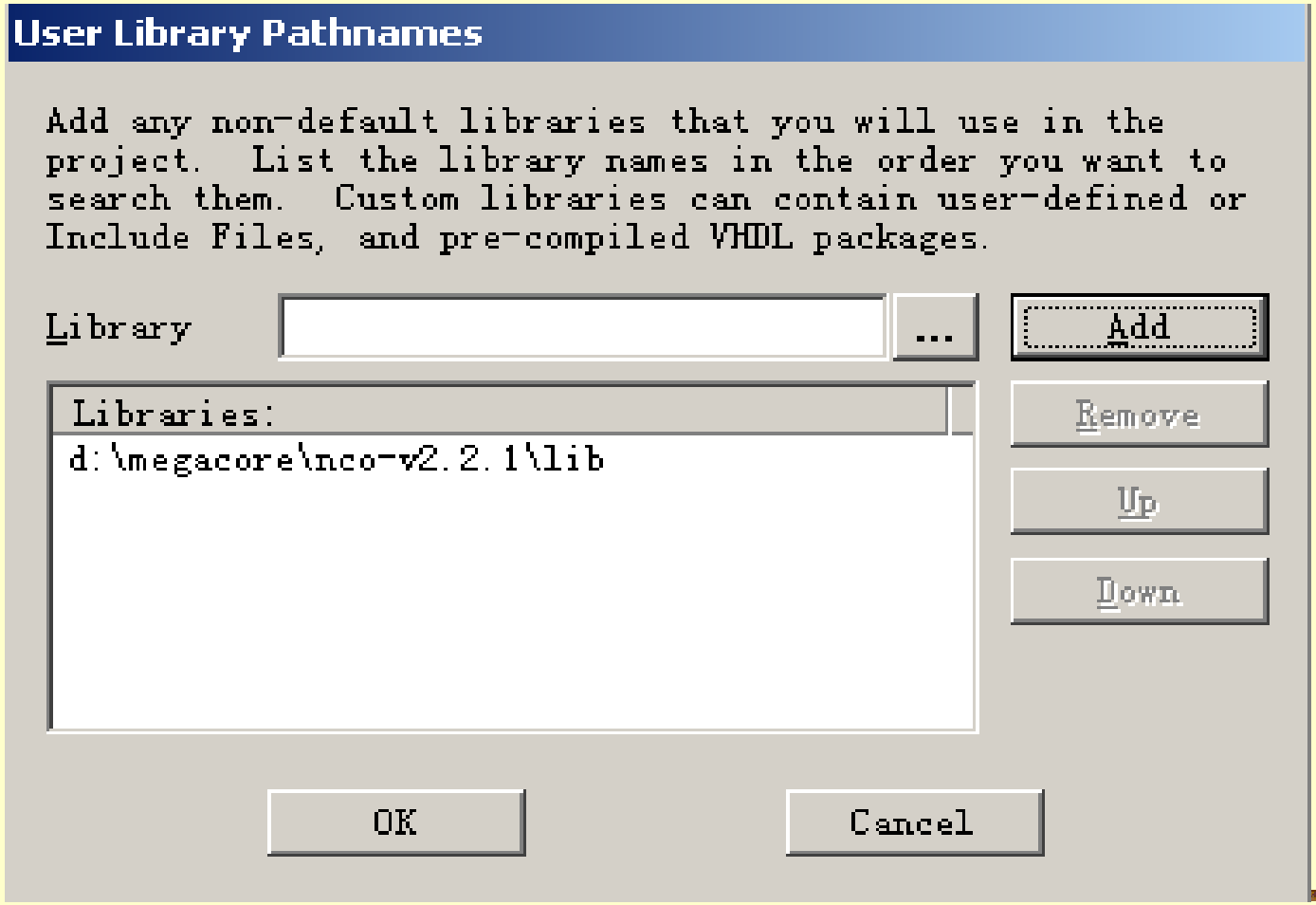


图7-39 加入NCO库

7.8 IP核NCO数控振荡器使用方法

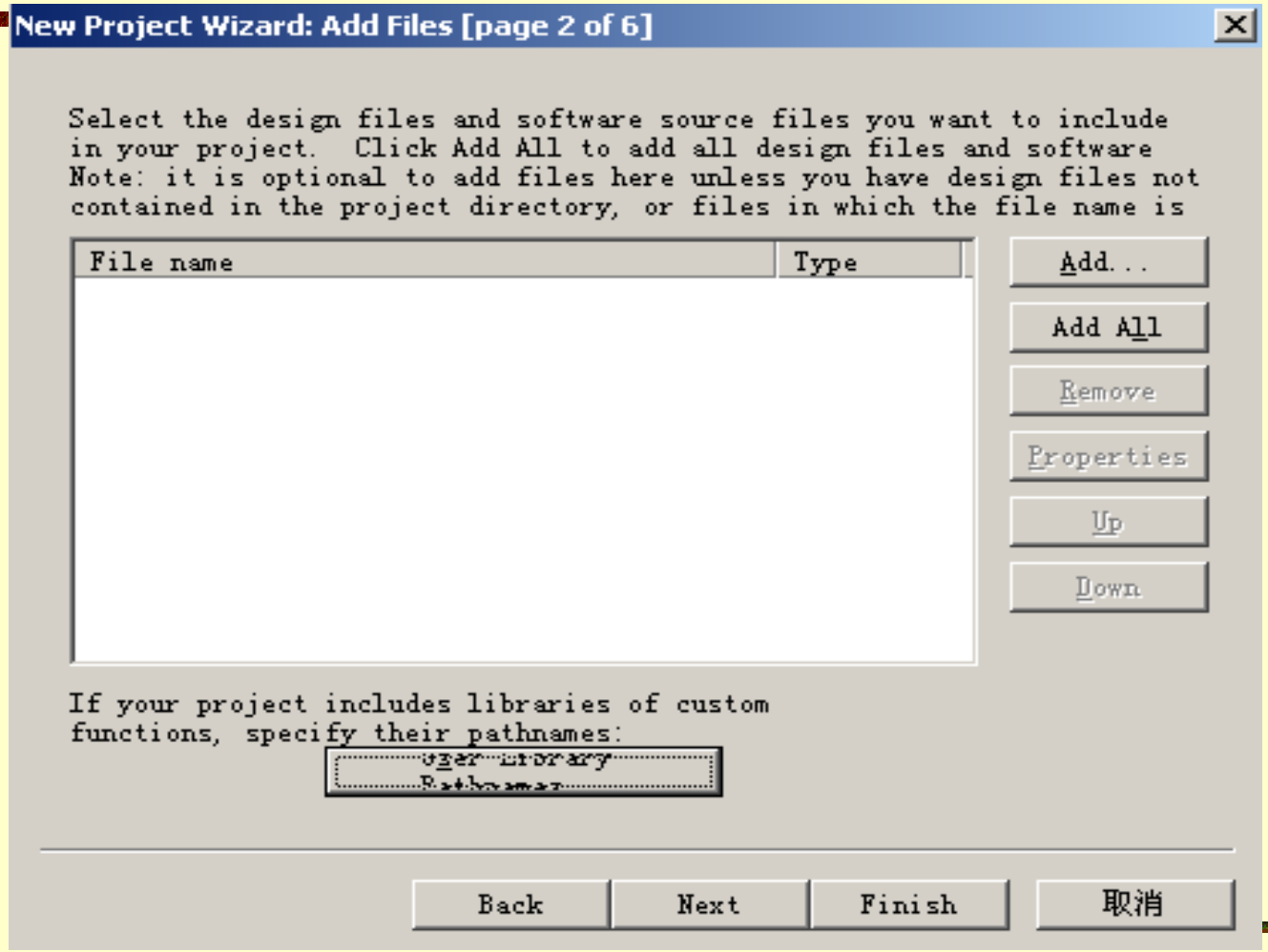


图7-40 已经在工程中加入NCO库

7.8 IP核NCO数控振荡器使用方法

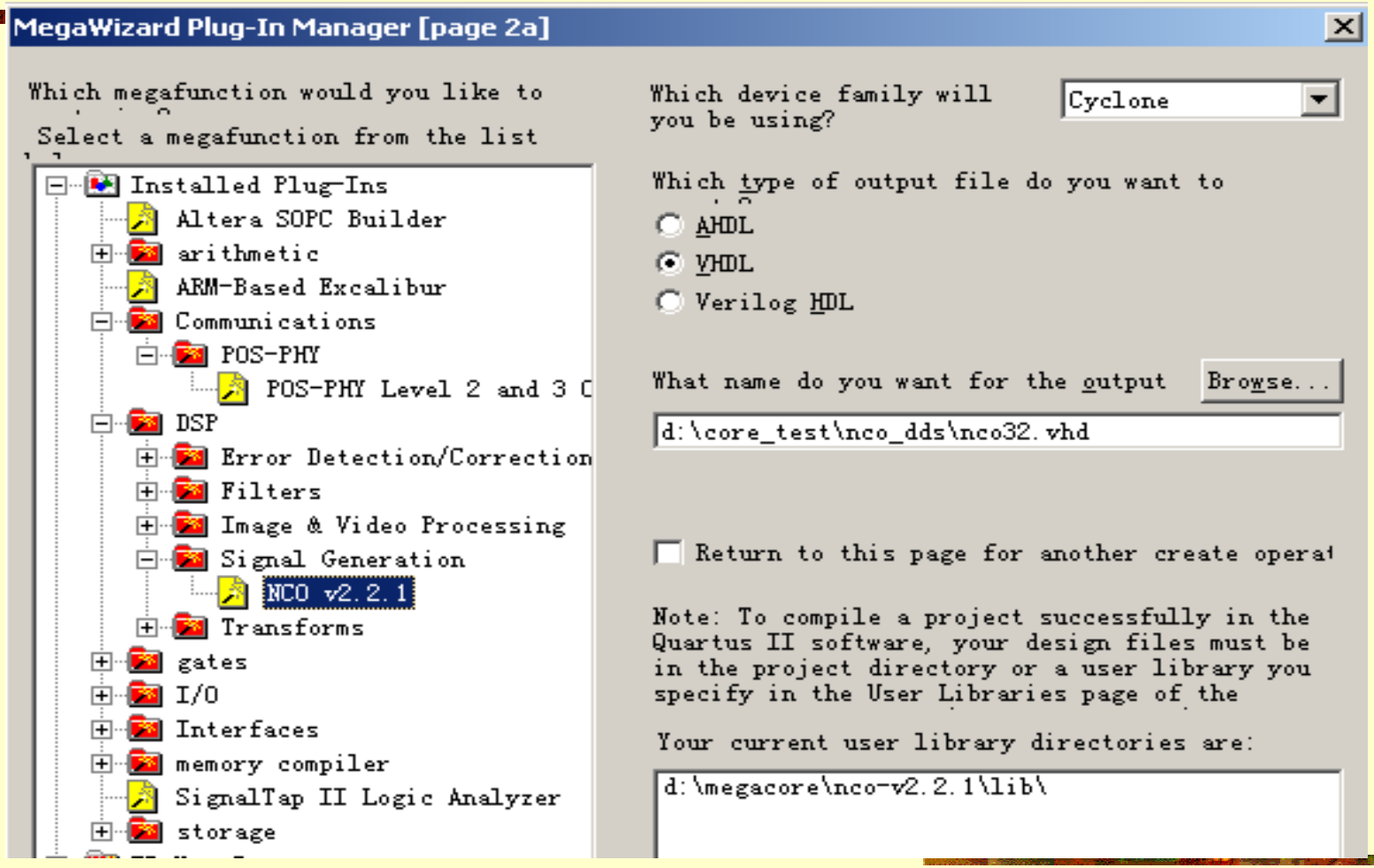


图7-41 打开Core设置管理窗

7.8 IP核NCO数控振荡器使用方法

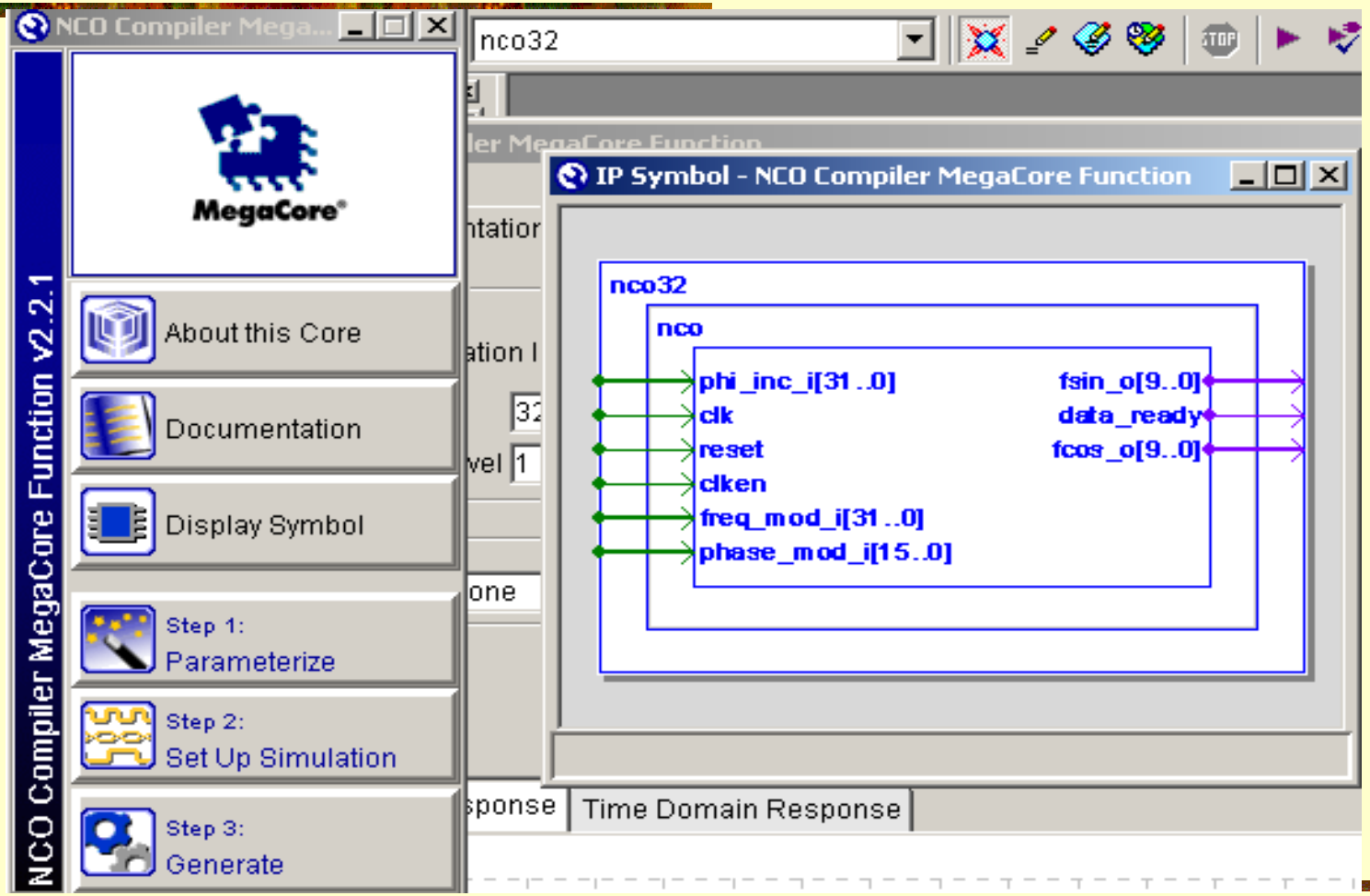


图7-42 开始进入Core参数设置窗Toolbench

康芯科技

7.8 IP核NCO数控振荡器使用方法

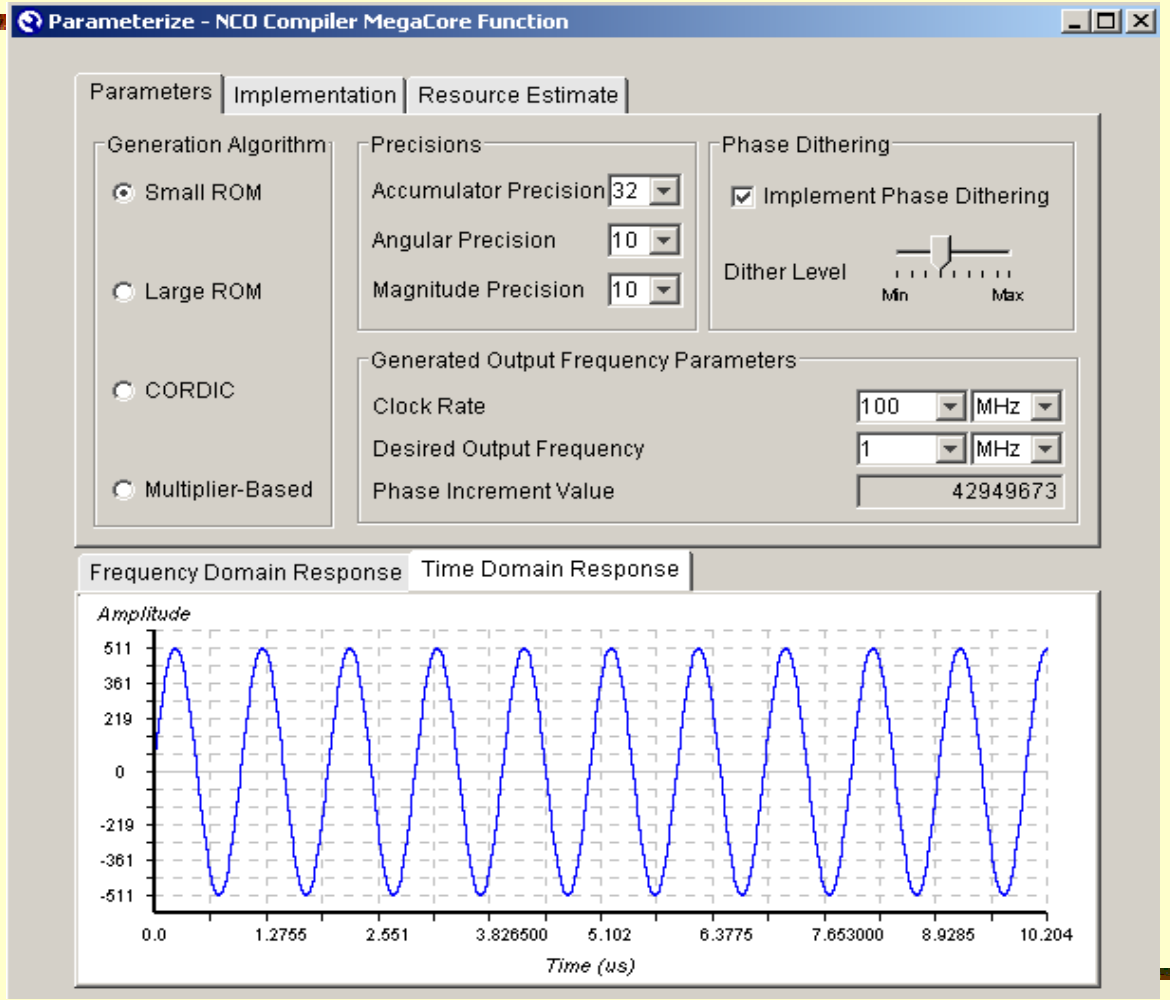


图7-43 设置NCO参数

7.8 IP核NCO数控振荡器使用方法

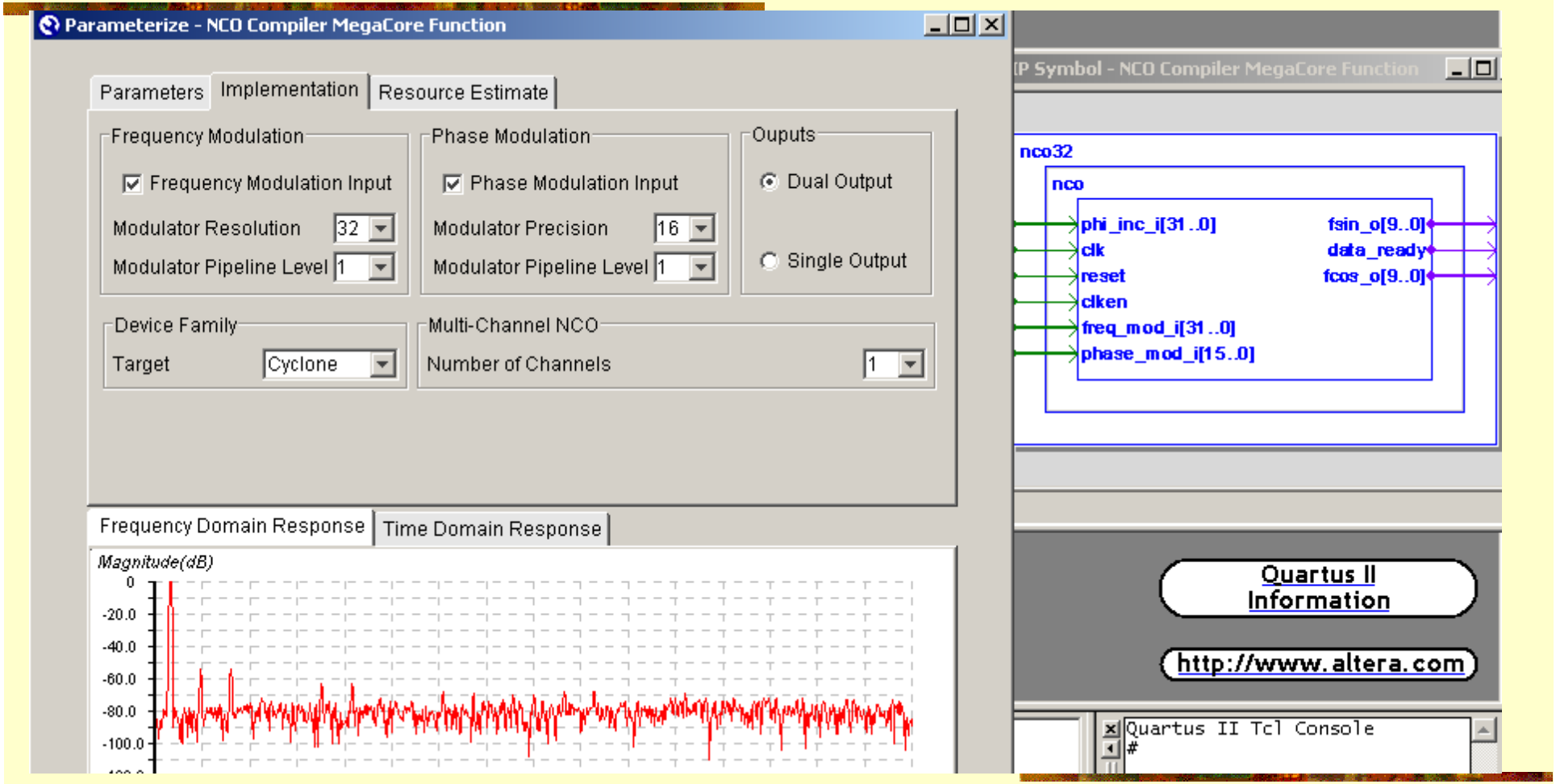


图7-44设置NCO参数

7.8 IP核NCO数控振荡器使用方法

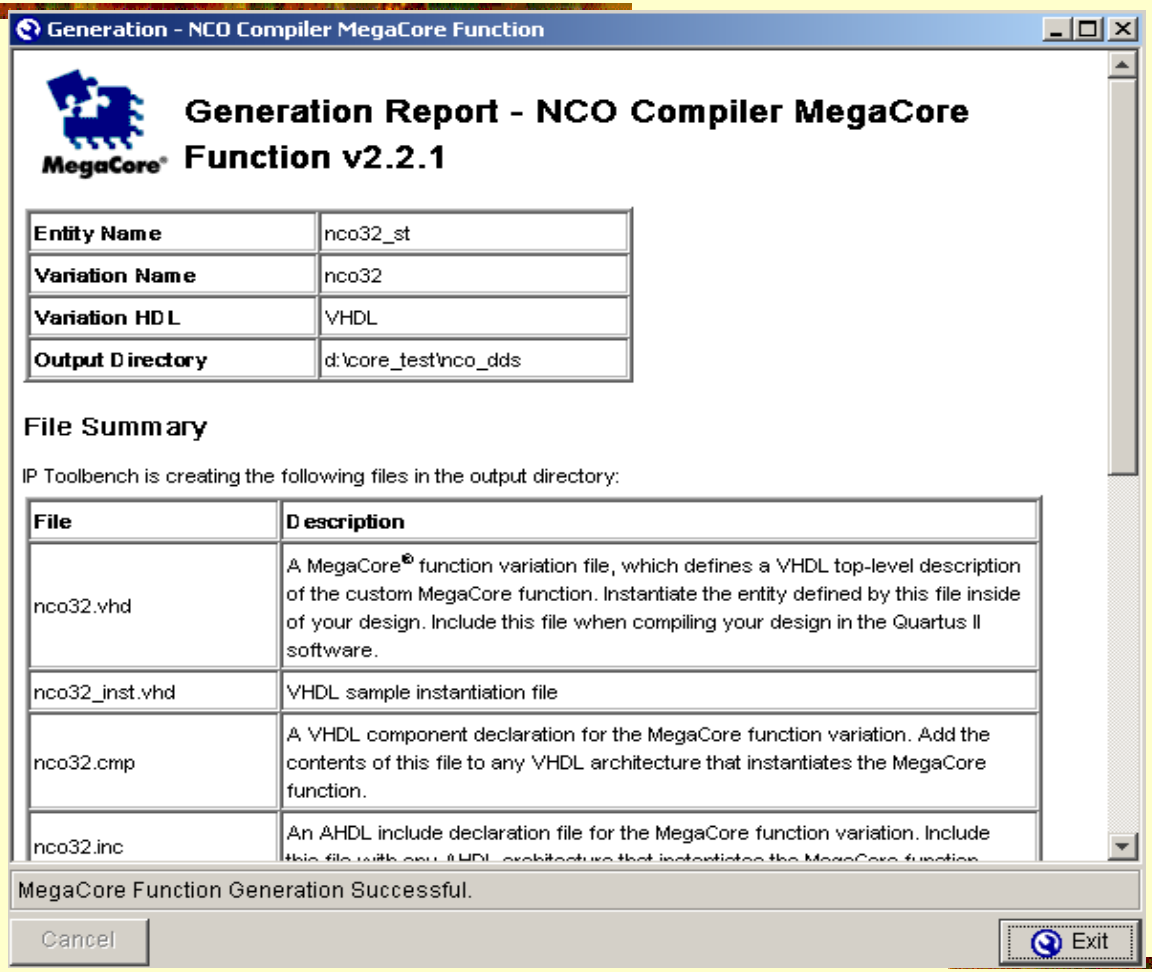


图7-45完成NCO参数设置并生成设计文件后的信息窗

KX 康芯科技

7.8 IP核NCO数控振荡器使用方法

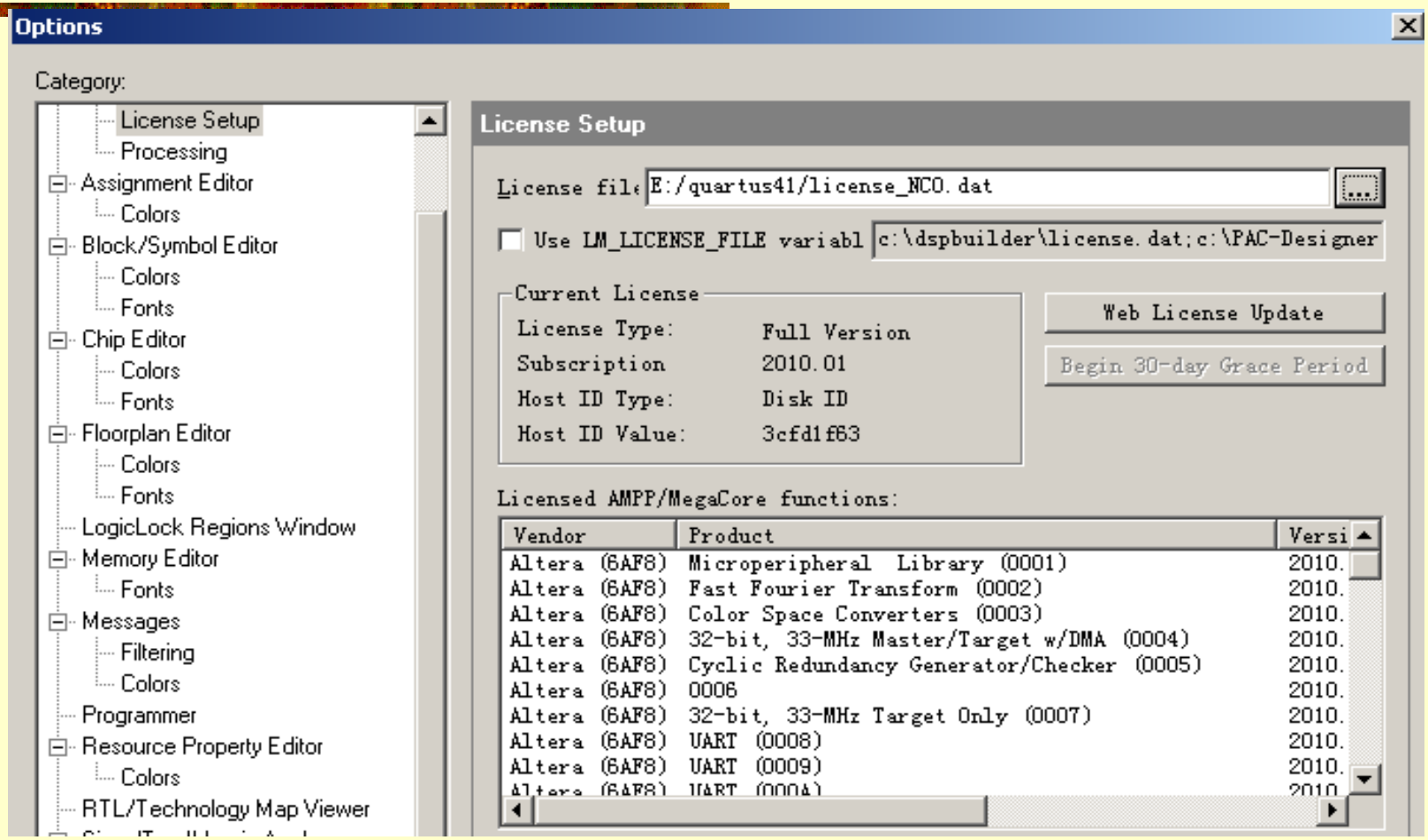


图7-46 加入NCO的授权文件

7.8 IP核NCO数控振荡器使用方法

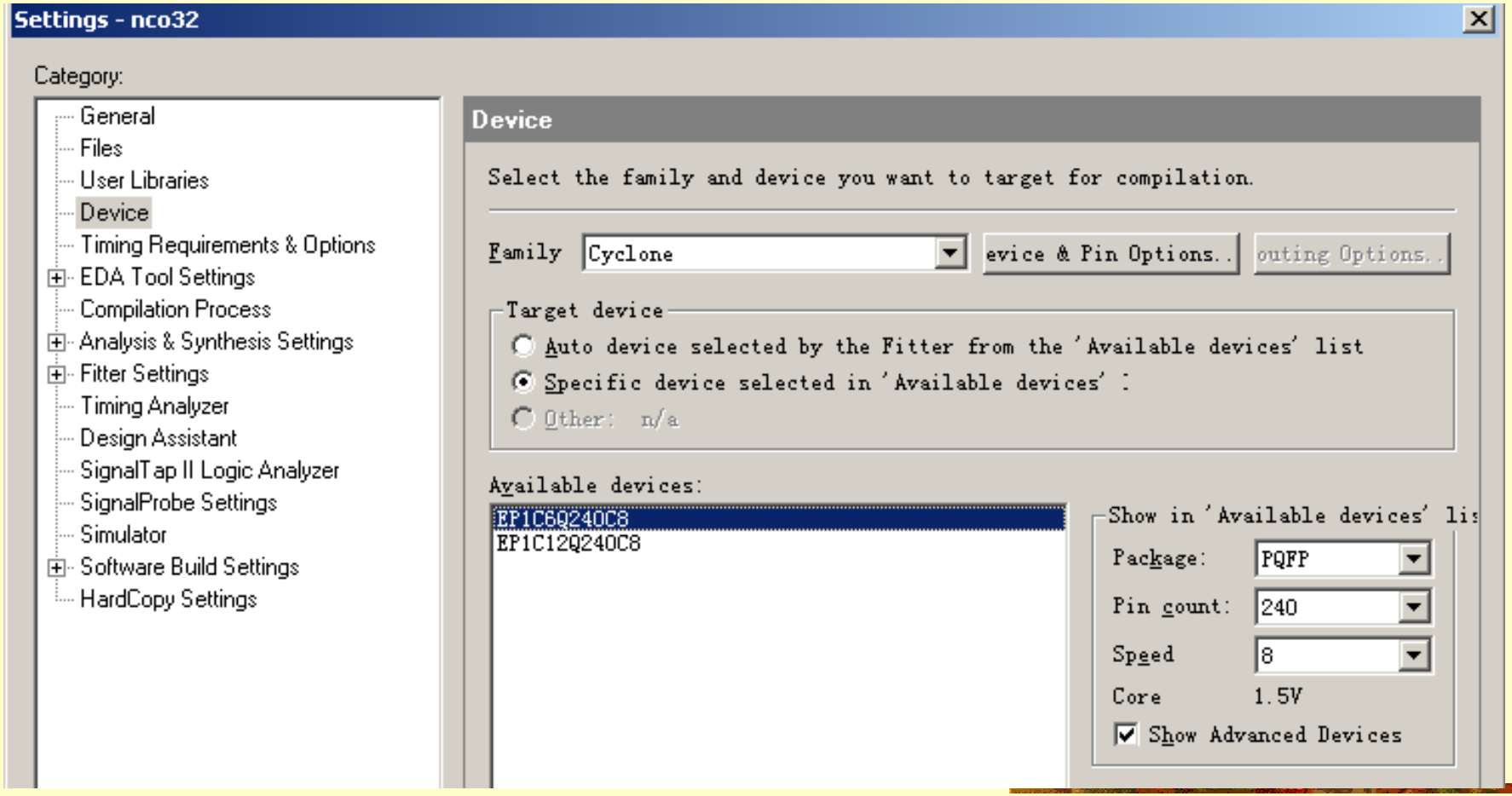


图7-47 选定FPGA目标器件

康芯科技

7.8 IP核NCO数控振荡器使用方法

