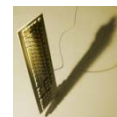
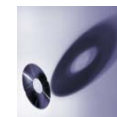


# 实验预备课

- 1、可编程逻辑器件设计
- 2、VHDL注意事项
- 2、软件使用
- 2、实验环境
- 3、实验介绍



# 可编程逻辑器件设计

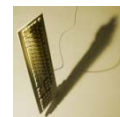
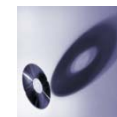
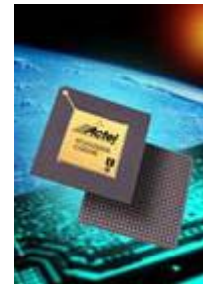
- 可编程器件简介
- 设计原则
- 设计流程



# 可编程器件简介

- 概述

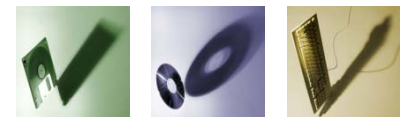
- PLD是电子设计领域中最具活力和发展前途的一项技术，它的影响丝毫不亚于70年代单片机的发明和使用。
- PLD能做什么呢？可以毫不夸张的讲，PLD能完成任何数字器件的功能，上至高性能CPU, 下至简单的位片电路，都可以用PLD来实现。
- 目前有多家公司生产CPLD/FPGA，主要有：ALTERA, XILINX, Lattice, Actel 。



# 可编程器件简介

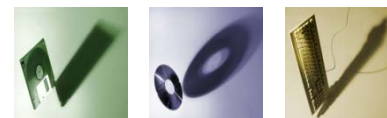
- FPGA

- Field Programmable Gate Array 现场可编程门阵列
- FPGA基于SRAM的架构，集成度高，以LE（包括查找表、触发器及其他）为基本单元，有内嵌Memory、DSP等，支持IO标准丰富。

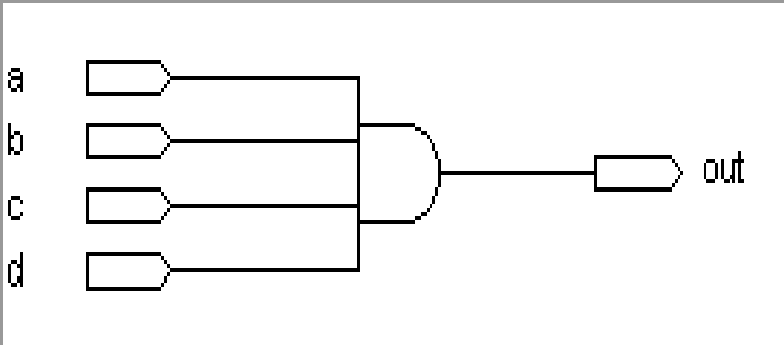
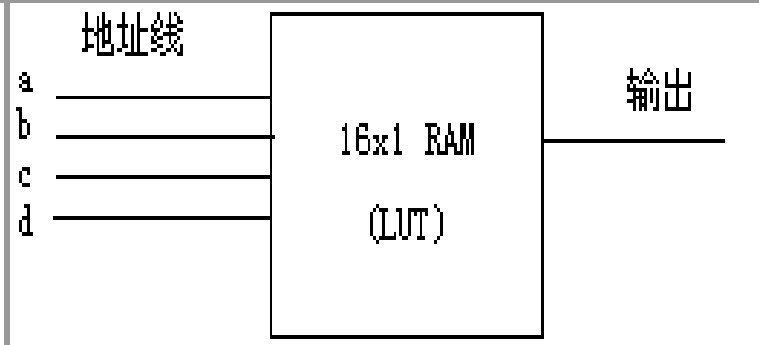


## 基于查找表（Look-Up-Table）的原理与结构：

- 采用这种结构的PLD芯片如altera的ACEX, APEX系列, xilinx的Spartan, Virtex系列等。
- 查找表（Look-Up-Table）简称为LUT，LUT本质上就是一个RAM
- 工作原理
  - 用户通过原理图或HDL语言描述逻辑电路
  - 软件自动计算逻辑电路的所有可能的结果，并把结果事先写入RAM
  - 每输入一个信号进行逻辑运算就等于输入一个地址进行查表，找出地址对应的内容，然后输出即可。

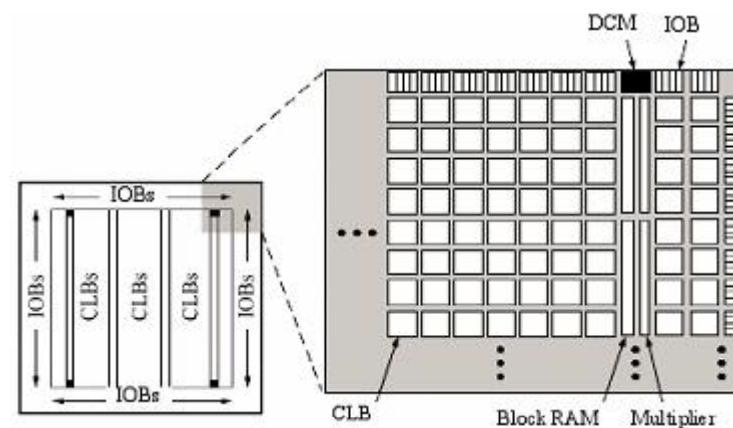


## 4输入 ‘与门’

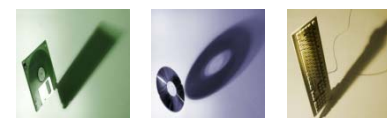
实际逻辑电路		LUT的实现方式	
			
a, b, c, d 输入	逻辑输出	地址	RAM中存储的内容
0000	0	0000	0
0001	0	0001	0
....	0	...	0
1111	1	1111	1



- Xilinx FPGA主要分为以下部分
  - 可编程输入输出单元(IOB)
  - 可编程逻辑块(CLB)
  - 时钟管理模块(DCM)
  - 片内RAM(BRAM)
  - 布线资源
  - 内嵌功能单元
  - 内嵌硬核



Spartan-III 系列结构



# Xilinx公司产品概述

- xilinx器件产品

- FPGA

- Virtex 系列
    - Spartan器件系列



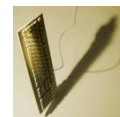
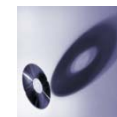
- CPLD

- XC9500系列
    - CoolRunner系列



- 其他

- 配置器件SPROM (S系列 P系列)
    - IP核





- 典型应用领域

- 数据采集

- 逻辑接口

- 电平接口

- 电平不同

- 单端到差分

- 数字信号处理

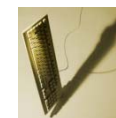
- IC设计验证

- 其他类，消费，医疗，工业控制

- 数字信号的基本上都可以用PLD实现



- 发展趋势
  - 高密度，大容量，高速度
  - 低成本，低电压，微功耗，微封装
  - 基于IP的设计方法
    - FPGA厂家
    - 开源硬件组织
  - 动态可重构
    - 通信系统
    - 重构计算机



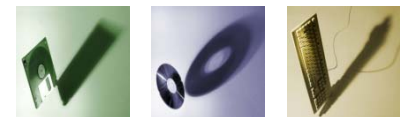
# 设计原则

- 面积和速度的平衡与互换
- 功耗考虑
- 硬件原则
- 系统原则
- 同步设计原则



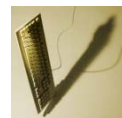
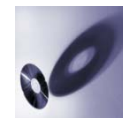
# 面积和速度的平衡与互换

- 面积和速度是数字系统设计考虑的两个重要指标，**FPGA**作为快速原型设计和系统验证的方法，首先就要考虑到这两个因素直接的平衡问题；
- 面积指某个**FPGA**设计综合之后占用的系统资源数，一般用占用的逻辑单元数量及**IO**接口数量来衡量，这一指标综合软件一般都能给出；
- 更小的面积通常代表更低的成本。



# 面积和速度的平衡与互换

- 简单说，速度通常指系统工作的频率，高频率常常代表高速度；
- 实际上，进行速度优化不仅仅是简单提高频率，而是要仔细考虑系统各个模块在各种工作状态下的时序要求；
- 另外可以通过并行操作提高速度；
- 在一定的工艺条件下，面积和速度常常是一对矛盾，因此需要考虑面积和速度的转换问题。