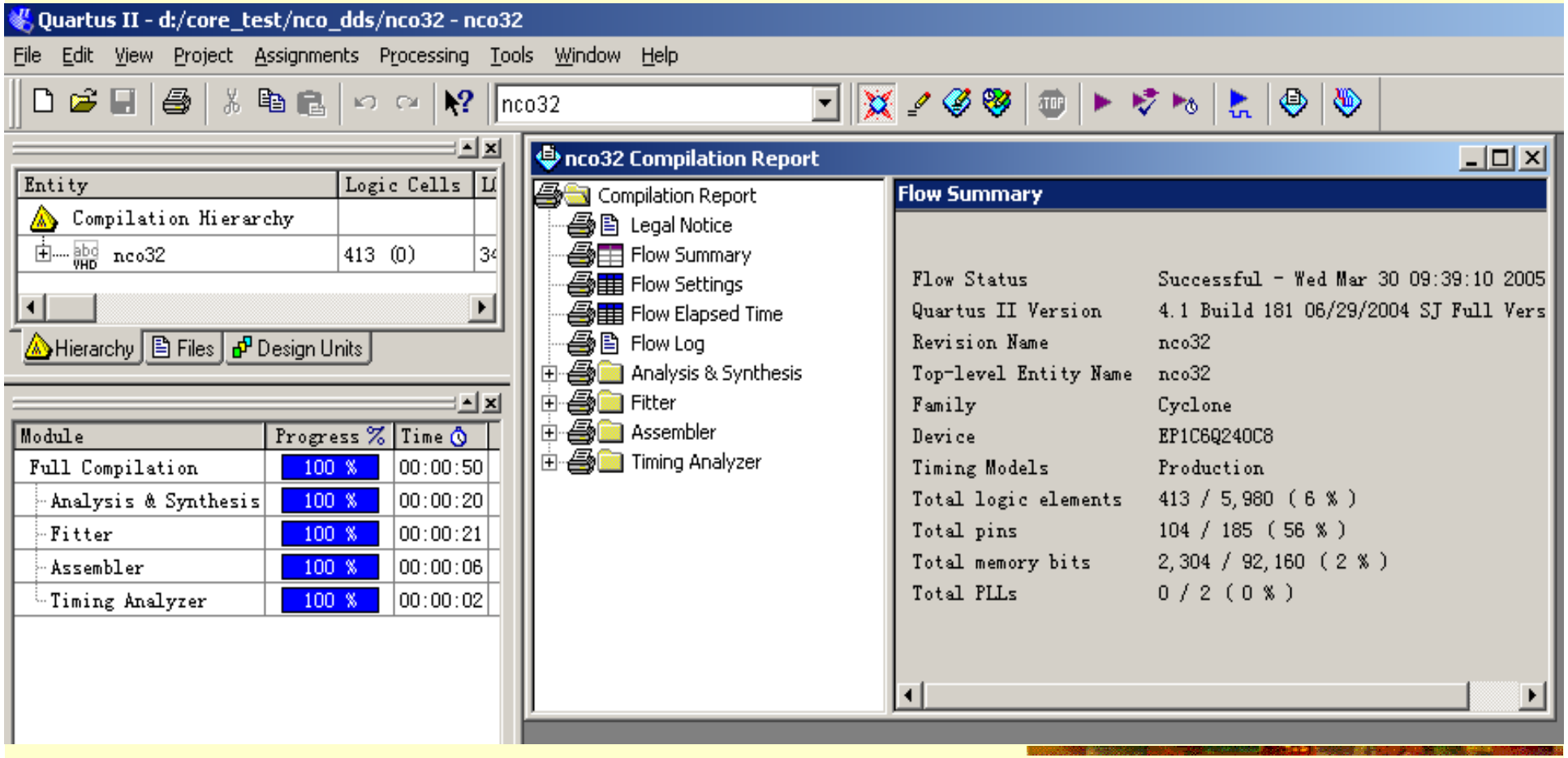


图7-48 设定工程后进行全程编译

KX 康芯科技

7.9 8051单片机IP核应用

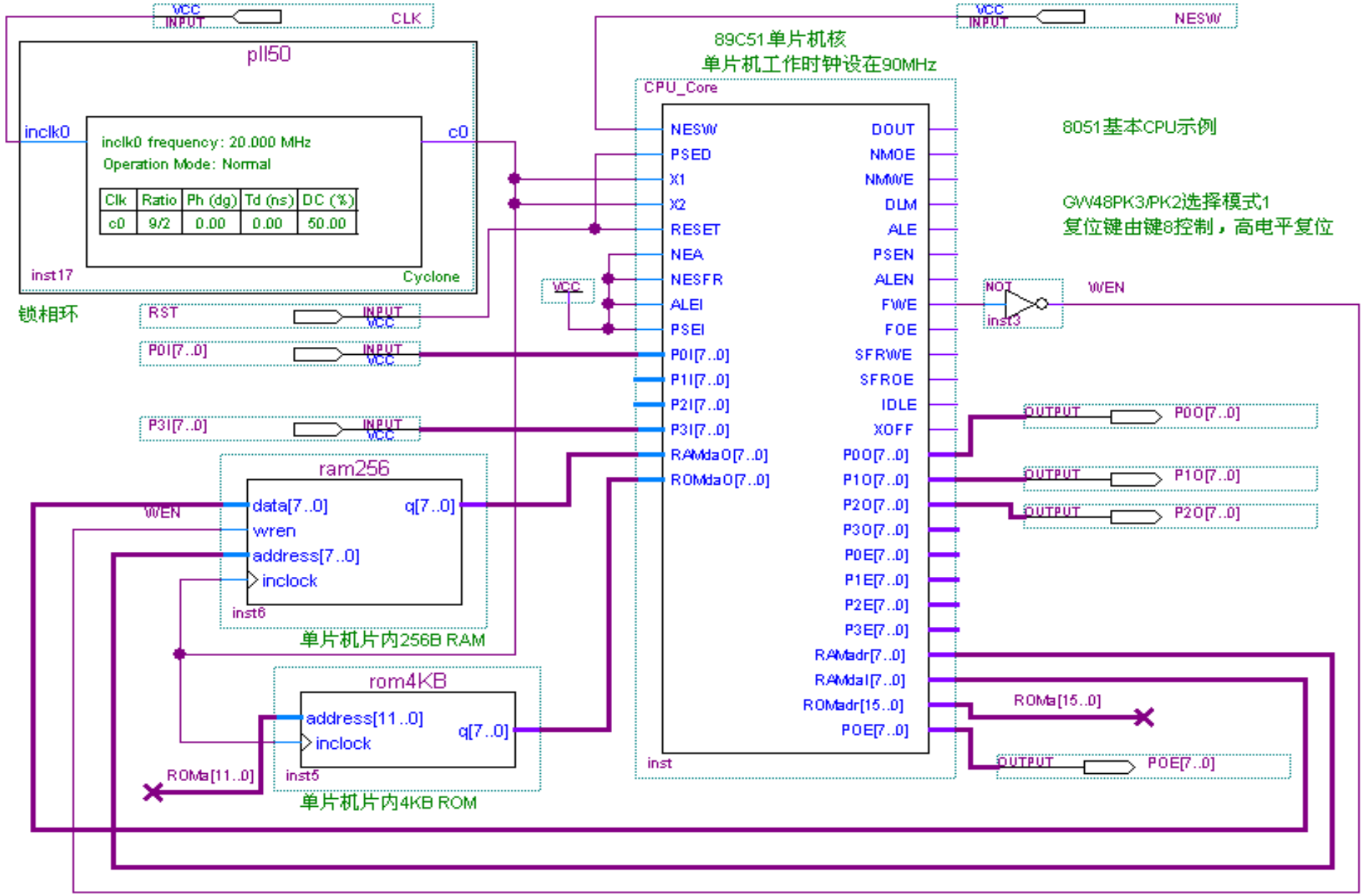


图7-49 基本8051CPU核应用电路示例

7.9 8051单片机IP核应用

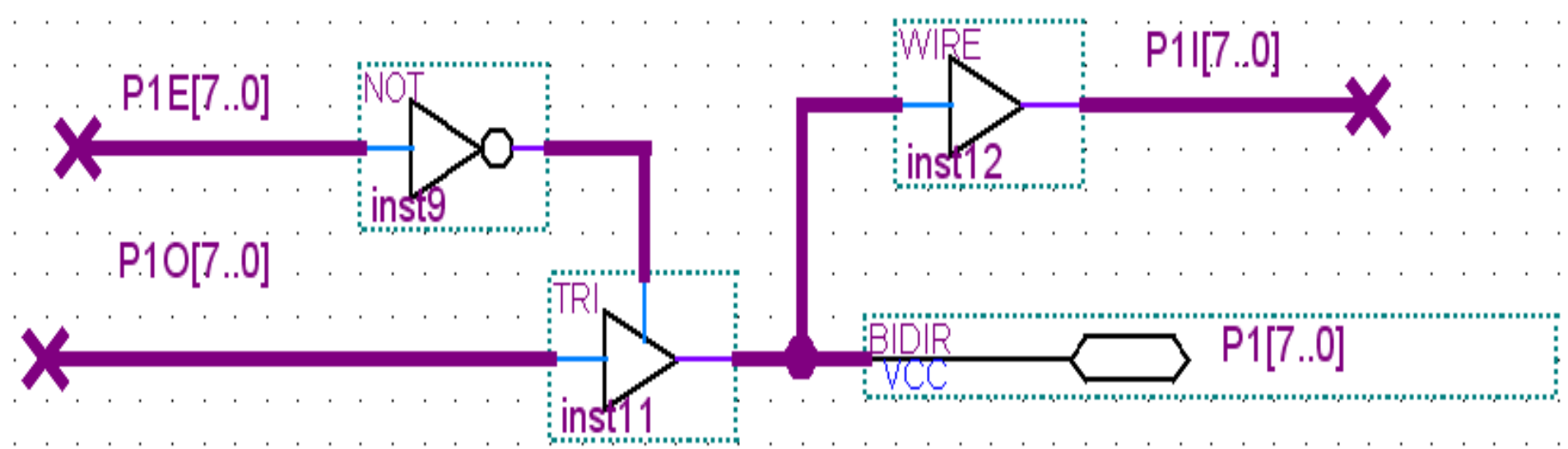


图7-50 单片机I/O口设置成双向口的电路

7.9 8051单片机IP核应用

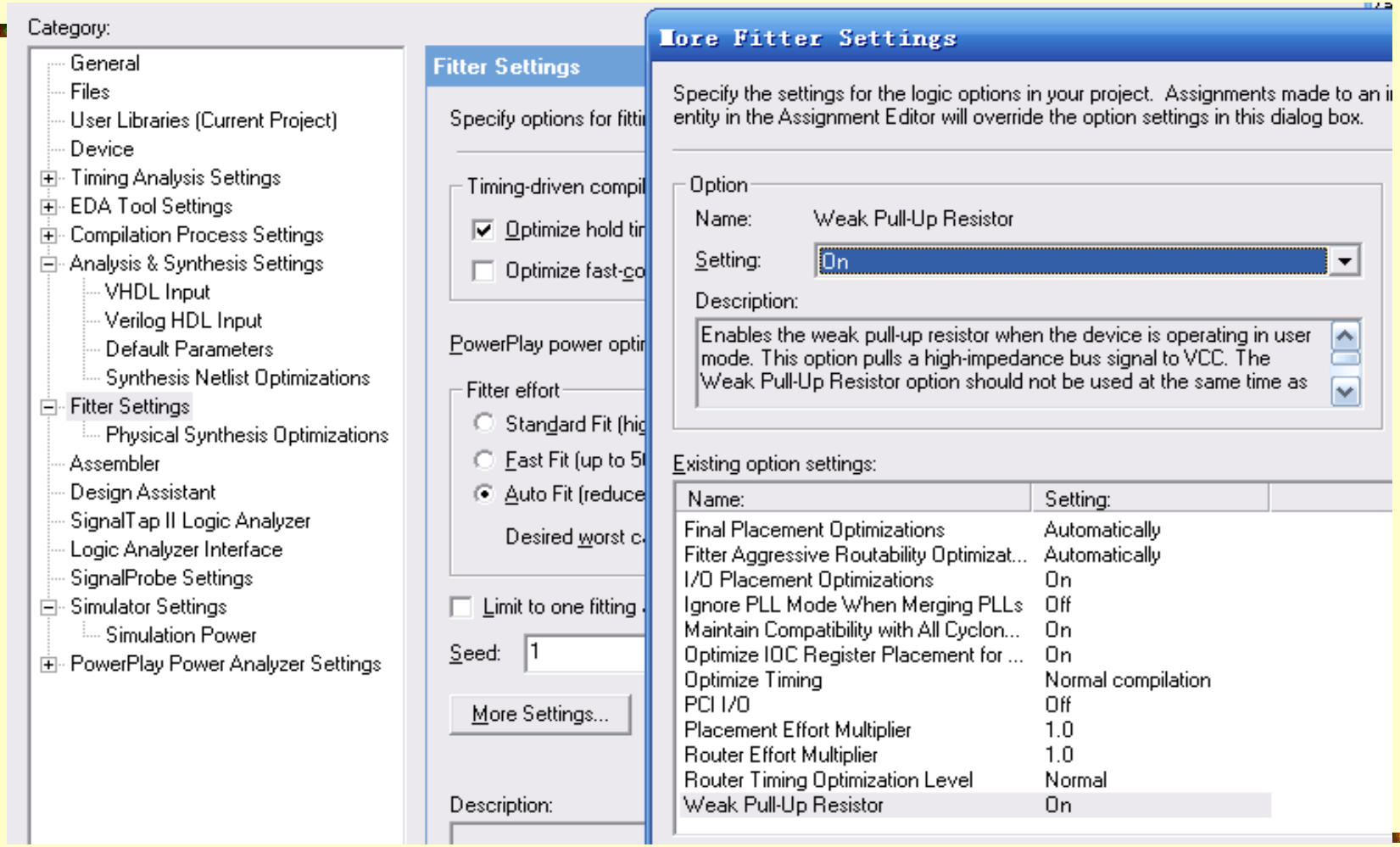


图7-51 设置FPGA的总线口输出为上拉

KX 康芯科技

7.9 8051单片机IP核应用

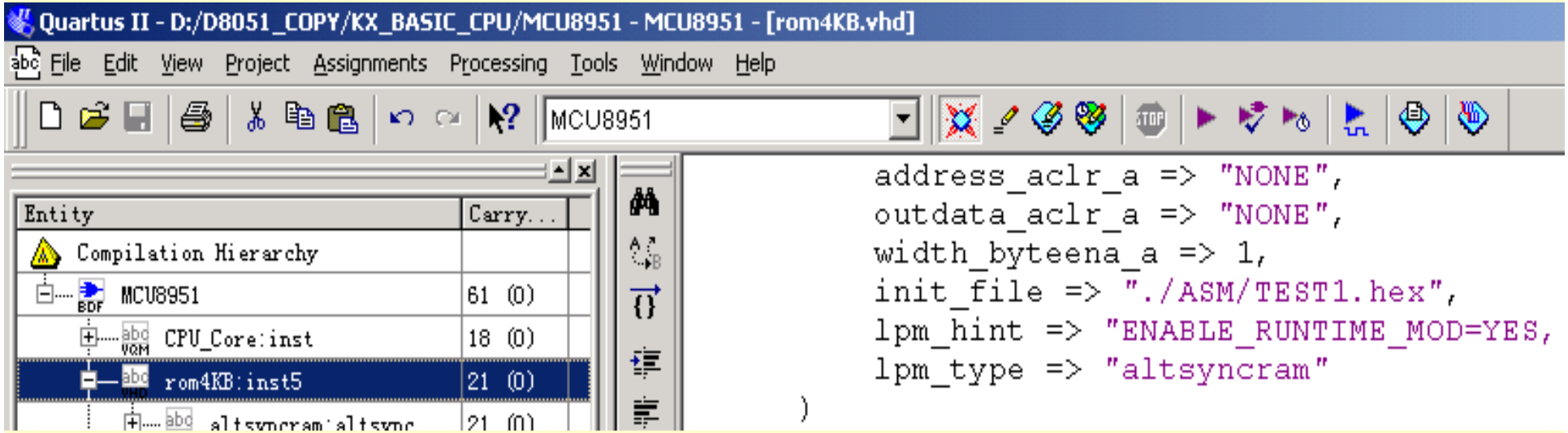


图7-52 LPM_ROM初始化文件路径

```
                ORG  0000H
MAIN  :         MOV     SP,#60H
                MOV     24H,#00H
                MOV     30H,#01H
ROUND:          LCALL    DELAY1
                MOV     A,24H
                INC     A
                MOV     24H,A
                MOV     P1,A
                MOV     A,30H
                RR      A
                MOV     P0,A
                MOV     30H,A
                NOP
                NOP
                MOV     A,P0
                MOV     B,P3
                ADD     A,B
                MOV     P2,A
                LCALL    DELAY1
                SJMP    ROUND
DELAY:          MOV     20H,#0FFH
    W1:         MOV     21H,#0FFH
    W2:         DJNZ    21H,W2
                DJNZ    20H,W1
                RET
DELAY1:         MOV     22H,#08H
     W3:        LCALL   DELAY
                DJNZ    22H,W3
                RET
                END
```

图7-53 TEST1.asm汇编程序

KX 康芯科技

7.9 8051单片机IP核应用

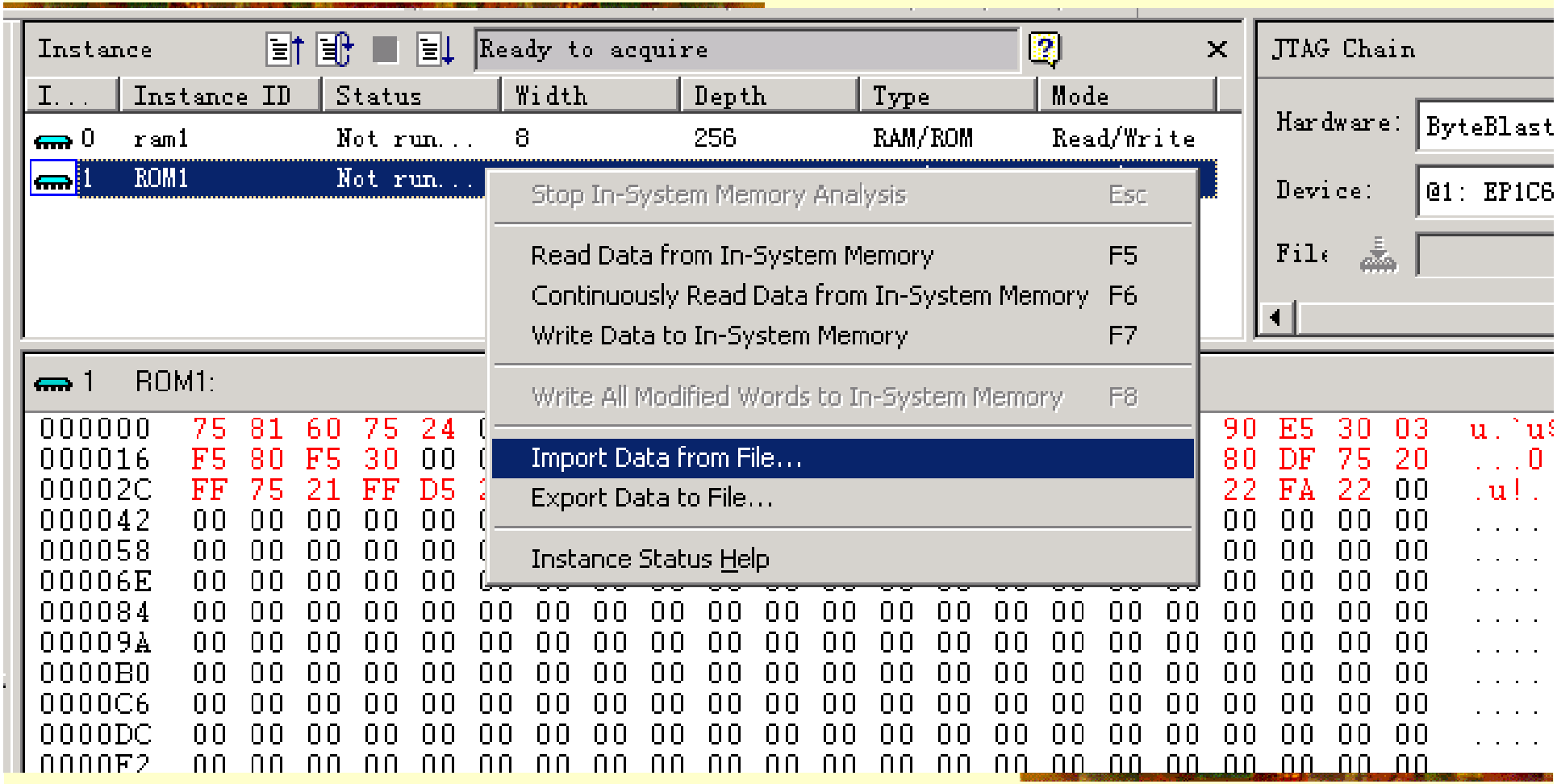


图7-54 下载汇编程序HEX代码

习 题

7-1. 如果不使用MegaWizard Plug-In Manager工具，如何在自己的设计中调用LPM模块？以计数器lpm_counter为例，写出调用该模块的程序，其中参数自定。