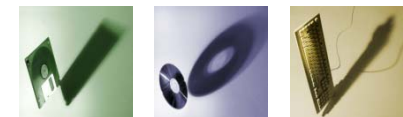
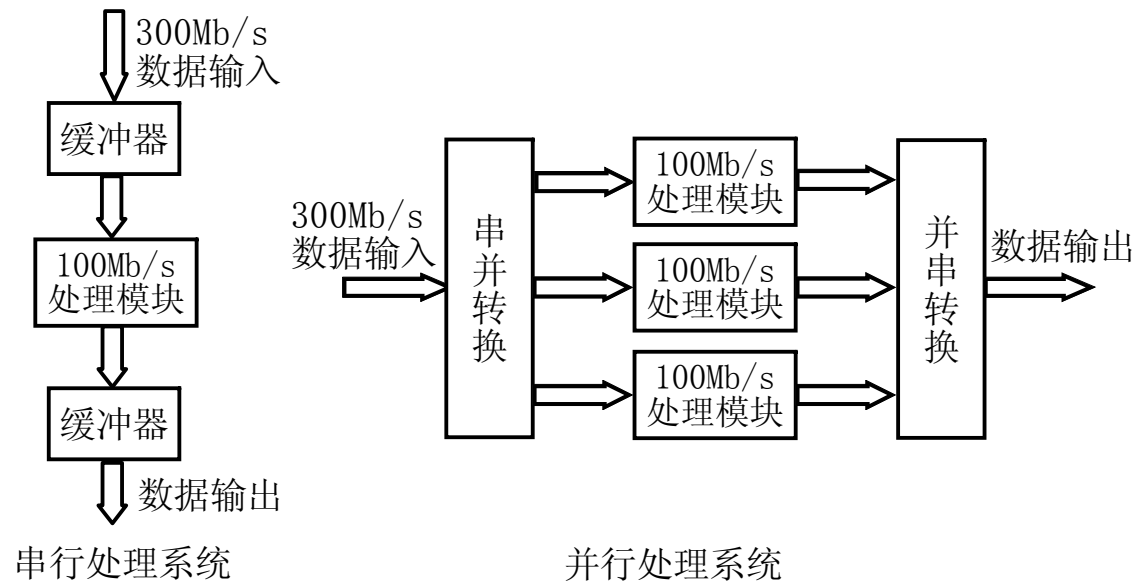


# 面积和速度的平衡与互换

- 面积换速度
  - 将原本复用的模块进行复制，变为并行操作的模块，以牺牲面积来换取速度；
  - 很多被复用的模块都是具有逻辑承接或时间先后关系的，无法直接并行化；
  - 需要修改硬件设计，重新对模块做规划。



# 面积和速度的平衡与互换



# 面积和速度的平衡与互换

- 速度换面积

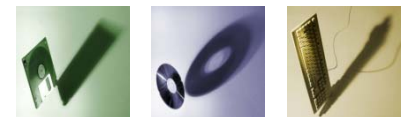
- 和利用面积换速度正好相反，把并行模块进行复用，以节省面积；
- 逻辑上更为简单，理论上，只要用足够的存储器，总能把并行的功能相同的模块进行复用；
- 但是要优化好存储器的管理，节省存储器，工作也并不简单。





# 功耗考虑

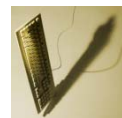
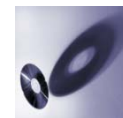
- 主要考虑动态功耗：
  - 动态功耗和逻辑翻转变化的频率成正比；
  - 设计时要考虑尽可能不要让所有的单元同时翻转，避免引起过大的功耗；
  - 例如，对于状态机的状态分配，之所以采用 **Gray** 码，另一个重要原因就是为了降低功耗；



# 硬件原则

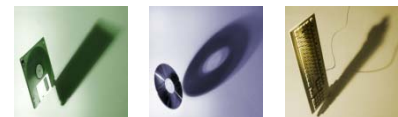
- 硬件原则

- 用**HDL**描述硬件进行数字系统，首先应该考虑的是硬件的实现；
- 程序的可读性，必须以硬件实现为前提；
- **HDL**描述的是硬件的结构和各个模块之间的连接关系，在设计前应对硬件本身有清晰的了解，然后用适当的**HDL**进行描述；
- 注意避免软件思维



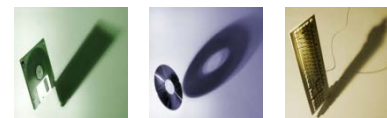
# 系统原则

- 数字系统设计应该从宏观和系统全局的角度进行考虑；
  - 例如对于模块等的复用和合理组织所得到的效果远比对于小部分代码的反复推敲大；



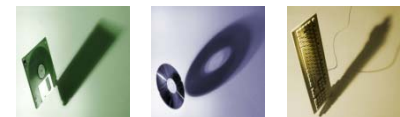
# 系统原则

- 设计前应对所用**FPGA**的底层硬件资源有所了解：
  - 底层可编程硬件单元
    - Xilinx的为Slice
    - Altera的为LE
  - Block RAM单元
  - 布线资源
  - 可配置IO单元
  - 时钟资源



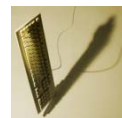
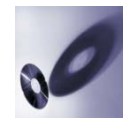
# 同步设计原则

- 在目前的条件下，采用异步电路设计并不理想，而现在的**FPGA**芯片都是为同步电路设计优化的；
- 单纯从**IC**设计角度看，同步电路比异步电路更加消耗资源；



# 同步设计原则

- 从资源考虑，关键要优化两种资源的比例；
- 另外，同步时序电路具有没有毛刺、信号稳定等优点；
- 同步时序电路中延时的产生；
- 同步时序电路中输入的同步；



# 设计流程

- 设计流程
  - 设计、输入和综合
    - 原理图
    - 硬件描述语言
  - 设计实现
    - 生成比特流文件
  - 设计验证
    - 门级仿真
    - 下载

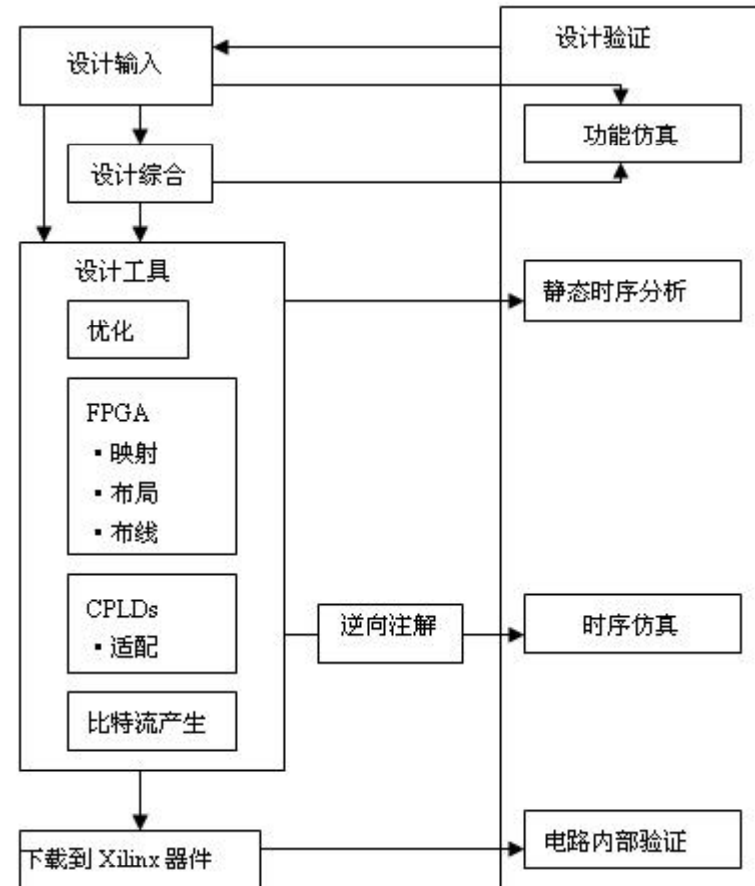
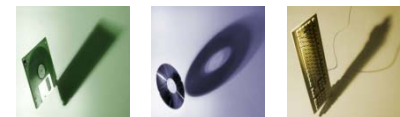
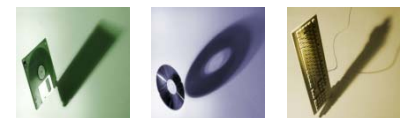


图 1.1 Xilinx 标准的设计流程

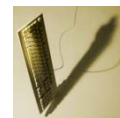
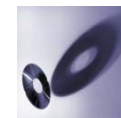


- 设计、输入和综合
  - 方案论证、系统设计和FPGA芯片选择等准备工作
  - 输入是将系统或电路以某种形式输入给EDA工具的过程
  - 创建设计后可以进行功能仿真，也称为前仿真，是在编译之前对用户所设计的电路进行逻辑功能验证
  - 综合就是将较高级抽象层次的描述转化成较低层次的描述。





- 层次化设计对于原理图和HDL输入都很重要
  - 可以将设计概念化
  - 将设计结构化
  - 使调试设计更容易
  - 使设计的不同部分的不同输入设计方法（原理图，HDL，本地编辑）能更容易结合
  - 使更容易的更新设计，其中包括设计，实现，以及在设计过程中验证个别元件
  - 减少优化时间
  - 便于并行设计