7-2. LPM_ROM、LPM_RAM、LPM_FIFO等模块与FPGA中嵌入的EAB，ESB，M4K有怎样的联系关系？

7-3. 参考QuartusII的Help（Contents），详细说明LPM元件altcam、altsyncram、lpm_fifo、lpm_shiftreg的使用方法，以及其中各参量的含义和设置方法。

7-4. 如果要设计一8051单片机，如何为它配置含有汇编程序代码的ROM（文件）？

7-5. 将例**7-4**的顶层程序和例**7-3**的**ROM**程序合并成为一个程序，要求用例化语句直接调用**LPM**模块altsyncram。编译验证，使之功能与原设计相同。

实验与设计

7-1. 正弦信号发生器设计

（1）实验目的：进一步熟悉QuartusII及其LPM_ROM与FPGA硬件资源的使用方法。

（2）实验原理：参考本章相关内容。

（3）实验内容1：根据例7-4，在Quartus II上完成正弦信号发生器设计，包括仿真和资源利用情况了解（假设利用Cyclone器件）。最后在实验系统上实测，包括SignalTap II测试、FPGA中ROM的在系统数据读写测试和利用示波器测试。最后完成EPCSx配置器件的编程。

（4）实验内容2：按照图7-49所示，用原理图方法设计正弦信号发生器，要调用3个LPM模块来构成：1、PLL，输入频率20MHz，32MHz单频率输出；2、6位二进制计数器；3、LPM ROM，加载的波形数据同上。注意，硬件实现时可以通过SignalTapII观察波形，但不能用0832输出，波形必须用高速DAC输出。

实验与设计

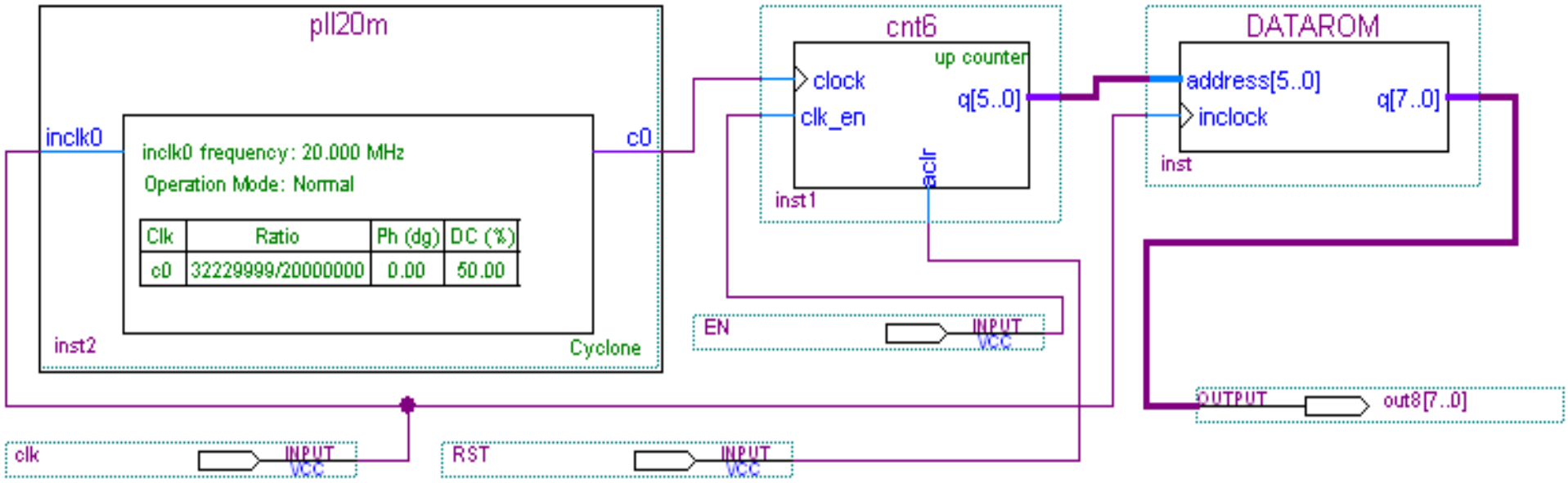


图7-55 调用了PLL元件信号发生器原理图

实验与设计

7-1. 正弦信号发生器设计

（5）实验内容3：修改例7-3的数据ROM文件，设其数据线宽度为8，地址线宽度也为8，初始化数据文件使用MIF格式，用C程序产生正弦信号数据，最后完成以上相同的实验。

（6）实验内容4：设计一任意波形信号发生器，可以使用LPM双口RAM担任波形数据存储器，利用单片机产生所需要的波形数据，然后输向FPGA中的RAM（可以利用GW48系统上与FPGA接口的单片机完成此实验，D/A可利用系统上配置的0832或5651高速器件）。

（7）实验报告：根据以上的实验内容写出实验报告，包括设计原理、程序设计、程序分析、仿真分析、硬件测试和详细实验过程。

实验与设计

7-2. 8位16进制频率计设计

(1) 实验目的：设计8位16进制频率计，学习较复杂的数字系统设计方法。

(2) 实验原理：根据频率的定义和频率测量的基本原理，测定信号的频率必须有一个脉宽为1秒的输入信号脉冲计数允许的信号；1秒计数结束后，计数值被锁入锁存器，计数器清0，为下一测频计数周期作好准备。测频控制信号可以由一个独立的发生器来产生，即图7-57中的FTCTRL。根据测频原理，测频控制时序可以如图7-56所示。

设计要求是：FTCTRL的计数使能信号CNT_EN能产生一个1秒脉宽的周期信号，并对频率计中的32位二进制计数器COUNTER32B（图7-57）的ENABL使能端进行同步控制。当CNT_EN高电平时允许计数；低电平时停止计数，并保持其所计的脉冲数。在停止计数期间，首先需要一个锁存信号LOAD的上跳沿将计数器在前1秒钟的计数值锁存进锁存器REG32B中，并由外部的16进制7段译码器译出，显示计数值。设置锁存器的好处是数据显示稳定，不会由于周期性的清0信号而不断闪烁。锁存信号后，必须有一清0信号RST_CNT对计数器进行清零，为下1秒的计数操作作准备。

实验与设计

7-2. 8位16进制频率计设计

(3) 实验内容1：分别仿真测试模块例7-7、例7-8和例7-9，再结合例7-10完成频率计的完整设计和硬件实现，并给出其测频时序波形及其分析。建议选实验电路模式5；8个数码管以16进制形式显示测频输出；待测频率输入FIN由clock0输入，频率可选4Hz、256HZ、3Hz...50MHz等；1HZ测频控制信号CLK1HZ可由clock2输入(用跳线选1Hz)。注意，这时8个数码管的测频显示值是16进制的。

(4) 实验内容2：参考例4-22，将频率计改为8位10进制频率计，注意此设计电路的计数器必须是8个4位的10进制计数器，而不是1个。此外注意在测频速度上给予优化。

(5) 实验内容3：用LPM模块取代例7-8和例7-9，再完成同样的设计任务。

(6) 实验报告：给出频率计设计的完整实验报告。

【例7-7】

```
LIBRARY IEEE;  --测频控制电路
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY FTCTRL IS
    PORT (CLKK : IN STD_LOGIC;                    -- 1Hz
          CNT_EN : OUT STD_LOGIC;                  -- 计数器时钟使能
         RST_CNT : OUT STD_LOGIC;                  -- 计数器清零
           Load : OUT STD_LOGIC    );               -- 输出锁存信号
 END FTCTRL;
ARCHITECTURE behav OF FTCTRL IS
    SIGNAL Div2CLK : STD_LOGIC;
BEGIN
    PROCESS( CLKK )
    BEGIN
        IF CLKK'EVENT AND CLKK = '1' THEN       -- 1Hz时钟2分频
            Div2CLK <= NOT Div2CLK;
        END IF;
    END PROCESS;
    PROCESS (CLKK, Div2CLK)
    BEGIN
        IF CLKK='0' AND Div2CLK='0' THEN RST_CNT<='1';-- 产生计数器清零信
号
          ELSE RST_CNT <= '0';  END IF;
    END PROCESS;
    Load  <= NOT Div2CLK;    CNT_EN <= Div2CLK;
END behav;
```

KX 康芯科技

【例7-8】

```
LIBRARY IEEE;  --32位锁存器
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY REG32B IS
     PORT (    LK : IN STD_LOGIC;
               DIN : IN STD_LOGIC_VECTOR(31
DOWNTO 0);
              DOUT : OUT STD_LOGIC_VECTOR(31
DOWNTO 0) );
END REG32B;
ARCHITECTURE behav OF REG32B IS
BEGIN
     PROCESS(LK, DIN)
    BEGIN
    IF LK'EVENT AND LK = '1' THEN  DOUT <= DIN;
         END IF;
     END PROCESS;
END behav;
```

【例7-9】

```
LIBRARY IEEE;  --32位计数器
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY COUNTER32B IS
    PORT (FIN : IN STD_LOGIC;                    -- 时钟信号
          CLR : IN STD_LOGIC;                      -- 清零信号
         ENABL : IN STD_LOGIC;                  -- 计数使能信号
          DOUT :  OUT STD_LOGIC_VECTOR(31 DOWNTO 0)); -- 计数结果
   END COUNTER32B;
ARCHITECTURE behav OF COUNTER32B IS
    SIGNAL CQI :  STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
    PROCESS(FIN, CLR, ENABL)
      BEGIN
        IF CLR = '1' THEN   CQI <= (OTHERS=>'0');   -- 清零
        ELSIF FIN'EVENT AND FIN = '1' THEN
            IF ENABL = '1' THEN CQI <= CQI + 1; END IF;
        END IF;
    END PROCESS;
    DOUT <= CQI;
END behav;
```

【例7-10】

```
LIBRARY IEEE;  --频率计顶层文件
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY FREQTEST IS
    PORT ( CLK1HZ : IN STD_LOGIC;
           FSIN : IN STD_LOGIC;
           DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END FREQTEST;
ARCHITECTURE struc OF FREQTEST IS
COMPONENT FTCTRL
    PORT (CLKK : IN STD_LOGIC;          -- 1Hz
        CNT_EN : OUT STD_LOGIC;         -- 计数器时钟使能
       RST_CNT : OUT STD_LOGIC;         -- 计数器清零
         Load : OUT STD_LOGIC    );     -- 输出锁存信号
 END COMPONENT;
COMPONENT COUNTER32B
    PORT (FIN : IN STD_LOGIC;           -- 时钟信号
          CLR : IN STD_LOGIC;           -- 清零信号
        ENABL : IN STD_LOGIC;           -- 计数使能信号
         DOUT :  OUT STD_LOGIC_VECTOR(31 DOWNTO 0)); -- 计数结果
END COMPONENT;
```

接下页

```
  COMPONENT REG32B
      PORT (    LK : IN STD_LOGIC;
                DIN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
               DOUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
  END COMPONENT;
      SIGNAL TSTEN1 : STD_LOGIC;
      SIGNAL CLR_CNT1 : STD_LOGIC;
      SIGNAL Load1 : STD_LOGIC;
      SIGNAL DTO1 : STD_LOGIC_VECTOR(31 DOWNTO 0);
      SIGNAL CARRY_OUT1 : STD_LOGIC_VECTOR(6 DOWNTO 0);
  BEGIN
    U1 :     FTCTRL PORT MAP(CLKK =>CLK1HZ,CNT_EN=>TSTEN1,
  RST_CNT =>CLR_CNT1,Load =>Load1);
    U2 :     REG32B PORT MAP(  LK => Load1,   DIN=>DTO1, DOUT =>
DOUT);
    U3 : COUNTER32B PORT MAP( FIN => FSIN, CLR => CLR_CNT1,
   ENABL => TSTEN1, DOUT=>DTO1 );
  END struc;
```

实验与设计

7-2. 8位16进制频率计设计

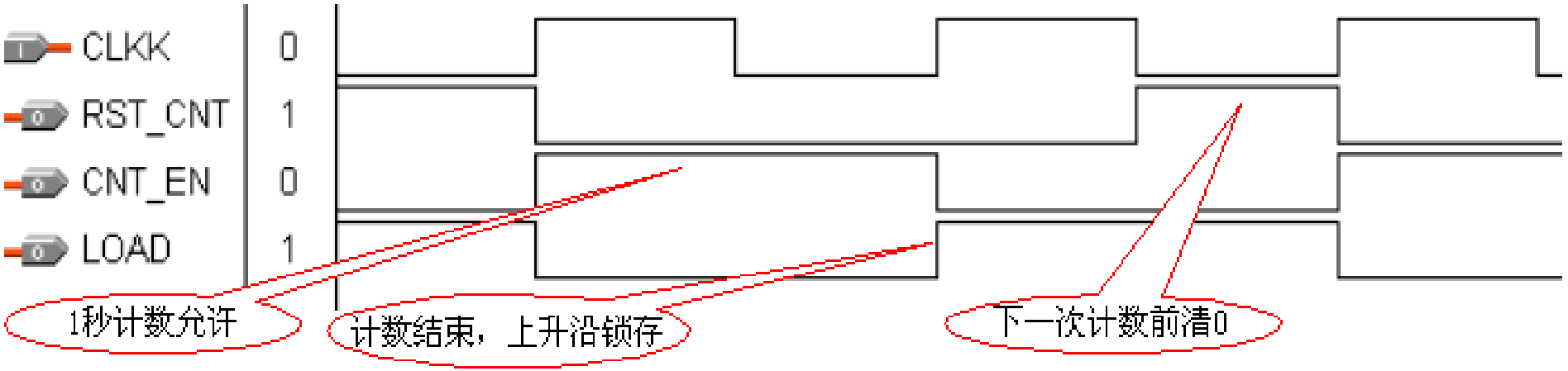


图7-56 频率计测频控制器FTCTRL测控时序图

实验与设计

7-2. 8位16进制频率计设计

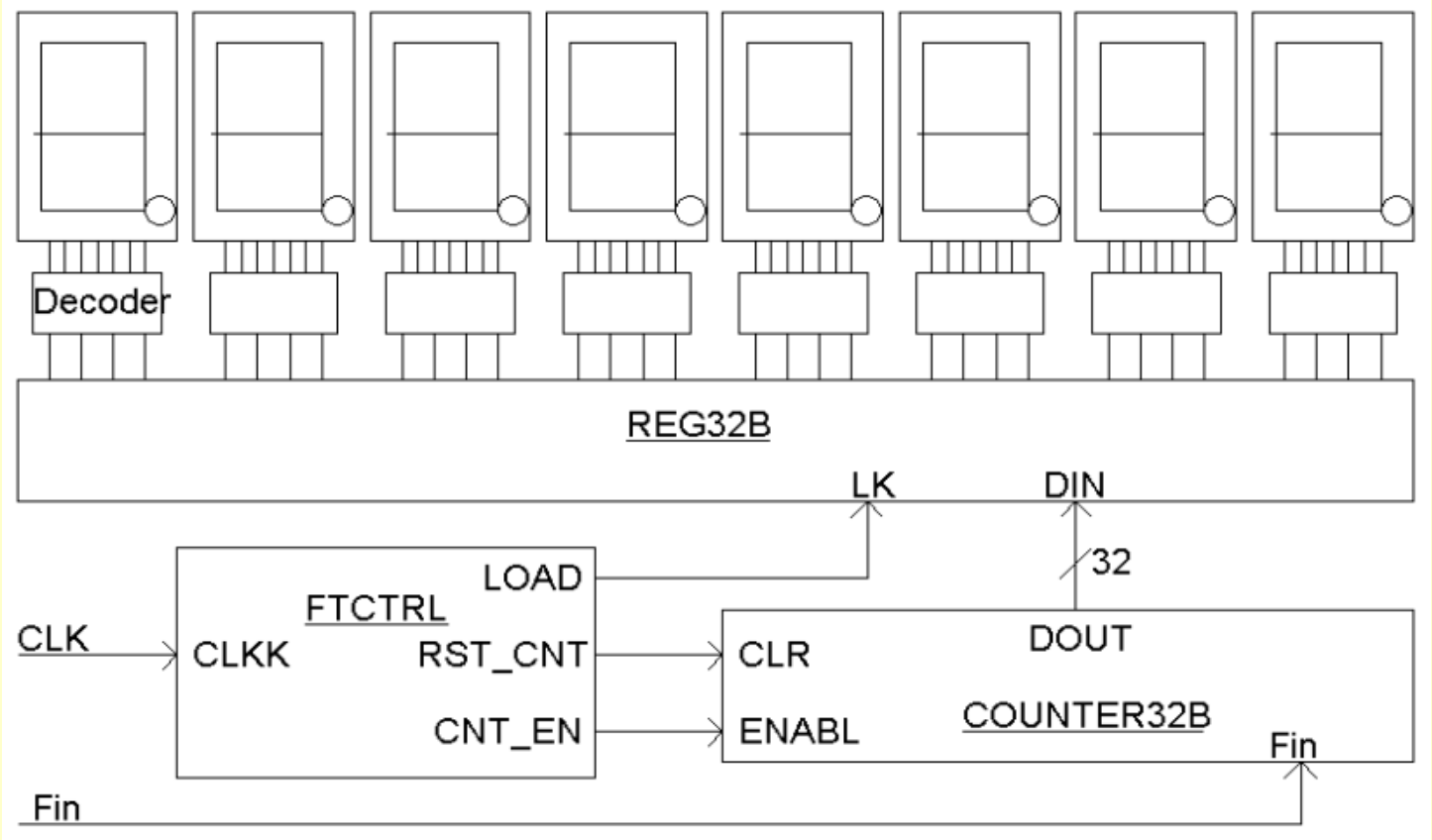


图7-57 频率计电路框图

实验与设计

7-3. 利用LPM_ROM设计乘法器

(1) 实验原理：硬件乘法器有多种设计方法，但相比之下，由**LPM_ROM**构成的乘法表方式的乘法器的运算速度最快。这里定制**LPM_ROM**的地址位宽为**8**；地址输入由时钟**inclock**的上升沿锁入；数据位宽也为**8**。最后为**ROM**配置乘法表数据文件。

LPM_ROM中作为乘法表的数据文件**rom_data.mif**如例**7-11**所示。其中的地址/数据表达方式是，冒号左边写**ROM**地址值，冒号右边写对应此地址放置的**16**进制数据。如**47**：**28**，表示**47**为地址，**28**为该地址中的数据，这样，地址高**4**位和低**4**位可以分别看成是乘数和被乘数，输出的数据可以看成是它们的乘积。

【例7-11】

```
WIDTH = 8 ;
DEPTH = 256 ;
ADDRESS_RADIX = HEX ;
DATA_RADIX = HEX ;
CONTENT  BEGIN
 00:00 ; 01:00 ; 02:00 ; 03:00 ; 04:00 ; 05:00 ; 06:00 ; 07:00 ; 08:00 ; 09:00;
 10:00 ; 11:01 ; 12:02 ; 13:03 ; 14:04 ; 15:05 ; 16:06 ; 17:07 ; 18:08 ; 19:09;
 20:00 ; 21:02 ; 22:04 ; 23:06 ; 24:08 ; 25:10 ; 26:12 ; 27:14 ; 28:16 ; 29:18;
 30:00 ; 31:03 ; 32:06 ; 33:09 ; 34:12 ; 35:15 ; 36:18 ; 37:21 ; 38:24 ; 39:27;
 40:00 ; 41:04 ; 42:08 ; 43:12 ; 44:16 ; 45:20 ; 46:24 ; 47:28 ; 48:32 ; 49:36;
 50:00 ; 51:05 ; 52:10 ; 53:15 ; 54:20 ; 55:25 ; 56:30 ; 57:35 ; 58:40 ; 59:45;
 60:00 ; 61:06 ; 62:12 ; 63:18 ; 64:24 ; 65:30 ; 66:36 ; 67:42 ; 68:48 ; 69:54;
 70:00 ; 71:07 ; 72:14 ; 73:21 ; 74:28 ; 75:35 ; 76:42 ; 77:49 ; 78:56 ; 79:63;
 80:00 ; 81:08 ; 82:16 ; 83:24 ; 84:32 ; 85:40 ; 86:48 ; 87:56 ; 88:64 ; 89:72;
 90:00 ; 91:09 ; 92:18 ; 93:27 ; 94:36 ; 95:45 ; 96:54 ; 97:63 ; 98:72 ; 99:81;
END ;
```

注意，以上“**CONTENT BEGIN**”下所示的数据格式只是为了节省篇幅，实用中应该使每一数据组（如**01:00 ;**）占一行。

实验与设计

7-3. 利用LPM_ROM设计乘法器

(2) 实验内容： 利用**LPM_ROM**设计**4X4**和**8X8**乘法器各一个，再利用**VHDL**语言描述，由逻辑宏单元构成同类乘法器各一，比较这两类乘法器的运行速度和资源耗用情况。

实验与设计

7-4. IP核应用实验

利用**IP**核完成如下**2**项设计：

1、利用**NCO**核分别设计：

（**1**）**FSK**；（**2**）**PSK**；（**3**）**DDS**；（**4**）移相信号发生器；　（**5**）扫频信号源；（**6**）全数字式锁相环。

2、利用**NCO**和**FIR**核设计数字正交调制解调器（参考清华大学出版社《**SOPC**技术实用教程》中的实验**6-5**）。

实验与设计

7-5. 8051单片机IP核应用实验

(1) 实验内容1：参考**7.9**节，在图**7-49**所示的基本电路平台上增加一些**LPM**或**VHDL**表述硬件模块（如锁存器、译码器、**PWM**发生器、**A/D**采样控制模块、液晶控制模块等），及与单片机的接口电路，利用单片机进行控制，再编辑对应的汇编软件，完成进一步的实验。

(2) 实验内容2：选择不同模式，和引脚锁定情况，协调软件与硬件设计，完成较大的软硬件综合设计模块。

(3) 实验内容3：编辑一段用于测试的汇编程序，利用时序仿真和逻辑分析仪，了解**8051**单片机的数据总线、指令总线、不同指令执行、地址总线、**ALE**、**PSEN**、个**IO**端口、不同指令对应下的端口方向控制信号**P0E**、**P1E**、**P2E**、**P3E**等信号间的时序情况，给出分析报告。

6.1.4 进程中的信号与变量赋值

【例6-3】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DFF3 IS
PORT ( CLK,D1 : IN STD_LOGIC ;
             Q1  : OUT STD_LOGIC  ) ;
END ;
 ARCHITECTURE bhv OF DFF3 IS
   SIGNAL A,B : STD_LOGIC ;
  BEGIN
  PROCESS (CLK) BEGIN
    IF  CLK'EVENT AND CLK = '1'  THEN
      A <= D1 ;
 B <= A ;
Q1 <= B ;
END IF;
  END PROCESS ;
END;
```

6.1 数据对象

6.1.4 进程中的信号与变量赋值

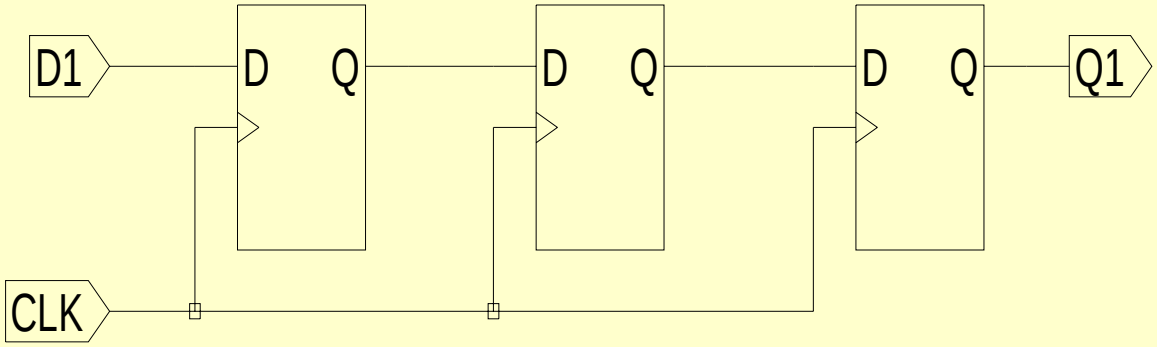


图6-1 例6-3的RTL电路

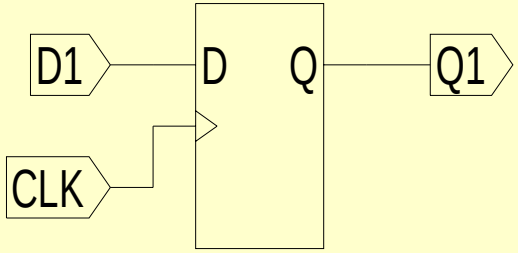


图6-2 D触发器电路

6.1.4 进程中的信号与变量赋值

【例6-4】

```
LIBRARY IEEE ;
 USE IEEE.STD_LOGIC_1164.ALL ;
 ENTITY DFF3 IS
 PORT ( CLK,D1 : IN STD_LOGIC ;
                Q1 : OUT STD_LOGIC  ) ;
 END ;
  ARCHITECTURE bhv OF DFF3 IS
   BEGIN
   PROCESS (CLK)
    VARIABLE A,B : STD_LOGIC ;
    BEGIN
     IF  CLK'EVENT AND CLK = '1'  THEN
       A := D1 ;
 B := A ;
Q1 <= B ;
     END IF;
  END PROCESS ;
    END ;
```

6.1 数据对象

6.1.4 进程中的信号与变量赋值

【例6-5】

```
SIGNAL  in1，in2，e1，  ... :  STD_LOGIC ;
...
PROCESS(in1，in2，  . . .)
VARIABLE c1，. . . : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
  BEGIN
    IF in1 = '1' THEN ...                  -- 第 1 行
        e1 <= "1010" ;                       -- 第 2 行
             ...
    IF in2 = '0' THEN . . .                -- 第 15+n 行
             ...
        c1 := "0011" ;                       -- 第 30+m 行
             ...
    END IF;
END PROCESS;
```

【例6-6】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;
ARCHITECTURE body_mux4 OF mux4 IS
signal muxval : integer range 7 downto 0;
BEGIN
process(i0,i1,i2,i3,a,b)
begin
                        muxval <= 0;
if (a = '1') then   muxval <= muxval + 1; end if;
if (b = '1') then   muxval <= muxval + 2; end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
     when others => null;
end case;
end process;
END body_mux4;
```

【例6-7】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;
ARCHITECTURE body_mux4 OF mux4 IS
BEGIN
process(i0,i1,i2,i3,a,b)
variable muxval : integer range 7 downto 0;
begin
                        muxval := 0;
if (a = '1') then   muxval := muxval + 1;  end if;
if (b = '1') then   muxval := muxval + 2;  end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
     when others => null;
end case;
end process;
END body_mux4;
```

6.1.4 进程中的信号与变量赋值

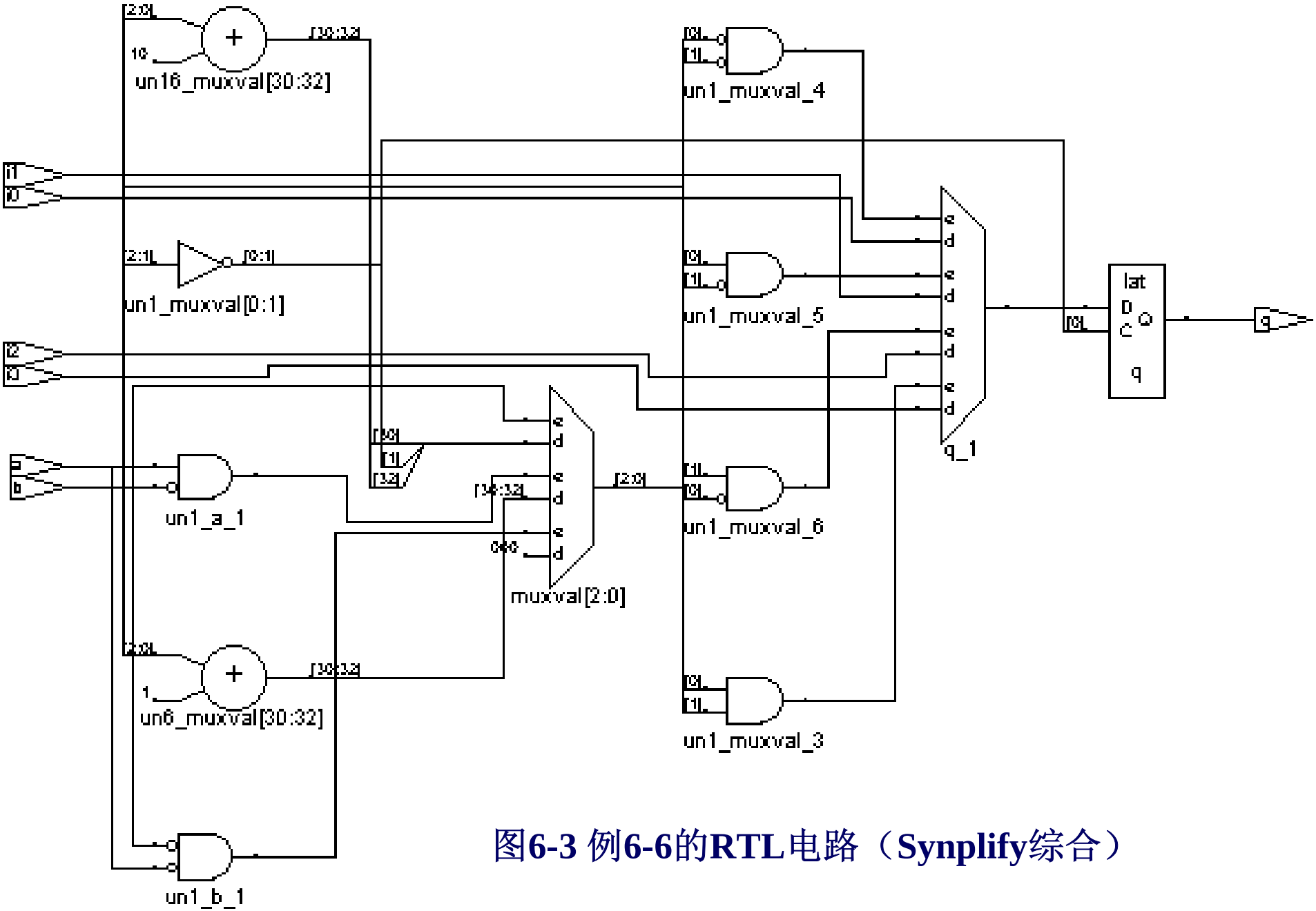


图6-3 例6-6的RTL电路（Synplify综合）

6.1.4 进程中的信号与变量赋值

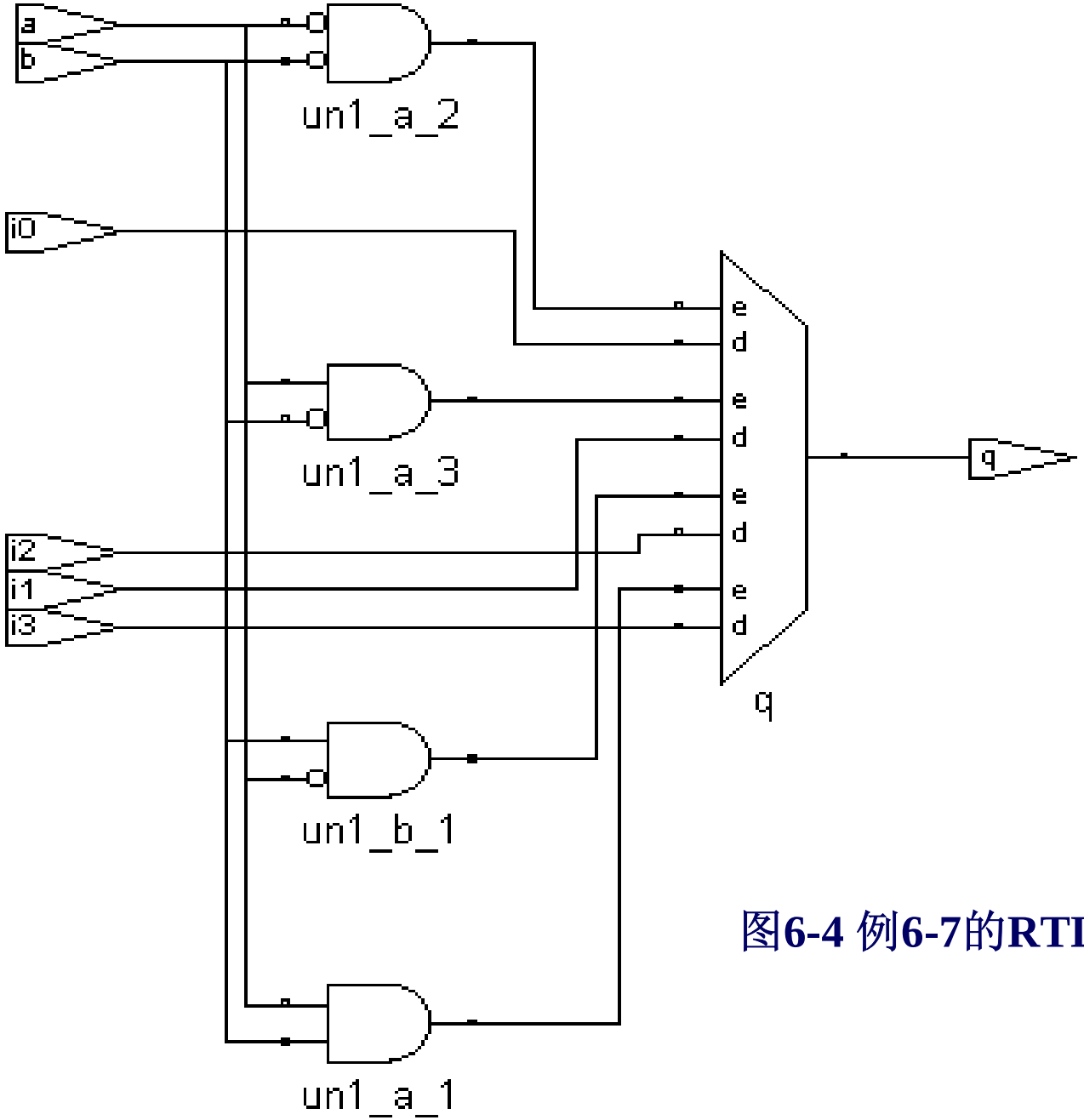


图6-4 例6-7的RTL电路（Synplify综合）

6.1 数据对象

6.1.4 进程中的信号与变量赋值

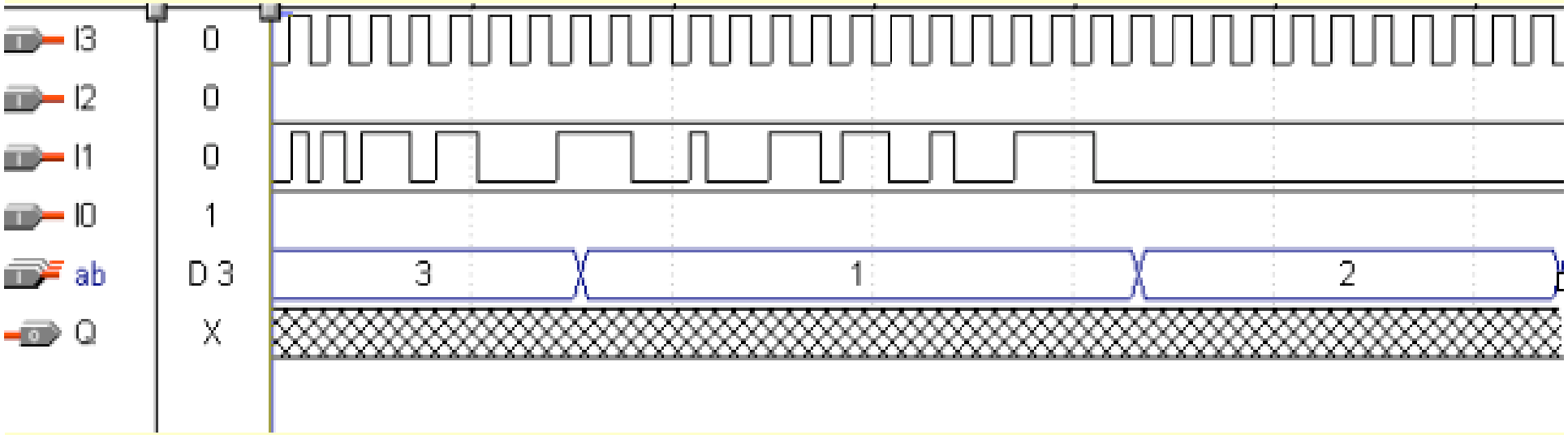


图6-5 例6-6中错误的工作时序

6.1 数据对象

6.1.4 进程中的信号与变量赋值

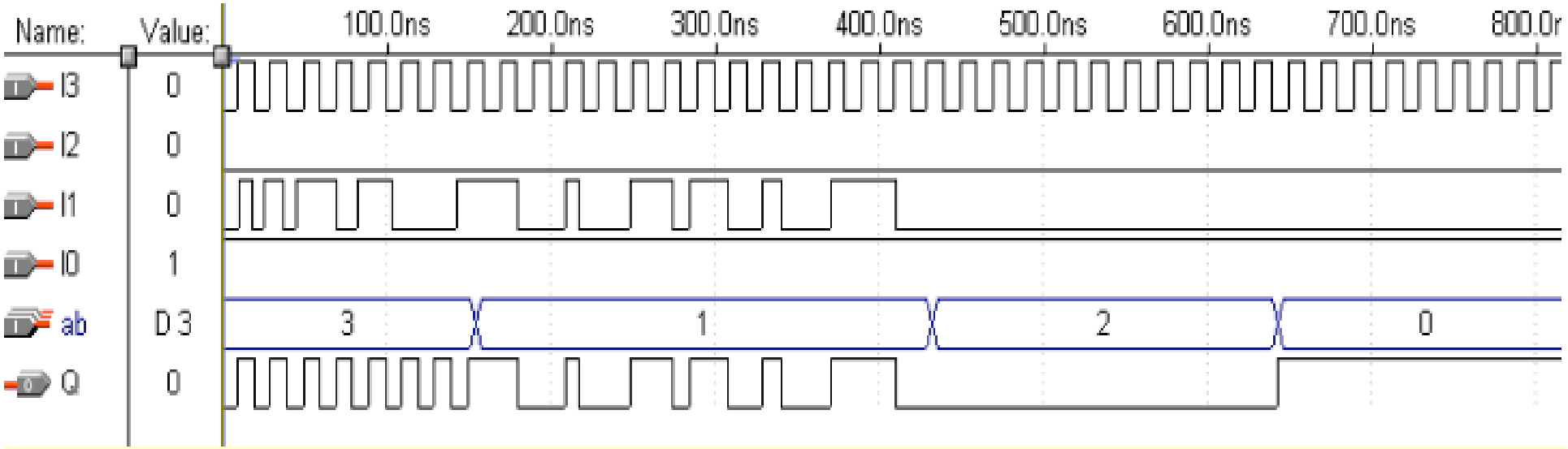


图6-6 例6-7中正确的工作时序

【例6-8】