- 约束

- 如果想要对设计中的时间参数或者布局参数进行约束，设计者可以指定映射、块布局、以及时间规范

- 映射约束

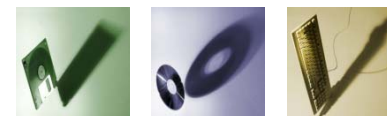
- 指定逻辑块如何映射到可配置的逻辑块

- 模块布局

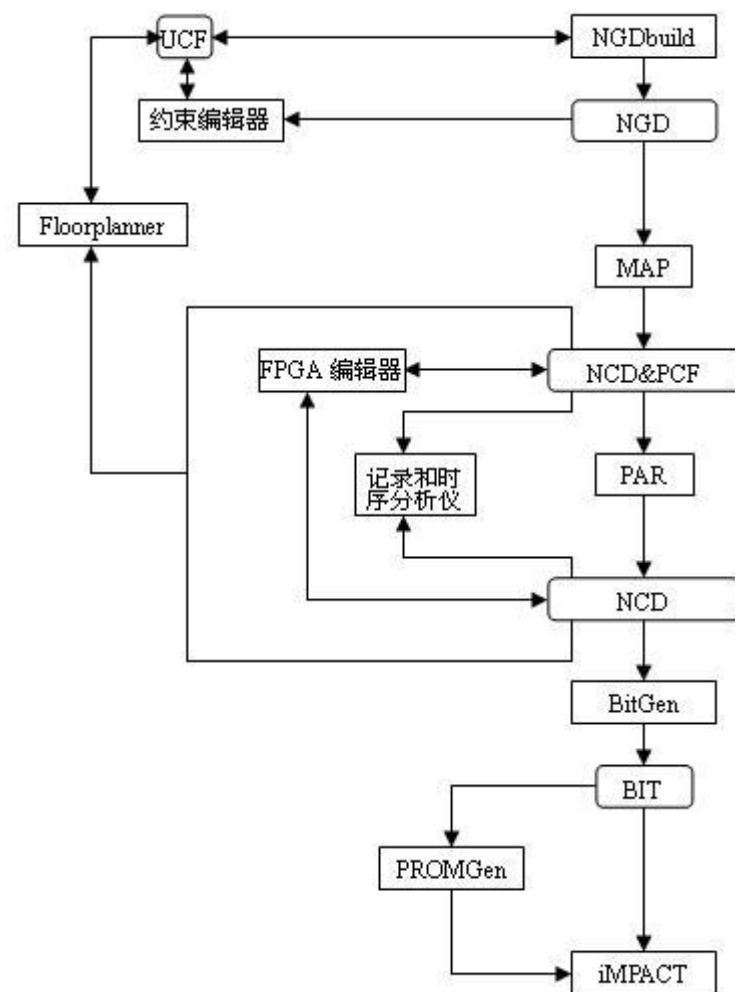
- 块布局可限制在指定位置，可以是多个位置中的其中一个，或者是一个位置区域。

- 时序规范

- 可以指定设计中路径的时间要求。



- 设计实现
  - 从逻辑设计文件映射或适配到指定的器件开始，到物理设计布线成功并生成比特流文件时结束。

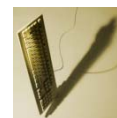
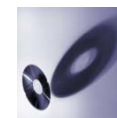


- 设计仿真
- 综合后仿真
  - 把综合生成的标准延时文件反标注到综合仿真模型中去，可估计门延时带来的影响。
- 时序仿真与验证
  - 也称后仿真，是指将布局布线的延时信息反标注到设计网表中来检测有无时序违规板级仿真与验证
- 板级仿真
  - 主要应用于高速电路设计中，对高速系统的信号完整性、电磁干扰等特征进行分析，一般都以第三方工具进行仿真和验证。
- 芯片编程与调试
  - 将编程数据下载到FPGA芯片中



# VHDL注意事项

- 描述方式
- 进程
- 信号与变量
- 类型转换
- 描述限制



# VHDL程序结构

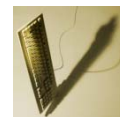
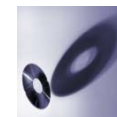
## VHDL程序基本结构

例 一个2输入的与门的逻辑描述

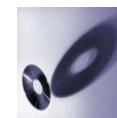
```
LIBRARY ieee;                                --库说明语句
USE ieee.std_logic_1164.ALL;                 --程序包说明语句
ENTITY and2 IS
    PORT(a,b      : IN  STD_LOGIC;
          y        : OUT STD_LOGIC);
END and2;
ARCHITECTURE and2x OF and2 IS
BEGIN
    y<=a AND b;
END and2x;
```

} 实体部分

} 结构体部分



- 描述方式：
  - 行为描述：
    - 描述该设计单元的功能，即该硬件能做什么。
  - 数据流（寄存器传输RTL）描述：
    - 数据流描述又称为寄存器传输描述，以此描述数据的传输和变换。
  - 结构化描述：
    - 描述该设计单元的硬件结构，即该硬件是如何构成的。



- 行为描述:

- 主要使用函数、过程、进程语句，以算法形式描述数据的变换和传送。

ARCHTECTURE behavioral OF compare IS

BEGIN

PROCESS(A,B)

BEGIN

IF(A=B) THEN

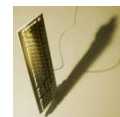
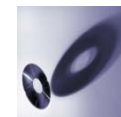
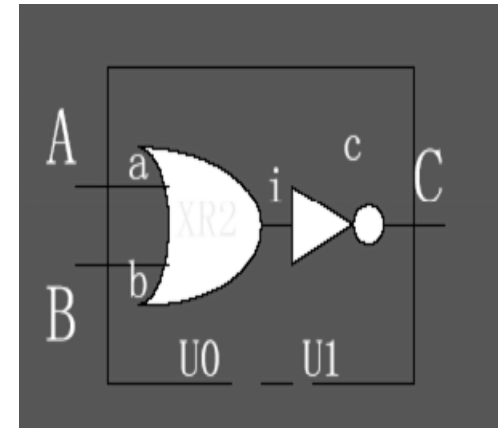
C<='1';

ELSE

C<='0';

END IF;

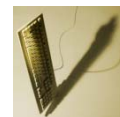
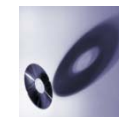
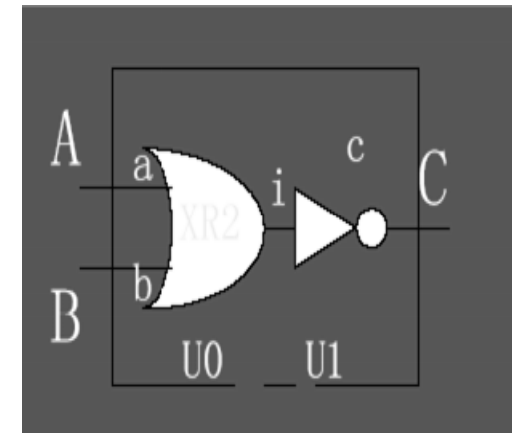
END behavioral;





- 数据流（寄存器传输RTL）描述：
  - 主要使用并行的信号赋值语句，即显示了该设计单元的行为，也隐含表示了该设计单元的结构

```
ARCHTECTURE data_flow OF compare IS  
BEGIN  
  C<=NOT(A XOR B);  
END data_flow;
```



- 结构化描述:

- 主要使用配置指定语句及元件例化语句来描述元件的类型及元件的互联关系。

ARCHTECTURE structral OF compare IS

SIGNAL I : BIT;

COMPONENT xor

PORT(A,B: IN BIT;

I : OUT BIT);

END COMPONENT;

COMPONENT inv

PORT(I: IN BIT;

C : OUT BIT);

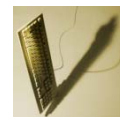
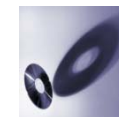
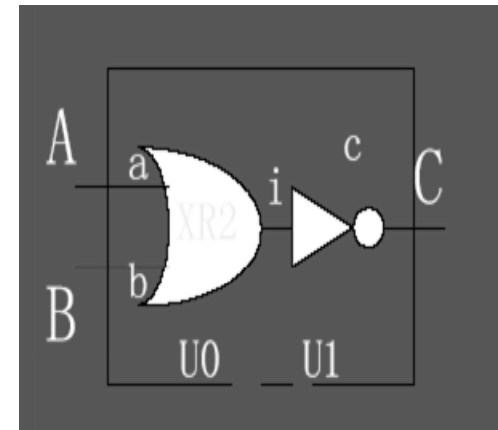
END COMPONENT;

BEGIN

U0: xor PORT MAP(A,B,I);

U1: inv PORT MAP(I,C);

END structral;



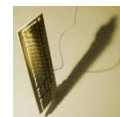
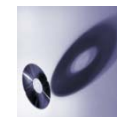
## 进程（PROCESS）语句结构

- 定义：

```
[进程名:] PROCESS [(敏感信号1, 敏感信号2, ...)]  
    [变量说明语句;]  
    BEGIN  
        ⋮  
    END PROCESS;
```

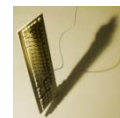
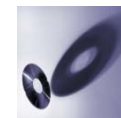
- 进程（PROCESS）中语句的顺序性

- 语句是顺序执行的
- 进程之间是并行执行的
- 顺序执行的语句只在PROCESS和SUBPROGRAM的结构中使用。



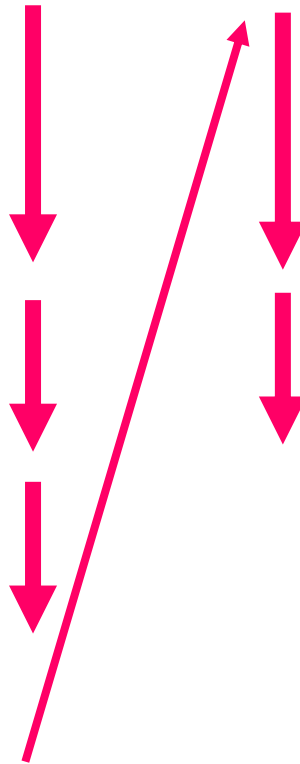
## ■ PROCESS的激活与挂起

- 进程有两种状态：激活与挂起
- 一般带有0个或多个敏感信号，是PROCESS的输入信号
- 运行开始时（初始化），所有进程均被激活，并执行
- 遇到wait语句进程挂起。
- 当等待语句条件满足时，进程再次被激活，并从等待语句处接着运行。
- 执行完最后一条语句后，返回第一条语句等待敏感信号发生变化，若无则接着执行。



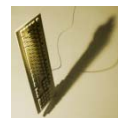
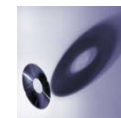
- 进程的执行过程

```
p1: process()  
begin  
  ...  
  wait on s1;  
  ...  
  wait on s2;  
  ...  
  wait on s1;  
  ...  
end process p1;
```



- 各进程之间并行执行，与顺序无关

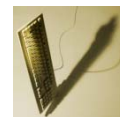
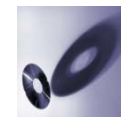
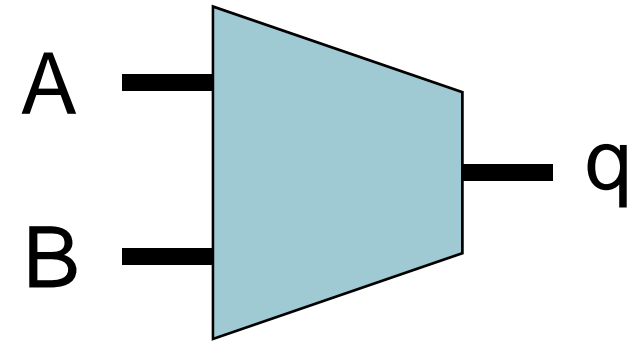
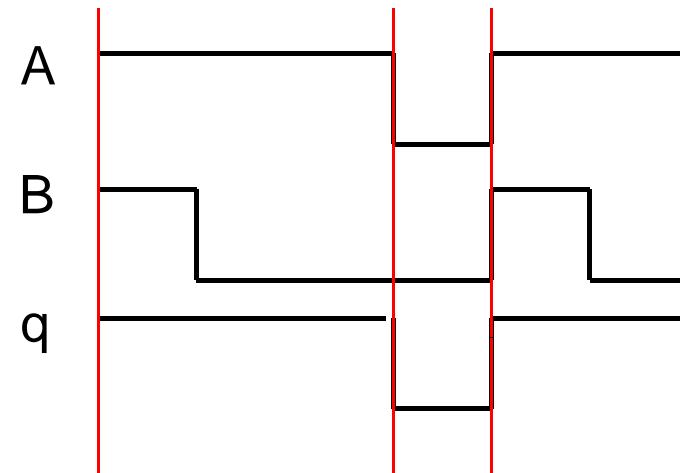
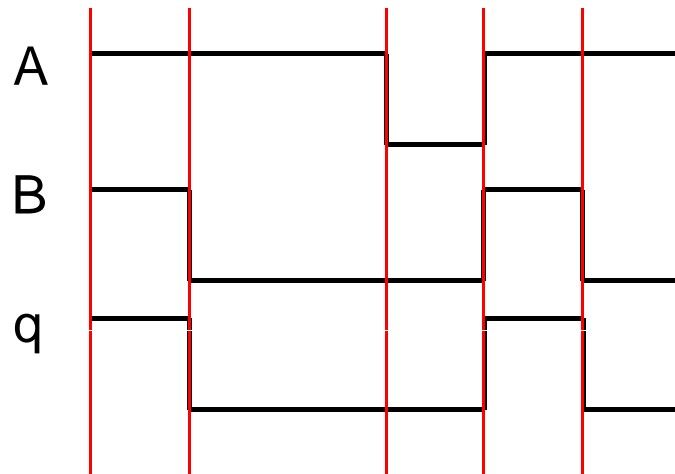
- 各进程内部顺序执行。



```

ARCHITECTURE arch OF and2 IS
BEGIN
  PROCESS (A,B)
    VARIABLE tmp1: STD_LOGIC;
  BEGIN
    tmp1:= A AND B;
    q <= tmp1;
  END PROCESS;
END arch;

```



# 对象

名称	意义	值
信号 Signal	<ul style="list-style-type: none"><li>● 控制模块或进程间通信的机制</li><li>● 定义两个模块或进程间的数据通路</li></ul>	<ul style="list-style-type: none"><li>● 时间序列（波形）</li><li>● 延迟赋值（延时）</li></ul>
变量 Variable	<ul style="list-style-type: none"><li>● 程序中临时使用的对象。</li></ul>	<ul style="list-style-type: none"><li>● 可变的单值</li><li>● 即时赋值（无延时）</li></ul>
常量 Constant	<ul style="list-style-type: none"><li>● 程序中不变的量</li></ul>	<ul style="list-style-type: none"><li>● 初始化时确定，运行过程中不改变</li></ul>

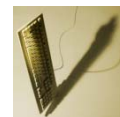
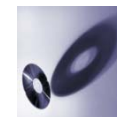
## 共同点:

- 具有相同的类型特性;
- 在初始化时均可赋初值;
- 可相互赋值。



- 信号与变量的区别

- 信号是硬件中连线的抽象描述，信号在元件的端口连接元件；变量在硬件中没有类似的对应关系，它们用于硬件特性的高层次建模所需的计算中。
- 信号赋值有延迟（如果不指定延迟，则延迟为0），而变量因为不是实际物理量，因此变量的赋值就不允许有延迟。
- 进程（PROCESS）语句只对信号敏感而不对变量敏感。
- 变量的赋值符 “:=”，其值立即被赋予变量，下一条语句该变量的值就为新赋的值。信号赋值符 “<=”，不会立即发生赋值，下一条语句仍使用原来的信号值。
- 信号是全局性的，变量是局部性的





# 信号赋值语句

## ◆格式

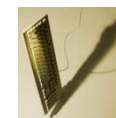
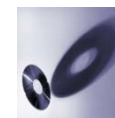
信号名 <= 波形{, 波形};

波形::= 值表达式 [after 时间表达式];

例: B <= A+C;

B <= A after 5 ns;

Clock <= '0' ,  
          '1' after 5 ns,  
          '0' after 10 ns,  
          '1' after 50 ns;



### ◆信号的延时赋值

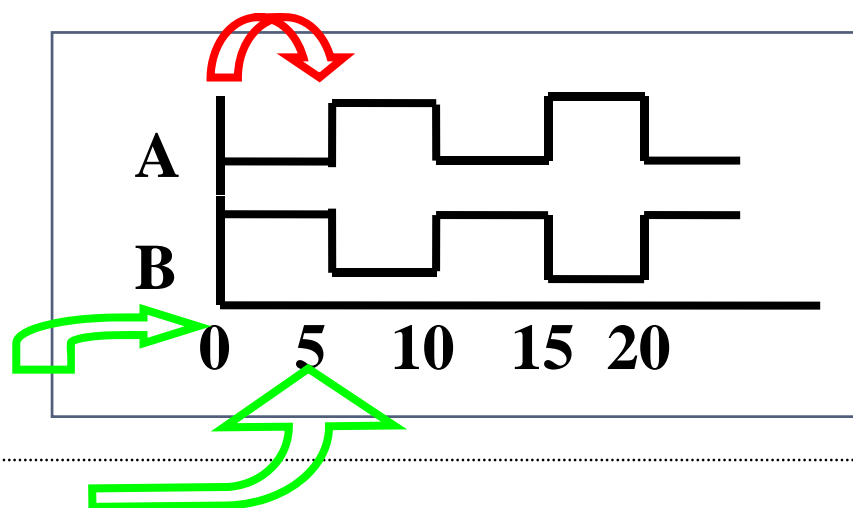
`A <= B after 5 ns;`

`B <= A after 5 ns;`

- 当前:  $A=0$ ,  $B=1$ ;

- 5 ns 后,  $A=1$ ,  $B=0$ ;

- 被赋值: 将来值。右边信号值: 当前值;



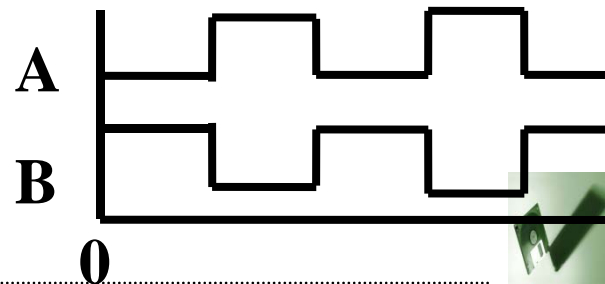
## 无延迟指定—— $\delta$ 延迟

```
B <= A;  
A <= B;
```