```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY SHIFT IS
  PORT (CLK,C0 :  IN STD_LOGIC;  --时钟和进位输入
        MD :  IN STD_LOGIC_VECTOR(2 DOWNTO 0); --移位模式控制字
        D  :  IN STD_LOGIC_VECTOR(7 DOWNTO 0); --待加载移位的数据
        QB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --移位数据输出
        CN : OUT STD_LOGIC);                    --进位输出
END ENTITY;
ARCHITECTURE BEHAV OF SHIFT IS
       SIGNAL REG  : STD_LOGIC_VECTOR(7 DOWNTO 0);
       SIGNAL   CY : STD_LOGIC ;
 BEGIN
PROCESS (CLK,MD,C0)
   BEGIN
    IF CLK'EVENT AND CLK = '1' THEN
      CASE MD IS
       WHEN "001" =>     REG(0) <= C0 ;
REG(7 DOWNTO 1) <= REG(6 DOWNTO 0); CY <= REG(7);--带进位循环左移

       WHEN “010” =>     REG(0) <= REG(7);
```

（接下页）

KX 康芯科技

（接上页）

```
  REG(7 DOWNTO 1) <= REG(6 DOWNTO 0);                --自循环左移
         WHEN "011" =>      REG(7) <= REG(0);
  REG(6 DOWNTO 0) <= REG(7 DOWNTO 1);                --自循环右移
         WHEN "100" =>      REG(7) <= C0 ;
  REG(6 DOWNTO 0) <= REG(7 DOWNTO 1); CY <= REG(0); --带进位循环右
移
         WHEN "101" =>  REG(7 DOWNTO 0) <=   D(7 DOWNTO 0); --加载
待移数
          WHEN OTHERS => REG <= REG ;  CY <= CY ;           --保持
       END CASE;
     END IF;
    END PROCESS;
        QB(7 DOWNTO 0) <= REG(7 DOWNTO 0); CN <= CY;     --移位后输出
  END BEHAV;
```

6.1 数据对象

6.1.4 进程中的信号与变量赋值

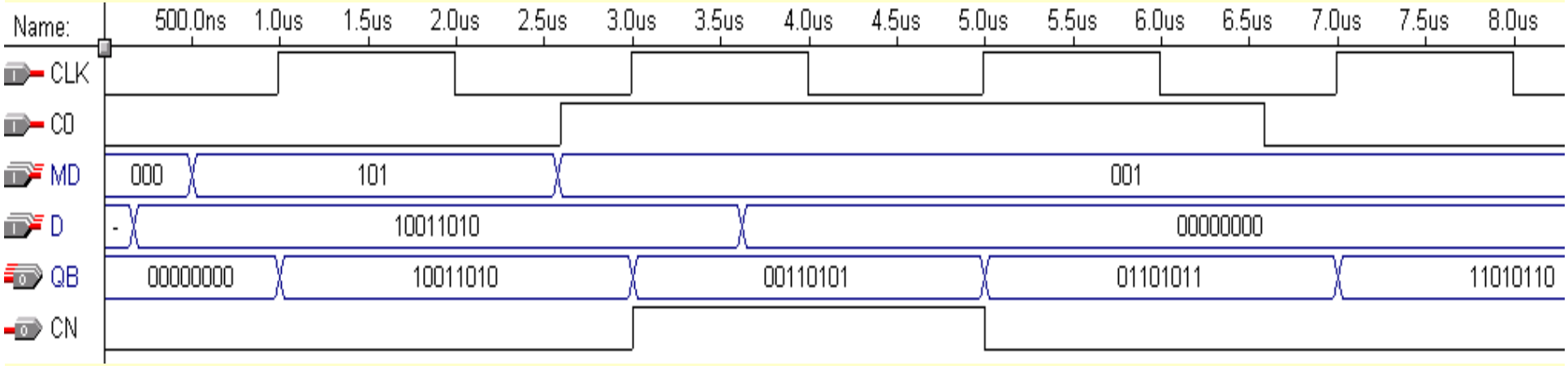


图6-7 例6-8中带进位循环左移仿真波形（MD="001"）

6.2 双向和三态电路信号赋值例解

6.2.1 三态门设计

【例6-9】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tri_s IS
    port (     enable : IN STD_LOGIC;
             datain : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            dataout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END tri_s ;
ARCHITECTURE bhv OF tri_s IS
BEGIN
PROCESS(enable,datain)
   BEGIN
   IF enable = '1' THEN dataout <= datain ;
     ELSE dataout <="ZZZZZZZZ" ;
   END IF ;
END PROCESS;
END bhv;
```

6.2 双向和三态电路信号赋值例解

6.2.1 三态门设计

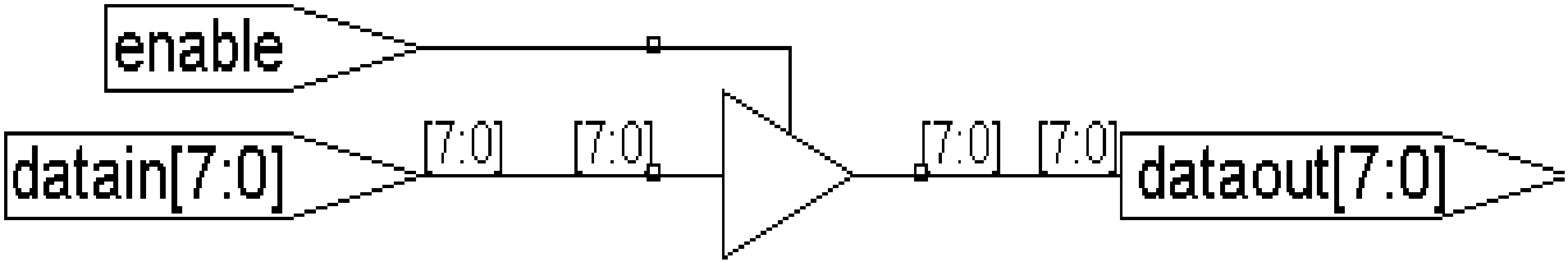


图6-8 8位3态控制门电路（Synplify综合）

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

【例6-10】

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_state is
port (control : in std_logic;
      in1: in std_logic_vector(7 downto 0);
       q : inout std_logic_vector(7 downto 0);
       x : out std_logic_vector(7 downto 0));
end tri_state;
architecture body_tri of tri_state is
begin
process(control,q,in1)
begin
if (control = '0') then   x <= q  ;
else      q <= in1; x<="ZZZZZZZZ" ;
end if;
end process;
end body_tri;
```

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

【例6-11】

(以上部分同上例)

```
process(control,q,in1)
begin
if (control='0') then  x <= q ; q <= "ZZZZZZZZ";
                       else  q <= in1; x <="ZZZZZZZZ";
end if;
end process;
end body_tri;
```

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

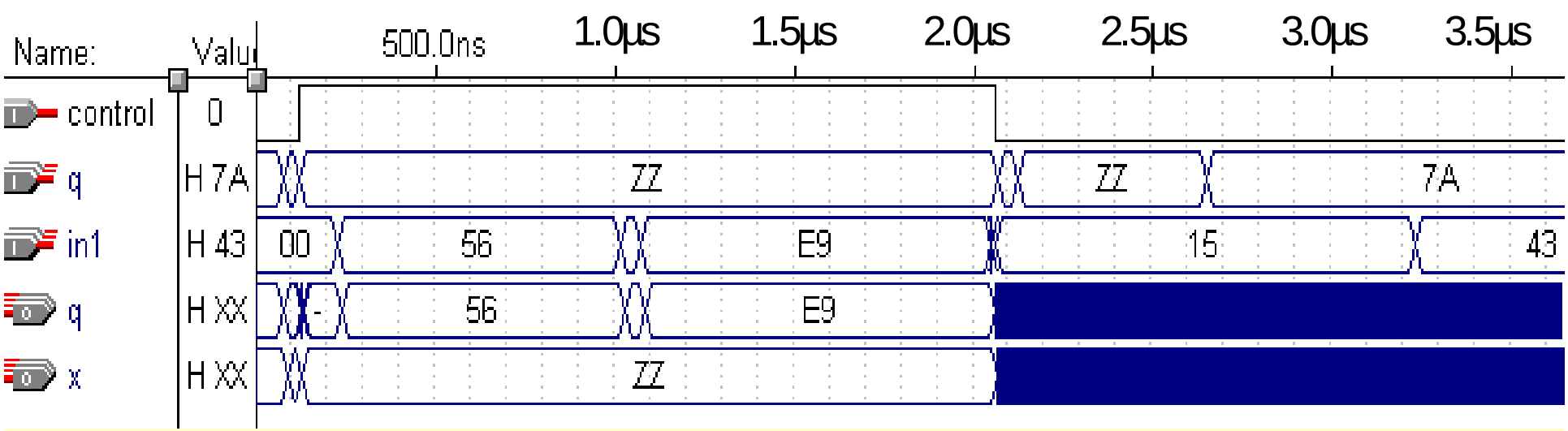


图6-9 例6-10的仿真波形图

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

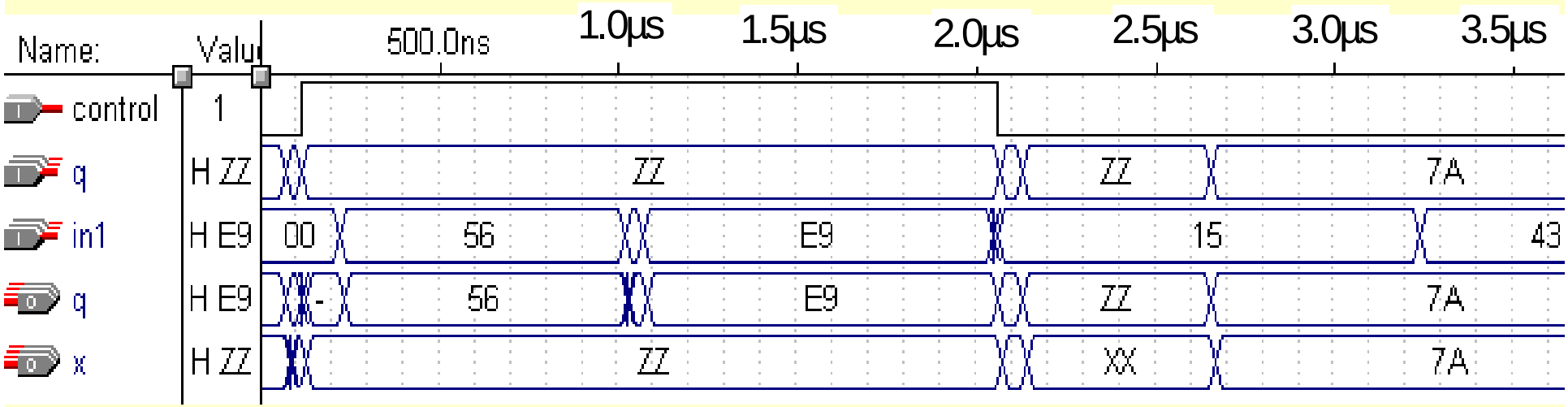


图6-10 例6-11的仿真波形图

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

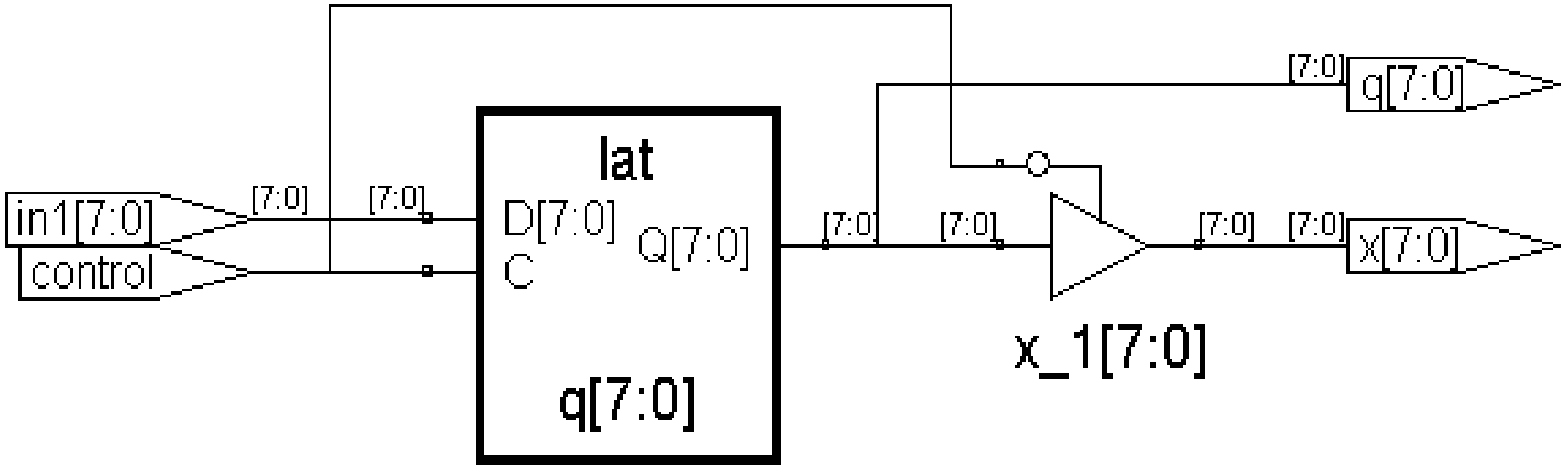


图6-11 例6-10的综合结果（Synplify综合）

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

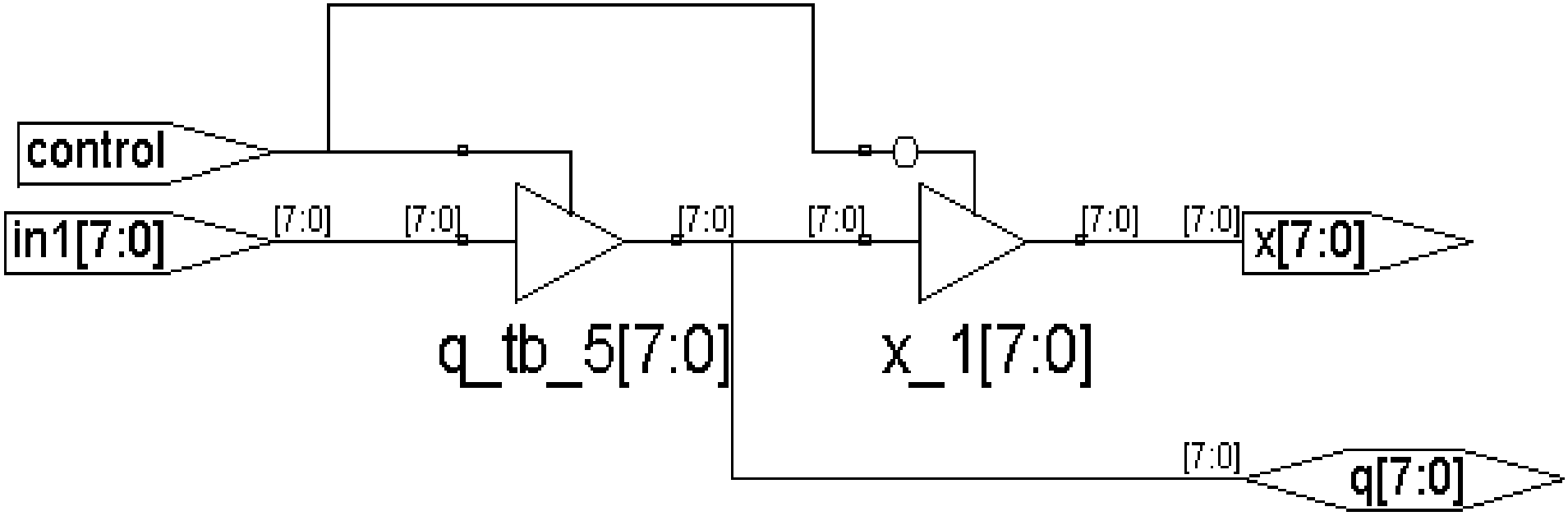


图6-12 例6-11的综合结果（Synplify综合）

6.2.3 三态总线电路设计

【例6-12】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY tristate2 IS
    port ( input3, input2, input1, input0 :
                IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    enable : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    output : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END tristate2 ;
ARCHITECTURE multiple_drivers OF tristate2 IS
BEGIN
PROCESS(enable,input3, input2, input1, input0 )
   BEGIN
   IF enable = "00" THEN output <= input3 ;
       ELSE output <=(OTHERS => 'Z');
   END IF ;
   IF enable = "01" THEN output <= input2 ;
       ELSE output <=(OTHERS => 'Z');
   END IF ;
```