$\Rightarrow$

```
B <= A after  $\delta$  fs;  
A <= B after  $\delta$  fs;
```

- ◆  $\delta$  是一个无穷小量;
- ◆ 信号赋值须经过  $\delta$  时间延迟。
- ◆ 当前值:  $A=0, B=1$ ;
- ◆  $\delta$  时间后:  $A=1, B=0$ ;



# 变量赋值语句

## ◆格式

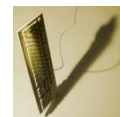
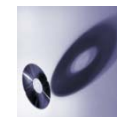
变量名 := 值;

例: B := A;

A := B;

结果就是A=B, 即时赋值。

- 无延迟特性, 直接赋值
- 一般情况下不允许有全程变量
  - 不能在实体和结构体声明中定义
  - 只能在进程内部或子程序中定义。
- 进程挂起时, 变量值保持不变。  
(仅在第一次执行进程时初始化)
- 子程序变量在每次调用时赋初值。



## 例 信号和变量值代入的区别举例

```
PROCESS (a, b, c, d)
BEGIN
```

```
    d<=a;
    x<=b+d;
    d<=c;
    y<=b+d;
END PROCESS;
    x<=b+c;
    y<=b+c;
```

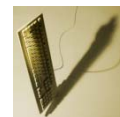
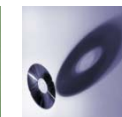
结果:

当进程运行时，信号赋值将自上而下顺序执行，但第一项赋值操作并不会发生，这是因为信号赋值是在进程结束时才起作用。因为在进程结束更新时，d的最后一个赋值为c，因此执行结果为该式。

```
PROCESS (a,b,c)
VARIABLE d: std_logic_vector(3 downto 0);
BEGIN
```

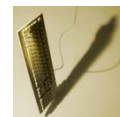
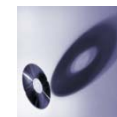
```
    d: =a;
    x<=b+d;
    d: =c;
    y<=b+d;
END PROCESS;
    x<=b+a;
    y<=b+c;
```

结果:



- 一般常用的端口数据类型STD\_LOGIC  
STD\_LOGIC\_VECTOR
  - 在STD\_LOGIC1164包中定义
  - 九值逻辑系统
    - ‘U’: 未初始化的
    - ‘X’: 强未知的
    - ‘0’: 强0
    - ‘1’: 强1
    - ‘Z’: 高阻态
    - ‘W’: 弱未知的
    - ‘L’: 弱0
    - ‘H’: 弱1
    - ‘-’: 忽略

CASE语  
句?



# 数据类型的转换

在VHDL程序中，不同类型的对象不能代入，因此要进行类型转换。类型转换的方法有：

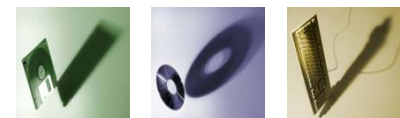
(1) 类型标记法。用类型名称来实现关系密切的标量类型之间的转换。

例如：        **VARIABLE x: INTEGER;**

**VARIABLE y: REAL;**

使用类型标记（即类型名）实现类型转换时，可采用赋值语句：

**x :=INTEGER(y);        y :=REAL(x)。**



## (2) 类型函数法。

VHDL程序包中提供了多种转换函数，换，以实现正确的赋值操作。常用的类型转换

该函数由  
*STD\_LOGIC\_UNSIGNED*

该函数由  
*STD\_LOGIC\_ARITH*

以下函数由  
*STD\_LOGIC\_1164*  
程序包定义

★*CONV\_INTEGER()*: 将*STD\_LOGIC*  
*INTEGER*

★*CONV\_STD\_LOGIC\_VECTOR()*

类型或 *SIGNED* 类型转换成*STD\_LOGIC\_VECTOR*类型。

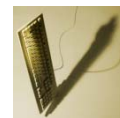
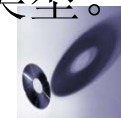
★*TO\_BIT()*: 将*STD\_LOGIC*类型转换成*BIT*类型。

★*TO\_BIT\_VECTOR()*: 将*STD\_LOGIC\_VECTOR*类型转换  
*BIT\_VECTOR* 类型。

★*TO\_STD\_LOGIC()*: 将*BIT*类型转换成*STD\_LOGIC*类型。

★*TO\_STD\_LOGIC\_VECTOR()*: 将*BIT\_VECTOR*类型转换成  
*STD\_LOGIC\_VECTOR*类型。

注意: 引用时必须首先 打开库和相应的程序包。





## 描述限制:

- a) 禁止在一个进程中存在*两个边沿检测*的寄存器描述
- b) 禁止使用IF语句中的 *ELSE* 项
- c) 寄存器描述中必须代入*信号值*

例7: PROCESS(clk1,clk2)

BEGIN

IF (clk1 'EVENT AND

y<=a;

END IF;

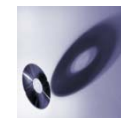
IF (clk2 'EVENT AND clk2='1') THEN

z<=b;

END IF;

END PROCESS;

在一个进程中不允许引入两个边沿检测的寄存器进行描述!



例: PROCESS(clk)  
BEGIN

IF (clk'EVENT AND clk = '1')

y<=a;

ELSE

y<=b;

END IF;

END PROCESS;

禁止使用  
ELSE!

不可能有这样的硬  
件电路与之对应!

-- 禁止使用

例9: PROCESS(clk)  
VARIABLE tmp: STD\_LOGIC;  
BEGIN

IF (clk'EVENT AND clk = '1') THEN

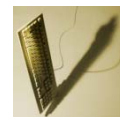
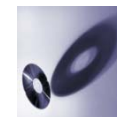
tmp:=a;

END IF;

y<=tmp;

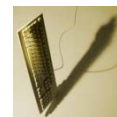
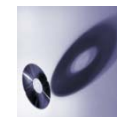
END PROCESS;

必须代入  
信号值!

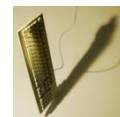
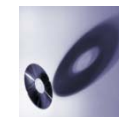
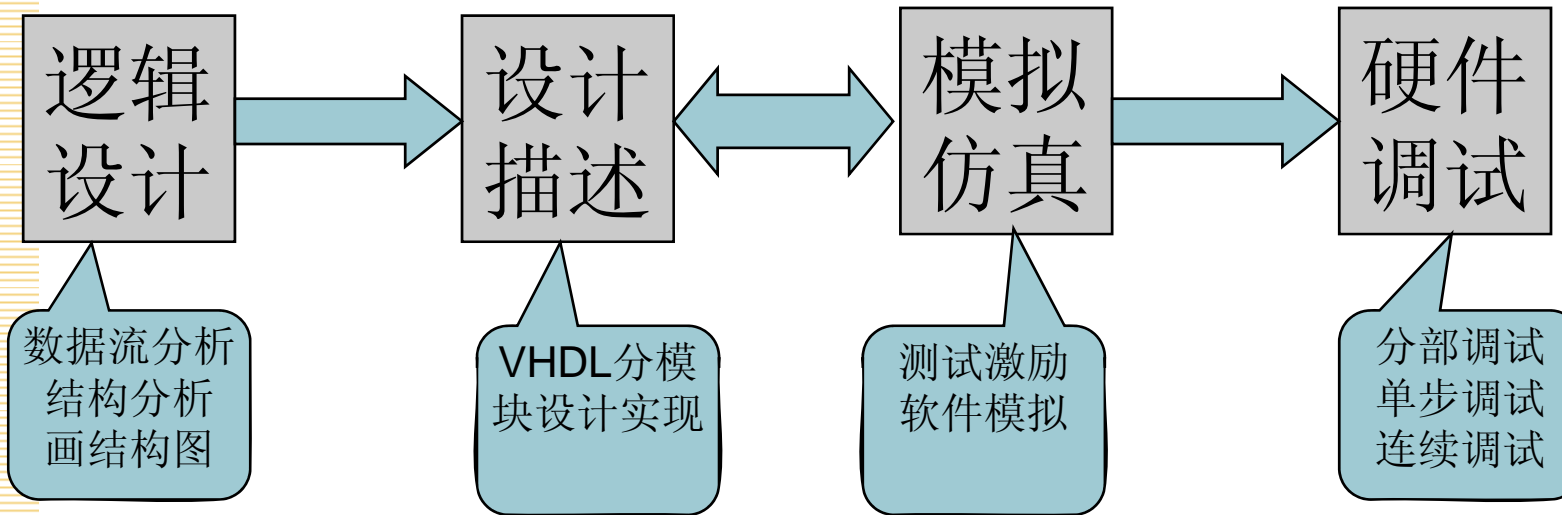


# 硬件描述语言与实验介绍

- 1、VHDL语言
- 2、实验环境
- 3、实验介绍

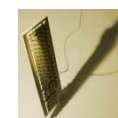


# 设计流程



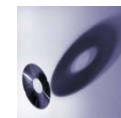
## 逻辑设计

- 总体规划，明确设计要求，想到达的目标
- 分析数据流程
- 设计结构，功能划分
- 设计文档



## 设计描述

- 这部分包括设计规划和程序的编写。
  - 设计规划主要包括设计方式的选择及是否进行模块划分。设计方式一般包括直接设计，自顶向下和自底向下设计，这个和其他软件语言差不多。
  - 最重要还是模块划分，这个和设计者的设计水平有很大关系。
- 编写各个模块的VHDL程序
- 将各模块的VHDL程序综合起来完成整个设计的VHDL描述



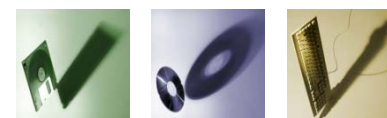
# 模拟仿真

- 建立模拟仿真环境
- 设计测试激励
- 软件模拟



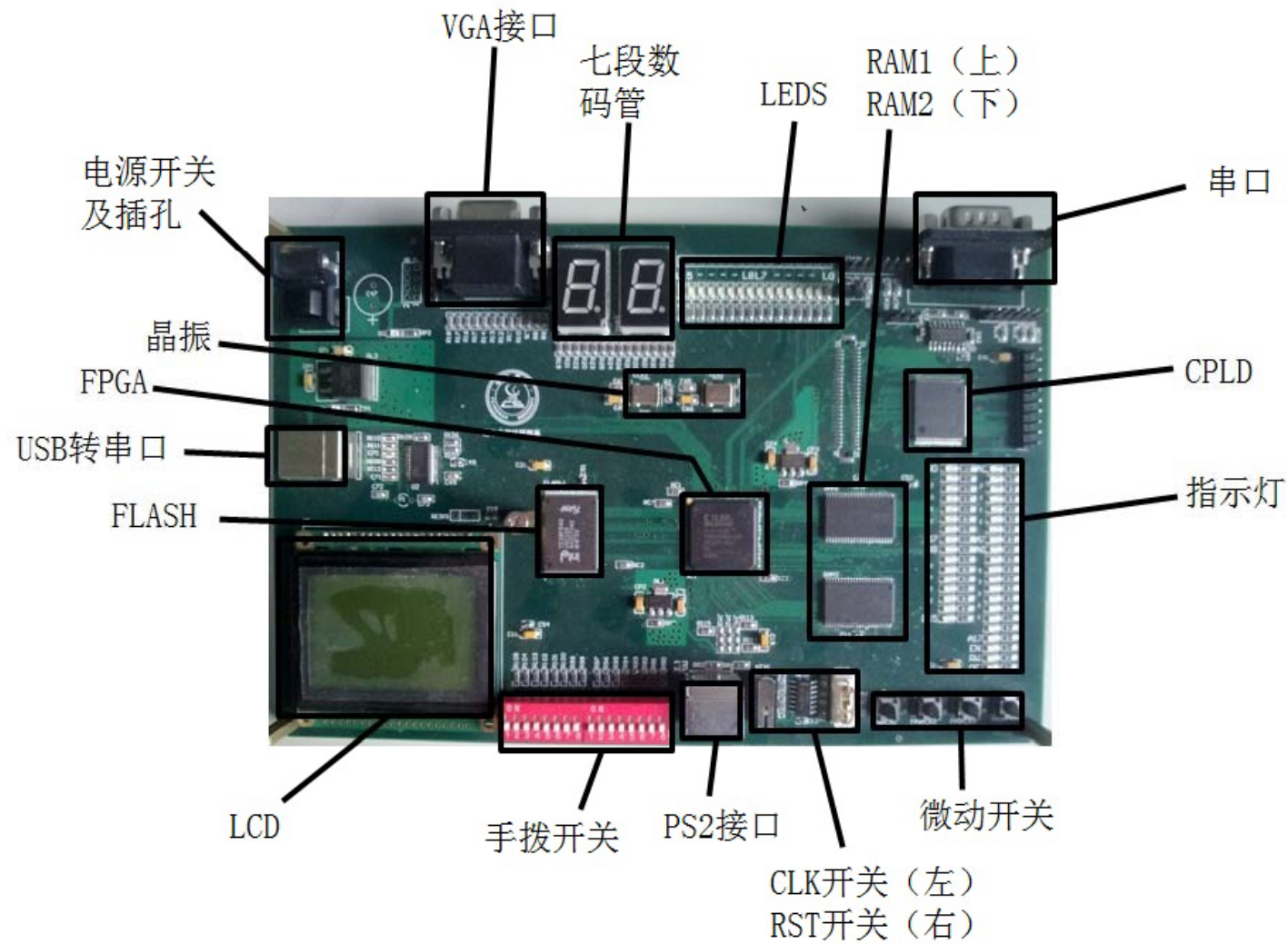
## 硬件调试

- 通过软件综合后，下载到FPGA中
- 实际硬件测试
  - 单步
  - 连续





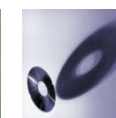
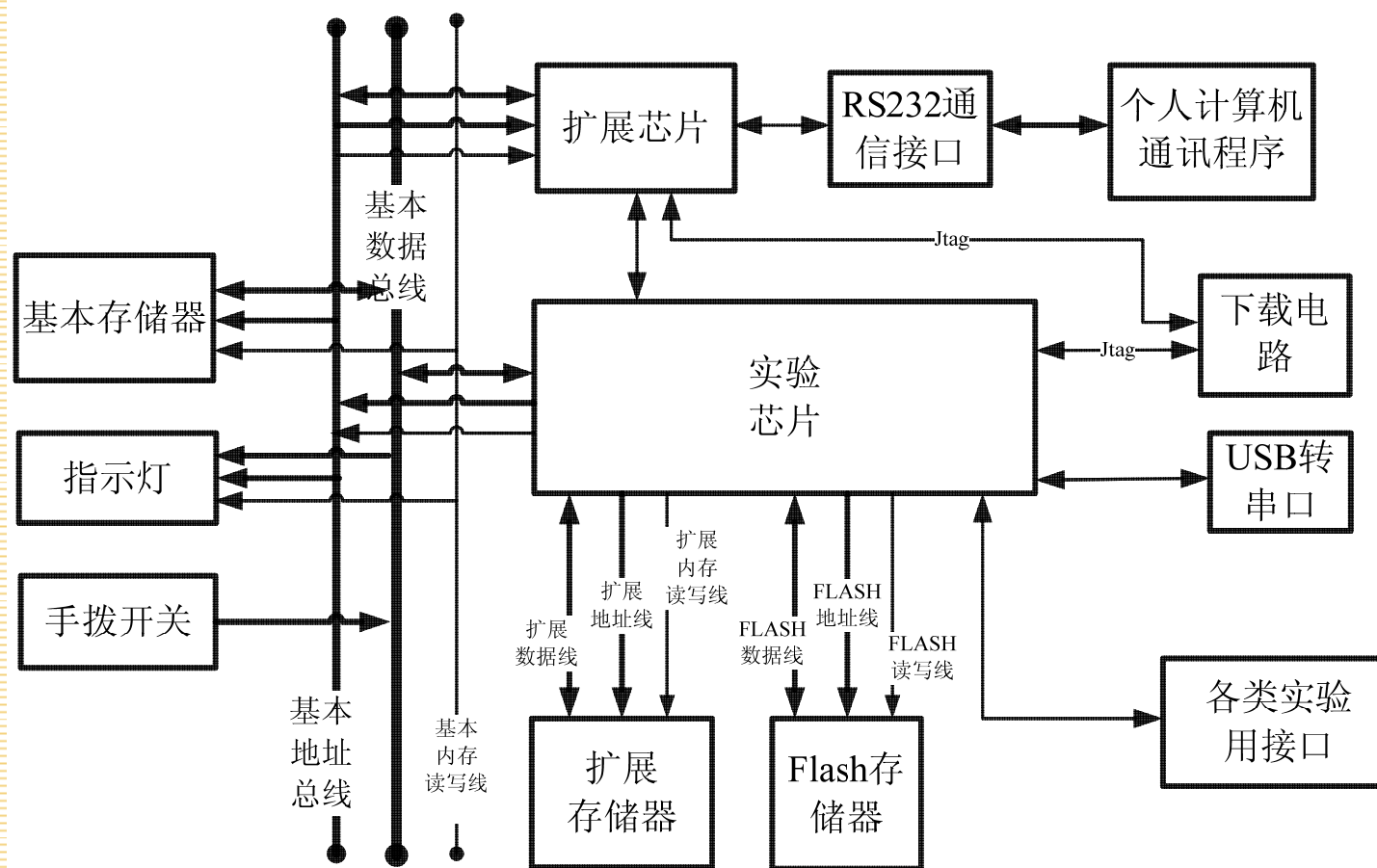
# 实验平台