（接下页）

6.2.3 三态总线电路设计

（接上页）

```
IF enable = "10" THEN output <= input1 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
    IF enable = "11" THEN output <= input0 ;
        ELSE output <=(OTHERS => 'Z');
    END IF ;
END PROCESS;
END multiple_drivers;
```

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

【例6-13】（注：MaxplusII不支持本例）

```
library ieee;
use ieee.std_logic_1164.all;
entity tri2 is
port (ctl : in  std_logic_vector(1 downto 0);
         datain1，  datain2，datain3，  datain4 :
              in std_logic_vector(7 downto 0);
        q : out std_logic_vector(7 downto 0)   );
end tri2;
architecture body_tri of tri2 is
begin
 q <= datain1   when ctl="00" else  (others =>'Z') ;
 q <= datain2   when ctl="01" else  (others =>'Z') ;
 q <= datain3   when ctl="10" else  (others =>'Z') ;
 q <= datain4   when ctl="11" else  (others =>'Z') ;
end body_tri;
```

6.2 双向和三态电路信号赋值例解

6.2.2 双向端口设计

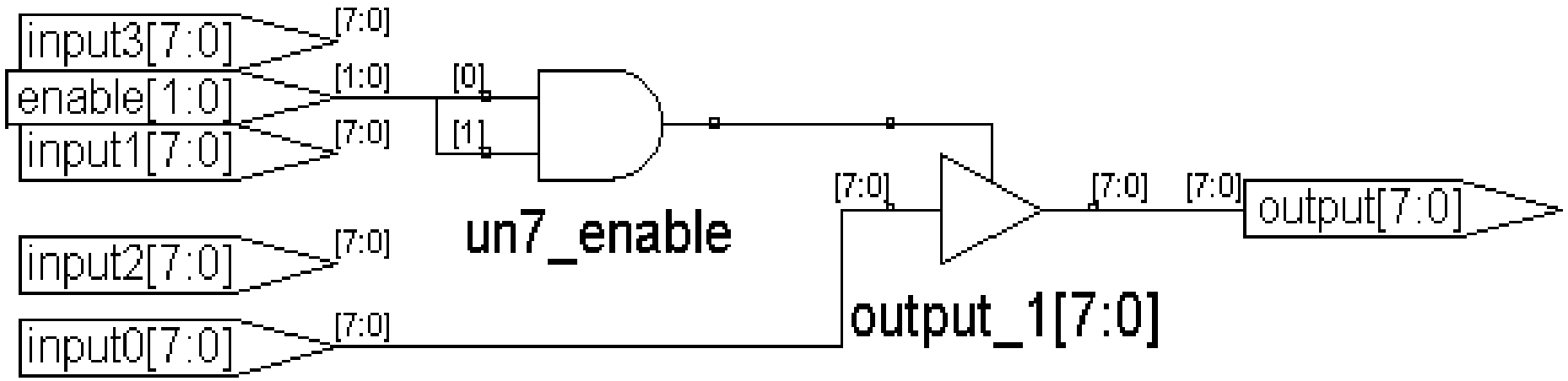


图6-13 例6-12错误的综合结果（Synplify综合结果）

6.2.2 双向端口设计

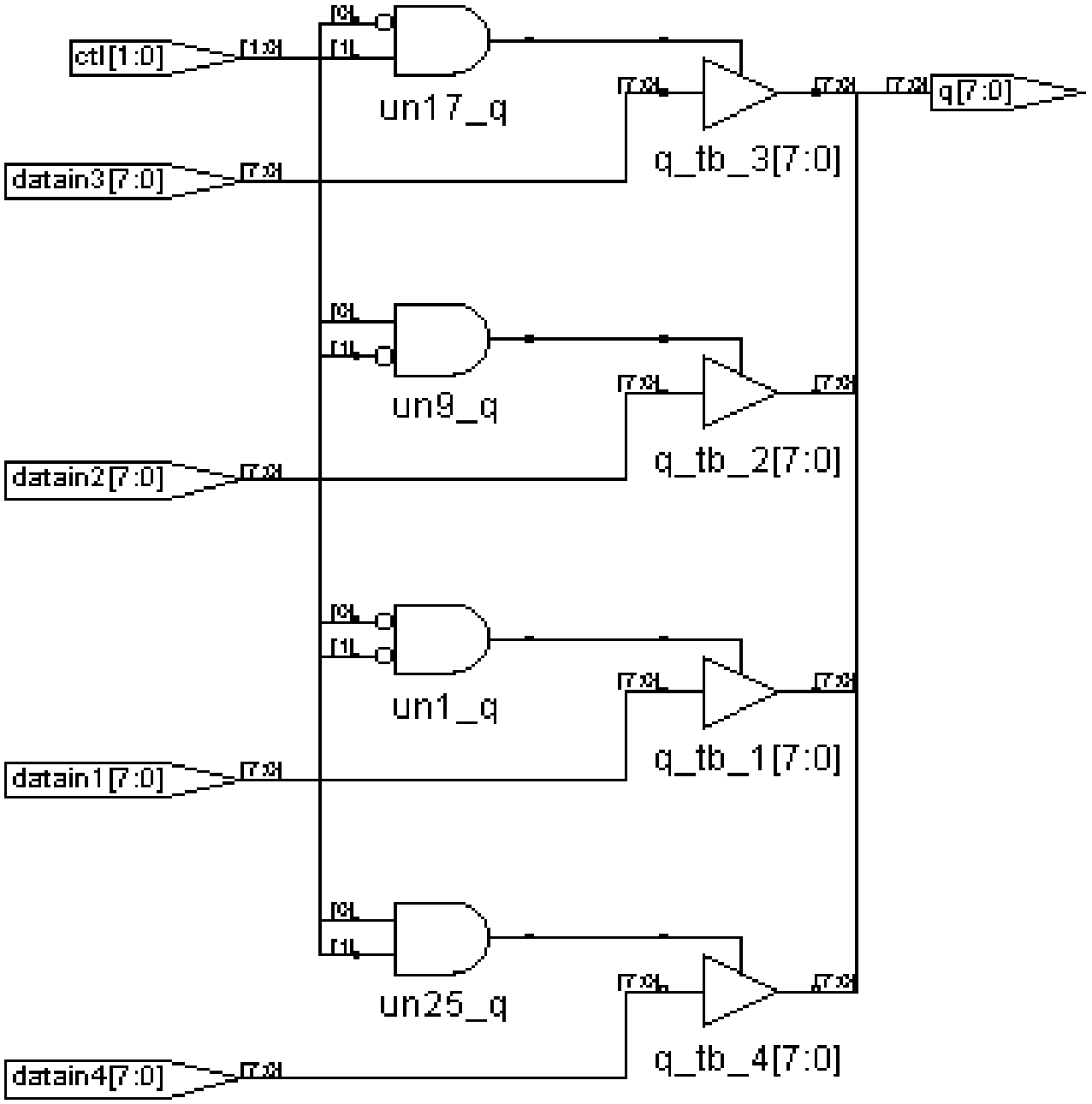


图6-14 例6-13正确的综合结果（Synplify综合结果）

6.3 IF语句概述

(1) IF 条件句 Then
顺序语句
END IF ;

(2) IF 条件句 Then
顺序语句
ELSE
顺序语句
END IF ;

(3) IF 条件句 Then
IF 条件句 Then
...
END IF
END IF

(4) IF 条件句 Then
顺序语句
ELSIF 条件句 Then
顺序语句
...
ELSE
顺序语句
END IF

6.3 IF语句概述

【例6-14】

```
  LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;
  ENTITY control_stmts IS
PORT (a, b, c: IN BOOLEAN;
        output: OUT BOOLEAN);
  END control_stmts;
  ARCHITECTURE example OF control_stmts IS
  BEGIN
    PROCESS (a, b, c)
      VARIABLE n: BOOLEAN;
    BEGIN
      IF  a  THEN   n := b;   ELSE   n := c;
      END IF;
      output <= n;
    END PROCESS;
END example;
```

6.3 IF语句概述

表6-2 8线-3线优先编码器真值表

输入								输出		
din0	din1	din2	din3	din4	din5	din6	din7	output0	output1	output2
x	x	x	x	x	x	x	0	0	0	0
x	x	x	x	x	x	0	1	1	0	0
x	x	x	x	x	0	1	1	0	1	0
x	x	x	x	0	1	1	1	1	1	0
x	x	x	0	1	1	1	1	0	0	1
x	x	0	1	1	1	1	1	1	0	1
x	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	1

注：表中的“x”为任意，类似VHDL中的“—”值。

【例6-15】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY coder IS
  PORT (    din : IN STD_LOGIC_VECTOR(0 TO 7);
          output : OUT STD_LOGIC_VECTOR(0 TO 2)  );
END coder;
ARCHITECTURE behav OF coder IS
  SIGNAL SINT : STD_LOGIC_VECTOR(4 DOWNTO 0);
  BEGIN
    PROCESS (din)
  BEGIN
       IF (din(7)='0') THEN  output <= "000" ;
    ELSIF (din(6)='0') THEN  output <= "100" ;
    ELSIF (din(5)='0') THEN  output <= "010" ;
    ELSIF (din(4)='0') THEN  output <= "110" ;
    ELSIF (din(3)='0') THEN  output <= "001" ;
    ELSIF (din(2)='0') THEN  output <= "101" ;
    ELSIF (din(1)='0') THEN  output <= "011" ;
                             ELSE  output <= "111" ;
  END IF ;
    END PROCESS ;
 END behav;
```

6.4 进程语句归纳

6.4.1 进程语句格式

PROCESS语句结构的一般表达格式如下

```
[进程标号: ] PROCESS [ ( 敏感信号参数表  ) ] [IS]
[进程说明部分]
 BEGIN
      顺序描述语句
    END PROCESS [进程标号];
```

6.4 进程语句归纳

6.4.2 进程结构组成

进程说明部分 → 数据类型、常数、变量、属性、子程序

顺序描述语句部分 → 赋值语句、进程启动语句、子程序调用语句、顺序描述语句、进程跳出语句

敏感信号参数表

6.4 进程语句归纳

6.4.3 进程要点

1. **PROCESS为一无限循环语句**
2. **PROCESS中的顺序语句具有明显的顺序/并行运行双重性**

```
PROCESS(abc)
  BEGIN
   CASE abc IS
    WHEN "0000" => so<="010" ;
    WHEN "0001" => so<="111" ;
    WHEN "0010" => so<="101" ;
    . . .
    WHEN "1110" => so<="100" ;
    WHEN "1111" => so<="000" ;
    WHEN OTHERS => NULL ;
   END CASE;
END PROCESS;
```

6.4 进程语句归纳

6.4.3 进程要点

3. 进程必须由敏感信号的变化来启动

4. 进程语句本身是并行语句

【例6-16】

```
ENTITY mul IS
PORT (a, b, c, selx, sely  : IN BIT;
                     data_out  : OUT BIT  );
END mul;
ARCHITECTURE ex OF mul IS
  SIGNAL  temp : BIT;
BEGIN
p_a : PROCESS (a, b, selx)
      BEGIN
       IF (selx = '0') THEN  temp <= a; ELSE  temp <= b;
      END IF;
  END PROCESS p_a;
   p_b:  PROCESS(temp, c, sely)
      BEGIN
    IF (sely = '0')  THEN  data_out <= temp; ELSE
data_out <= c;
    END IF;
  END PROCESS p_b;
END ex;
```

6.4 进程语句归纳

6.4.3 进程要点

5. 信号是多个进程间的通信线

6. 一个进程中只允许描述对应于一个时钟信号的同步时序逻辑

6.5 并行语句例解

【例6-17】

```
ARCHITECTURE dataflow OF mux IS
SIGNAL seiect :  INTEGER RANGE 15 DOWNTO 0;
BEGIN
Select <= 0 WHEN s0='0' AND s1='0' ELSE
          1 WHEN s0='1' AND s1='0' ELSE
          2 WHEN s0='0' AND s1='1' ELSE
          3 ;
     x <= a WHEN select=0  ELSE
          b WHEN select=1  ELSE
          c WHEN select=2  ELSE
          d ;
. . .
```

6.6 仿真延时

6.6.1 固有延时

```
z <= x XOR y AFTER 5ns ;

z <= x XOR y ;

B <=  A AFTER 20ns ; --固有延时模型
```

6.6 仿真延时

6.6.2 传输延时

```
B <= TRANSPORT A AFTER 20 ns;--
```
传输延时模型

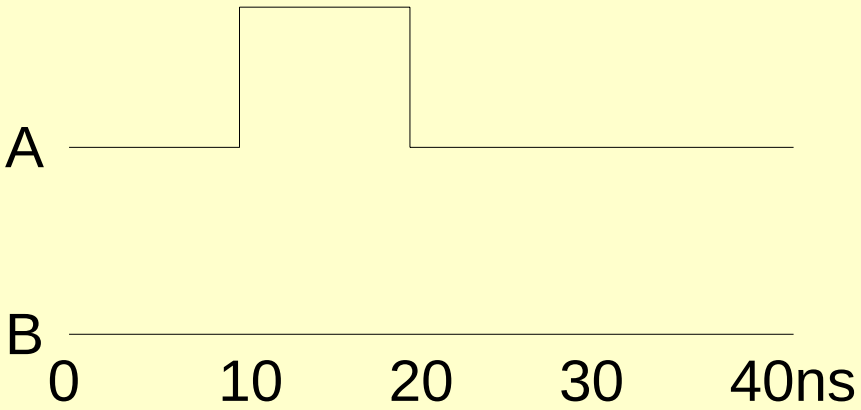


图6-15 固有延时输入输出波形

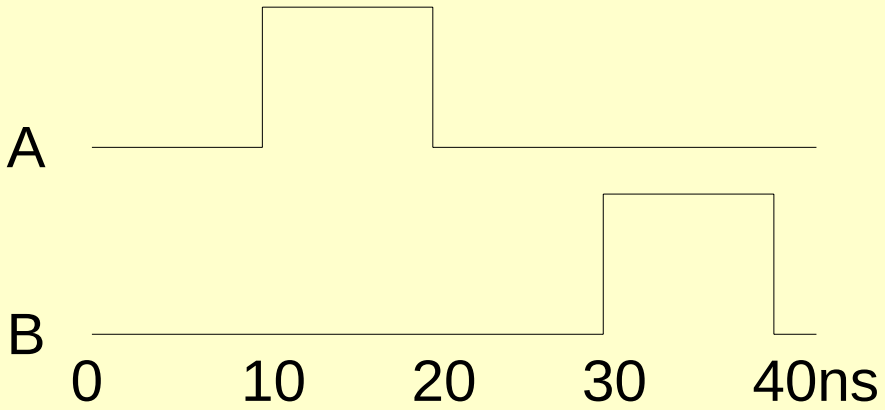


图6-16传输延时输入输出波形

6.6 仿真延时

6.6.3 仿真 δ

VHDL仿真器和综合器将自动为系统中的信号赋值配置一足够小而又能满足逻辑排序的延时量，即仿真软件的最小分辨时间，这个延时量就称为仿真δ（**Simulation Delta**），或称 δ 延时，从而使并行语句和顺序语句中的并列赋值逻辑得以正确执行。由此可见，在行为仿真、功能仿真乃至综合中，引入 δ 延时是必需的。仿真中，δ 延时的引入由**EDA**工具自动完成，无需设计者介入。

习 题

6-1. 什么是固有延时？什么是惯性延时？

6-2. δ 是什么？在**VHDL**中，δ 有什么用处？

6-3. 哪些情况下需要用到程序包**STD_LOGIC_UNSIGNED**？试举一例。

6-4. 说明信号和变量的功能特点，应用上的异同点。

6-5. 在**VHDL**设计中，给时序电路清**0**(复位)有两种方法，它们是什么？

6-6. 哪一种复位方法必须将复位信号放在敏感信号表中？给出这两种电路的**VHDL**描述。

6-7. 什么是重载函数？重载算符有何用处？如何调用重载算符函数？

习 题

6-8. 判断下面3个程序中是否有错误，若有则指出错误所在，并给出完整程序。

程序1：

```
Signal A, EN : std_logic;
Process (A, EN)
    Variable B : std_logic;
Begin
if EN = 1 then   B <= A;  end if;
end process;
```

程序2：

```
Architecture one of sample is
    variable a, b, c : integer;
begin
    c <= a + b;
end;
```

习 题

程序3:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux21 is
    port ( a, b : in std_logic; sel : in std_logic; c :
out std_logic;);
end sam2;
architecture one of mux21 is
begin
if sel = '0' then   c := a; else    c := b;  end if;
end two;
```

习 题

6-9. 根据例**4-23**设计**8**位左移移位寄存器，给出时序仿真波形。

6-10. 将例**6-12**中的**4**个**IF**语句分别用**4**个并列进程语句表达出来。

实 验 与 设 计

6-1. 七段数码显示译码器设计

(1) 实验目的：学习7段数码显示译码器设计；学习VHDL的CASE语句应用及多层次设计方法。

(2) 实验原理：7段数码是纯组合电路，通常的小规模专用IC，如74或4000系列的器件只能作十进制BCD码译码，然而数字系统中的数据处理和运算都是2进制的，所以输出表达都是16进制的，为了满足16进制数的译码显示，最方便的方法就是利用译码程序在FPGA/CPLD中来实现。例6-18作为7段译码器，输出信号LED7S的7位分别接如图6-18数码管的7个段，高位在左，低位在右。例如当LED7S输出为“1101101”时，数码管的7个段：g、f、e、d、c、b、a分别接1、1、0、1、1、0、1；接有高电平的段发亮，于是数码管显示“5”。注意，这里没有考虑表示小数点的发光管，如果要考虑，需要增加段h，例6-18中的`LED7S:OUT STD_LOGIC_VECTOR(6 DOWNTO 0)`应改为 `... (7 DOWNTO 0)` 。

实 验 与 设 计

(3) 实验内容1：说明例**6-18**中各语句的含义，以及该例的整体功能。在**QuartusII**上对该例进行编辑、编译、综合、适配、仿真，给出其所有信号的时序仿真波形。

提示：用输入总线的方式给出输入信号仿真数据，仿真波形示例图如图**6-17**所示。

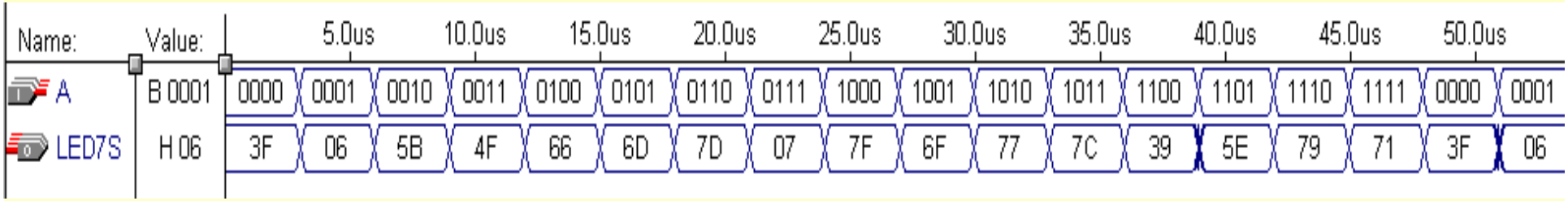


图6-17 7段译码器仿真波形

【例6-18】

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
ENTITY DECL7S IS
 PORT ( A  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
     LED7S : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)  ) ;
END ;
ARCHITECTURE one OF DECL7S IS
BEGIN
 PROCESS( A )
 BEGIN
 CASE  A  IS
  WHEN "0000" =>  LED7S <= "0111111" ;
  WHEN "0001" =>  LED7S <= "0000110" ;
  WHEN "0010" =>  LED7S <= "1011011" ;
  WHEN "0011" =>  LED7S <= "1001111" ;
  WHEN "0100" =>  LED7S <= "1100110" ;
  WHEN "0101" =>  LED7S <= "1101101" ;
  WHEN "0110" =>  LED7S <= "1111101" ;
  WHEN "0111" =>  LED7S <= "0000111" ;
  WHEN "1000" =>  LED7S <= "1111111" ;
  WHEN "1001" =>  LED7S <= "1101111" ;
WHEN "1010" =>  LED7S <= "1110111" ;
  WHEN "1011" =>  LED7S <= "1111100" ;
  WHEN "1100" =>  LED7S <= "0111001" ;
  WHEN "1101" =>  LED7S <= "1011110" ;
  WHEN "1110" =>  LED7S <= "1111001" ;
  WHEN "1111" =>  LED7S <= "1110001" ;
  WHEN OTHERS =>  NULL ;
  END CASE ;
 END PROCESS ;
END ;
```

实 验 与 设 计

(4) 实验内容2：引脚锁定及硬件测试。建议选**GW48**系统的实验电路模式**6**（参考附录图**6**），用数码**8**显示译码输出**(PIO46-PIO40)**，键**8**、键**7**、键**6**和键**5**四位控制输入，硬件验证译码器的工作性能。

(5) 实验内容3：用第**4**章介绍的例化语句，按图**6-19**的方式连接成顶层设计电路（用**VHDL**表述），图中的**CNT4B**是一个**4**位二进制加法计数器，可以由例**4-22**修改获得；模块**DECL7S**即为例**6-18**实体元件，重复以上实验过程。注意图**6-20**中的**tmp**是**4**位总线，**led**是**7**位总线。对于引脚锁定和实验，建议选电路模式**6**，用数码**8**显示译码输出，用键**3**作为时钟输入**(**每按**2**次键为**1**个时钟脉冲**)**，或直接接时钟信号**clock0**。

(8) 实验报告：根据以上的实验内容写出实验报告，包括程序设计、软件编译、仿真分析、硬件测试和实验过程；设计程序、程序分析报告、仿真波形图及其分析报告。

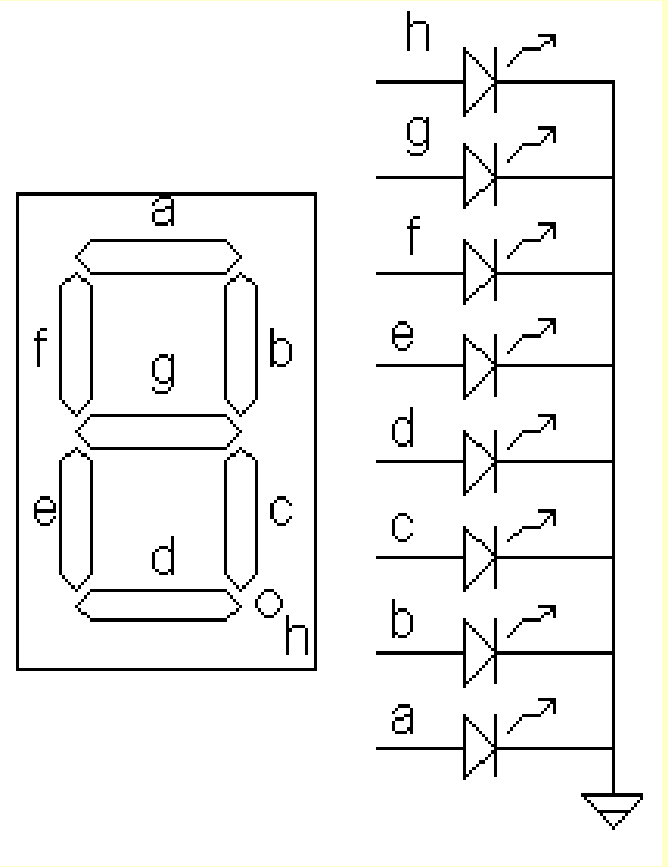


图6-18共阴数码管及其电路

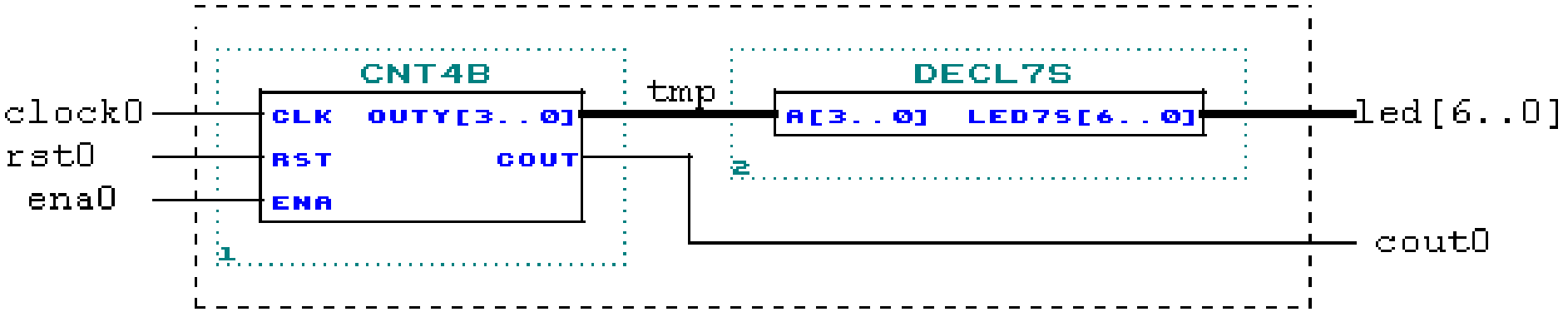


图6-19 计数器和译码器连接电路的顶层文件原理图

KX 康心科技

实 验 与 设 计

6-2. 八位数码扫描显示电路设计

(1) 实验目的：学习硬件扫描显示电路的设计。

(2) 实验原理：图6-20所示的是8位数码扫描显示电路，其中每个数码管的8个段：h、g、f、e、d、c、b、a（h是小数点）都分别连在一起，8个数码管分别由8个选通信号k1、k2、…k8来选择。被选通的数码管显示数据，其余关闭。如在某一时刻，k3为高电平，其余选通信号为低电平，这时仅k3对应的数码管显示来自段信号端的数据，而其它7个数码管呈现关闭状态。根据这种电路状况，如果希望在8个数码管显示希望的数据，就必须使得8个选通信号k1、k2、…k8分别被单独选通，并在此同时，在段信号输入口加上希望在该对应数码管上显示的数据，于是随着选通信号的扫变，就能实现扫描显示的目的。

实 验 与 设 计

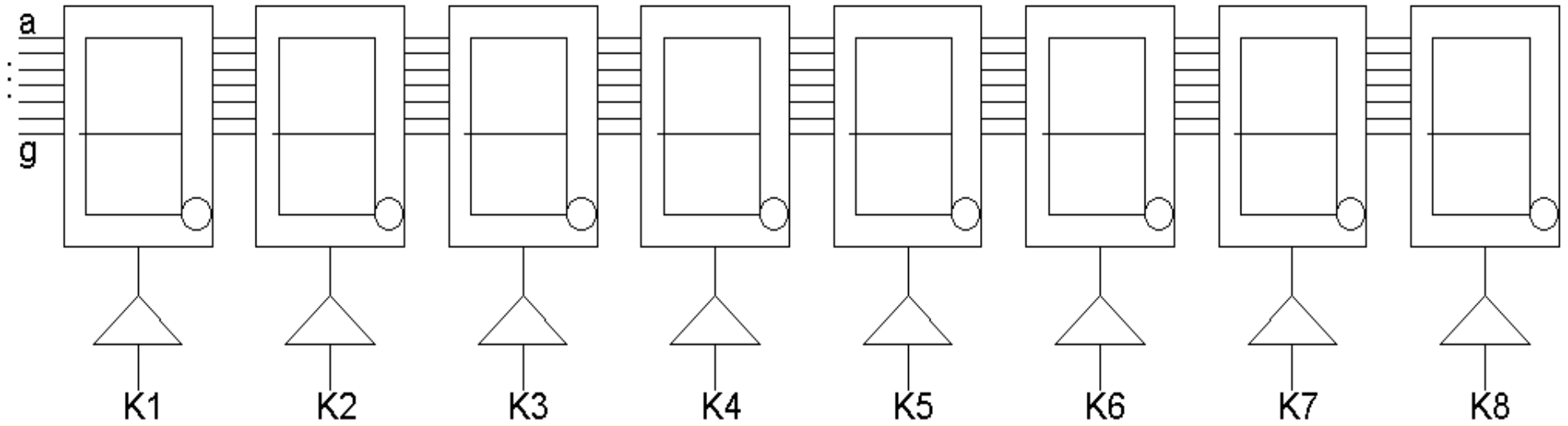


图6-20 8位数码扫描显示电路

【【例6-19】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SCAN_LED IS
  PORT ( CLK : IN STD_LOGIC;
         SG  : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);--段控制信号输出
         BT  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) );--位控制信号输出
 END;
ARCHITECTURE one OF SCAN_LED IS
    SIGNAL CNT8  : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL    A  : INTEGER RANGE 0 TO 15;
BEGIN
P1: PROCESS( CNT8 )
    BEGIN
        CASE  CNT8  IS
          WHEN "000" =>  BT <= "00000001" ; A <= 1 ;
          WHEN "001" =>  BT <= "00000010" ; A <= 3 ;
          WHEN "010" =>  BT <= "00000100" ; A <= 5 ;
          WHEN "011" =>  BT <= "00001000" ; A <= 7 ;
          WHEN "100" =>  BT <= "00010000" ; A <= 9 ;
          WHEN "101" =>  BT <= "00100000" ; A <= 11 ;
          WHEN "110" =>  BT <= "01000000" ; A <= 13 ;
          WHEN "111" =>  BT <= "10000000" ; A <= 15 ;
          WHEN OTHERS =>  NULL ;
        END CASE ;
     END PROCESS P1;
```

接下页

接上页

```
P2: PROCESS(CLK)
        BEGIN
         IF CLK'EVENT AND CLK = '1' THEN CNT8 <= CNT8 + 1;
         END IF;
        END PROCESS P2 ;
     P3: PROCESS( A ) --译码电路
      BEGIN
       CASE  A  IS
        WHEN 0 => SG <= "0111111";  WHEN 1  => SG <= "0000110";
        WHEN 2 => SG <= "1011011";  WHEN 3  => SG <= "1001111";
        WHEN 4 => SG <= "1100110";  WHEN 5  => SG <= "1101101";
        WHEN 6 => SG <= "1111101";  WHEN 7  => SG <= "0000111";
        WHEN 8 => SG <= "1111111";  WHEN 9  => SG <= "1101111";
        WHEN 10=> SG <= "1110111";  WHEN 11 => SG <= "1111100";
        WHEN 12=> SG <= "0111001";  WHEN 13 => SG <= "1011110";
        WHEN 14=> SG <= "1111001";  WHEN 15 => SG <= "1110001";
        WHEN OTHERS =>  NULL ;
       END CASE ;
      END PROCESS P3;
END;
```

实 验 与 设 计

(3) 实验内容1：说明例**6-19**中各语句的含义，以及该例的整体功能。对该例进行编辑、编译、综合、适配、仿真，给出仿真波形。实验方式：若考虑小数点，**SG**的**8**个段分别与**PIO49**、**PIO48**、…、**PIO42**（高位在左）、**BT**的**8**个位分别与**PIO34**、**PIO35**、…、**PIO41**（高位在左）；电路模式不限，引脚图参考附录图**10**。将**GW48EDA**系统左下方的拨码开关全部向上拨，这时实验系统的**8**个数码管构成图**6-20**的电路结构，时钟**CLK**可选择**clock0**，通过跳线选择**16384Hz**信号。引脚锁定后进行编译、下载和硬件测试实验。将实验过程和实验结果写进实验报告。

(4) 实验内容2：修改例**6-19**的进程**P1**中的显示数据直接给出的方式，增加**8**个**4**位锁存器，作为显示数据缓冲器，使得所有**8**个显示数据都必须来自缓冲器。缓冲器中的数据可以通过不同方式锁入，如来自**A/D**采样的数据、来自分时锁入的数据、来自串行方式输入的数据，或来自单片机等。

实 验 与 设 计

6-3. 数控分频器的设计

(1) 实验目的：学习数控分频器的设计、分析和测试方法。

(2) 实验原理：数控分频器的功能就是当在输入端给定不同输入数据时，将对输入的时钟信号有不同的分频比，数控分频器就是用计数值可并行预置的加法计数器设计完成的，方法是将计数溢出位与预置数加载输入信号相接即可，详细设计程序如例**6-20**所示。

(3) 分析：根据图**6-21**的波形提示，分析例**6-20**中的各语句功能、设计原理及逻辑功能，详述进程**P_REG**和**P_DIV**的作用，并画出该程序的**RTL**电路图。

实 验 与 设 计

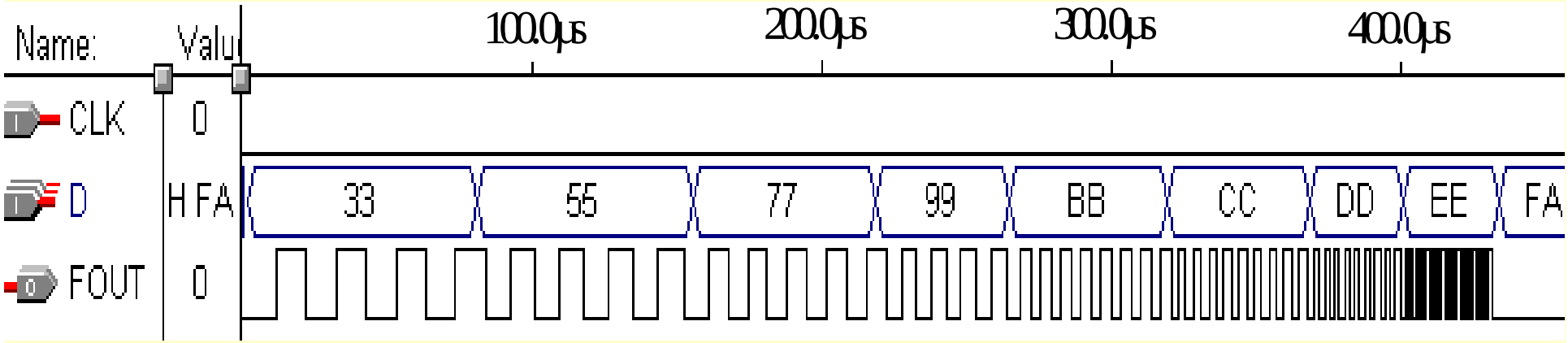


图6-21 当给出不同输入值D时，FOUT输出不同频率(CLK周期=50ns)

实 验 与 设 计

(4) 仿真：输入不同的CLK频率和预置值D，给出如图6-21的时序波形。

(5) 实验内容1：在实验系统上硬件验证例6-20的功能。可选实验电路模式1（参考附录图3）；键2/键1负责输入8位预置数D(PIO7-PIO0)；CLK由clock0输入，频率选65536Hz或更高(确保分频后落在音频范围)；输出FOUT接扬声器(SPKER)。编译下载后进行硬件测试：改变键2/键1的输入值，可听到不同音调的声音。

(6) 实验内容2：将例6-20扩展成16位分频器，并提出此项设计的实用示例，如PWM的设计等。

(7) 思考题：怎样利用2个例6-20给出的模块设计一个电路，使其输出方波的正负脉宽的宽度分别由可两个8位输入数据控制？

(8) 实验报告：根据以上的要求，将实验项目分析设计，仿真和测试写入实验报告。

【例6-20】

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY DVF IS
     PORT (    CLK  : IN STD_LOGIC;
                 D  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               FOUT : OUT STD_LOGIC  );
END;
ARCHITECTURE one OF DVF IS
    SIGNAL   FULL : STD_LOGIC;
BEGIN
  P_REG: PROCESS(CLK)
   VARIABLE CNT8 : STD_LOGIC_VECTOR(7 DOWNTO 0);
   BEGIN
      IF CLK'EVENT AND CLK = '1' THEN
          IF CNT8 = "11111111" THEN
          CNT8 := D;   --当CNT8计数计满时，输入数据D被同步预置给计数器CNT8
              FULL <= '1'; --同时使溢出标志信号FULL输出为高电平
                ELSE   CNT8 := CNT8 + 1;   --否则继续作加1计数
                     FULL <= '0'; --且输出溢出标志信号FULL为低电平
            END IF;
      END IF;
```

接下页

接上页

```
END PROCESS P_REG ;
   P_DIV: PROCESS(FULL)
     VARIABLE CNT2 : STD_LOGIC;
   BEGIN
   IF FULL'EVENT AND FULL = '1' THEN
     CNT2 := NOT CNT2; --如果溢出标志信号FULL为高电平，D触发器
输出取反
          IF CNT2 = '1' THEN  FOUT <= '1'; ELSE FOUT <= '0';
         END IF;
   END IF;
    END PROCESS P_DIV ;
END;
```

6-4. 32位并进/并出移位寄存器设计

仅用例6-8一个8位移位寄存器，再增加一些电路，如4个8位锁存器等，设计成为一个能为32位二进制数进行不同方式移位的移位寄存器。这个电路模型十分容易用到CPU的设计中。