## — 实验配件

- **USB转串口** 1
- 串口线 1
- 下载线 1
- 电源线 1





# 硬件资源

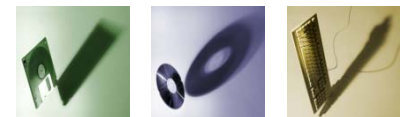
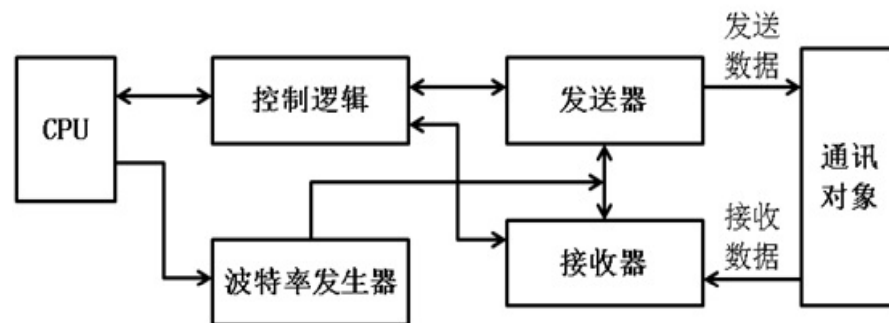
- FPGA

器件名称	逻辑单元	系统门数	CLB阵列	CLB总数	最大用户I/O	BlockRAM容量 bit
XC3S100E	2160	100k	22×16	240	108	72k
XC3S250E	5508	250k	34×26	612	172	216k
XC3S500E	10476	500k	46×34	1164	232	360k
XC3S1200E	19512	1200k	60×46	2168	304	504k
XC3S1600E	33192	1600k	76×58	3688	376	648k



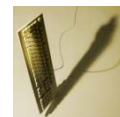
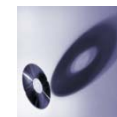
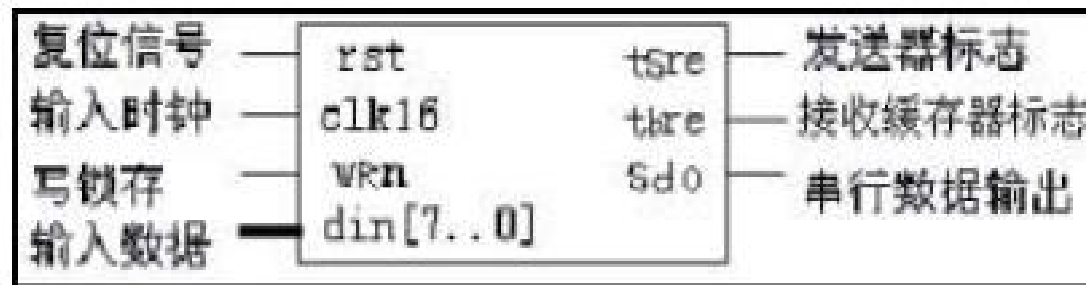
# 硬件资源

- CPLD
  - XC95144XL
  - 串口、PS2电平转换、ROM
  - 串口数据格式为8位数据位、1位校验位和1位停止位，波特率为9600bit/s。



# 硬件资源

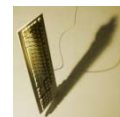
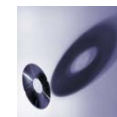
- 数据的发送
  - 给出wrn信号
  - 在数据发送过程中用输出信号tbre、tsre作为标志信号，当一帧数据由发送缓冲器tbr[7..0]送到发送移位寄存器tsr[7..0]时，tbre信号为1
  - 数据由发送移位寄存器tsr[7..0]串行发送完毕时，tsre信号为1，通知CPU在下一个时钟装入新数据。



# 硬件资源

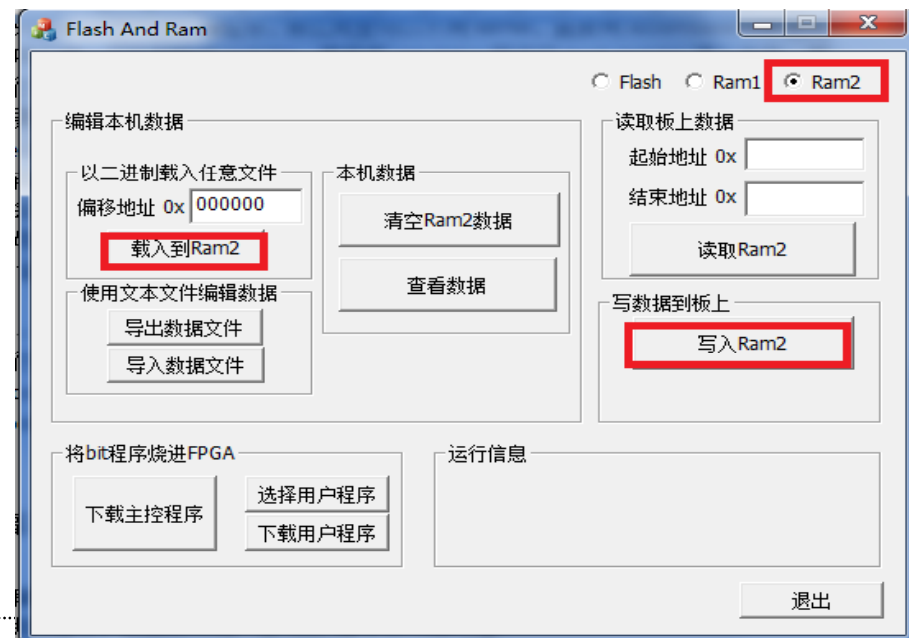
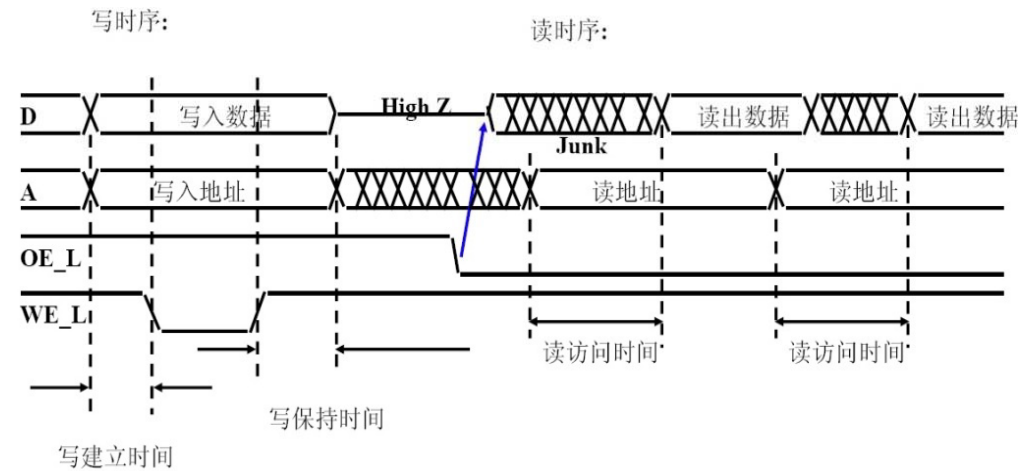
- 数据的接收
  - 发送来的数据由逻辑1变为逻辑0可以视为一个数据帧的开始。
  - 接收器先要捕捉起始位，确定rxn输入由1到0，逻辑0要8个CLK16时钟周期，才是正常的起始位，然后在每隔16个CLK16时钟周期采样接收数据，
  - 数据接收标志信号标志数据接收完

复位信号	—	rst	dout[7..0]	—	输出数据总线
时钟信号	—	clk16	data_ready	—	数据接收完毕
接收串行输入数据	—	rxn	framing_error	—	帧错误信号
读锁存信号	—	rdn	parity_error	—	校验错误信号



# 硬件资源

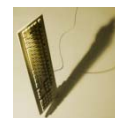
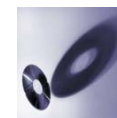
- 存储器
  - SRAM
    - 256K\*16
  - FLASH
    - 8MB





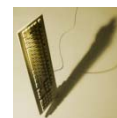
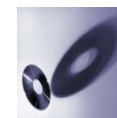
# 硬件资源

- 总线及接口
  - 注意三态控制
  - PS2、VGA、开关、数码管



# 软件使用

- 安装软件ISE
  - 版本不要太低
  - 选择器件库时一定要选spartan2
  - 充分使用各种模拟仿真软件
    - Modelsim
    - ActiveHDL



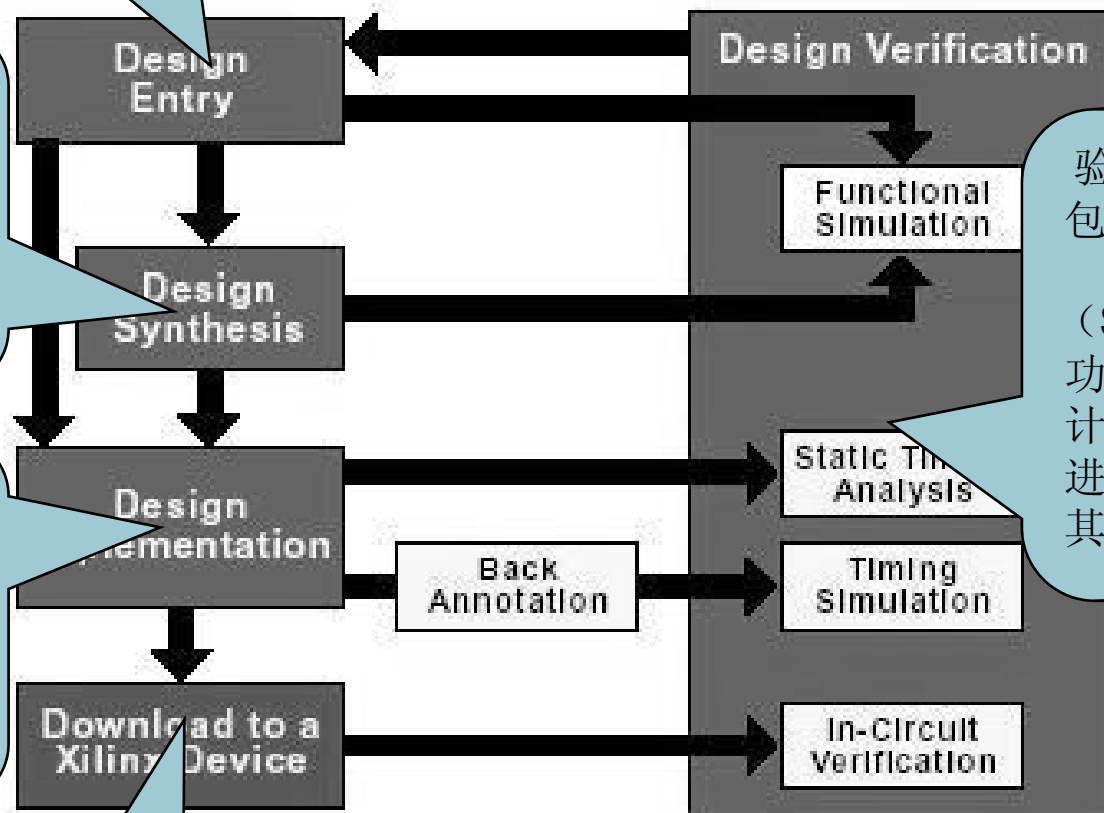
## • ISE工程设

原理图、状态机、  
波形图、硬件描  
述语言

将模型、算法、行为  
和功能描述转换为  
FPGA/CPLD基本结  
构相对应的网表文件，  
即构成对应的映射关  
系

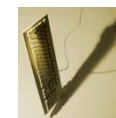
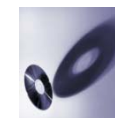
根据芯片将逻辑网表  
适配到器件上。ISE  
分为：翻译  
(Translate)、映射  
(Map)、布局布线  
(Place & Route) 3  
个步骤。

将生成的配置文件  
写入芯片中进行测  
试

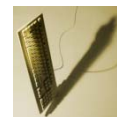
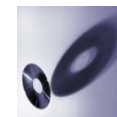
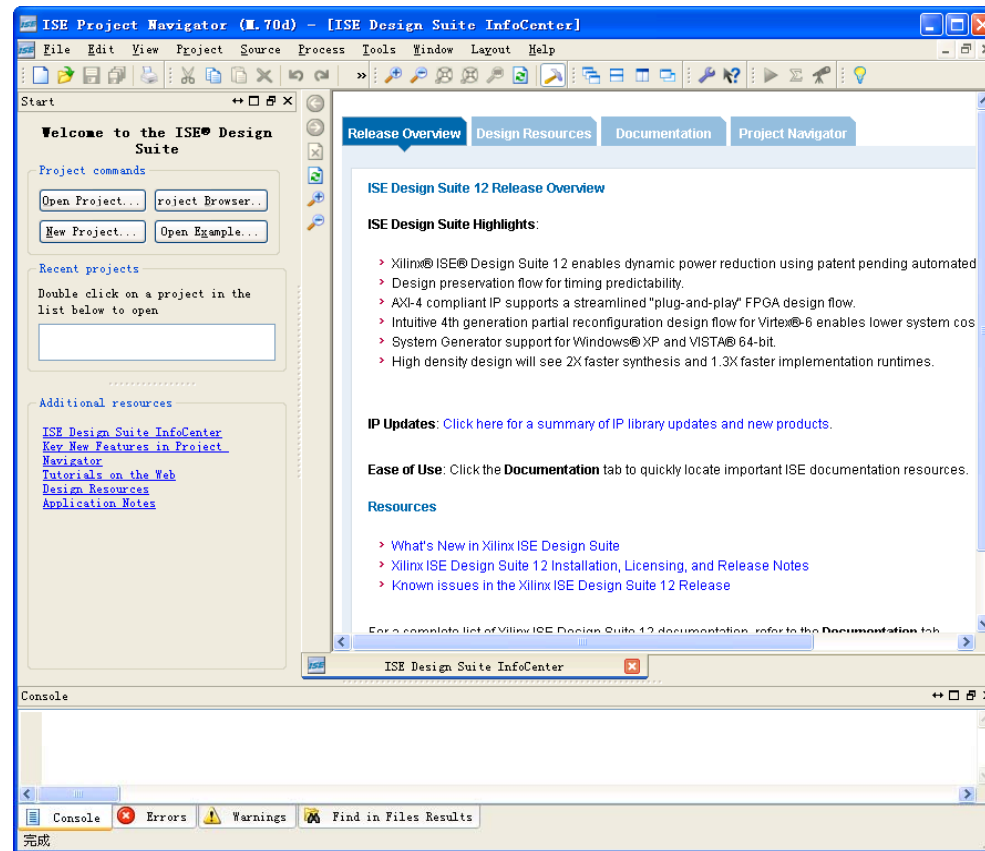


验证

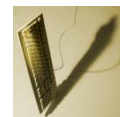
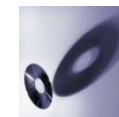
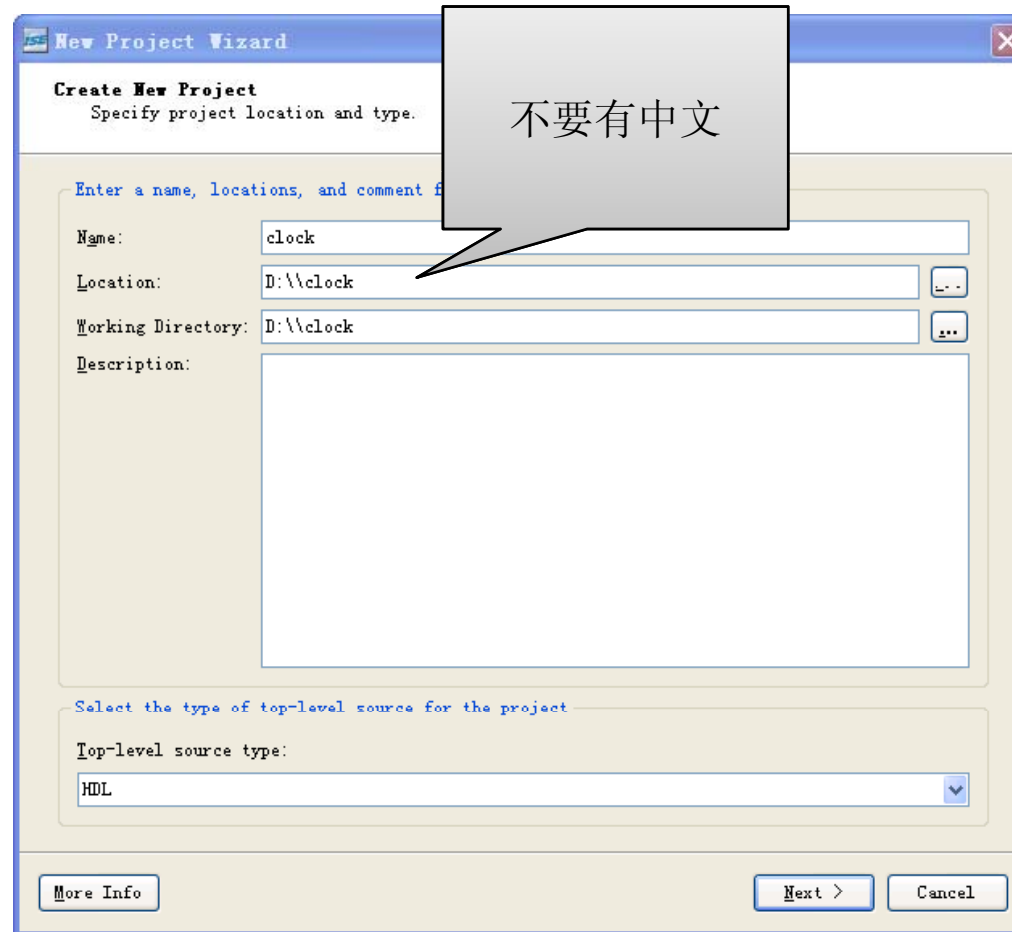
验证(Verification)  
包含后仿真和功能  
仿真  
(Simulation)，。  
功能仿真就是对设  
计电路的逻辑功能  
进行模拟测试，看  
其是否满足设计要  
求，



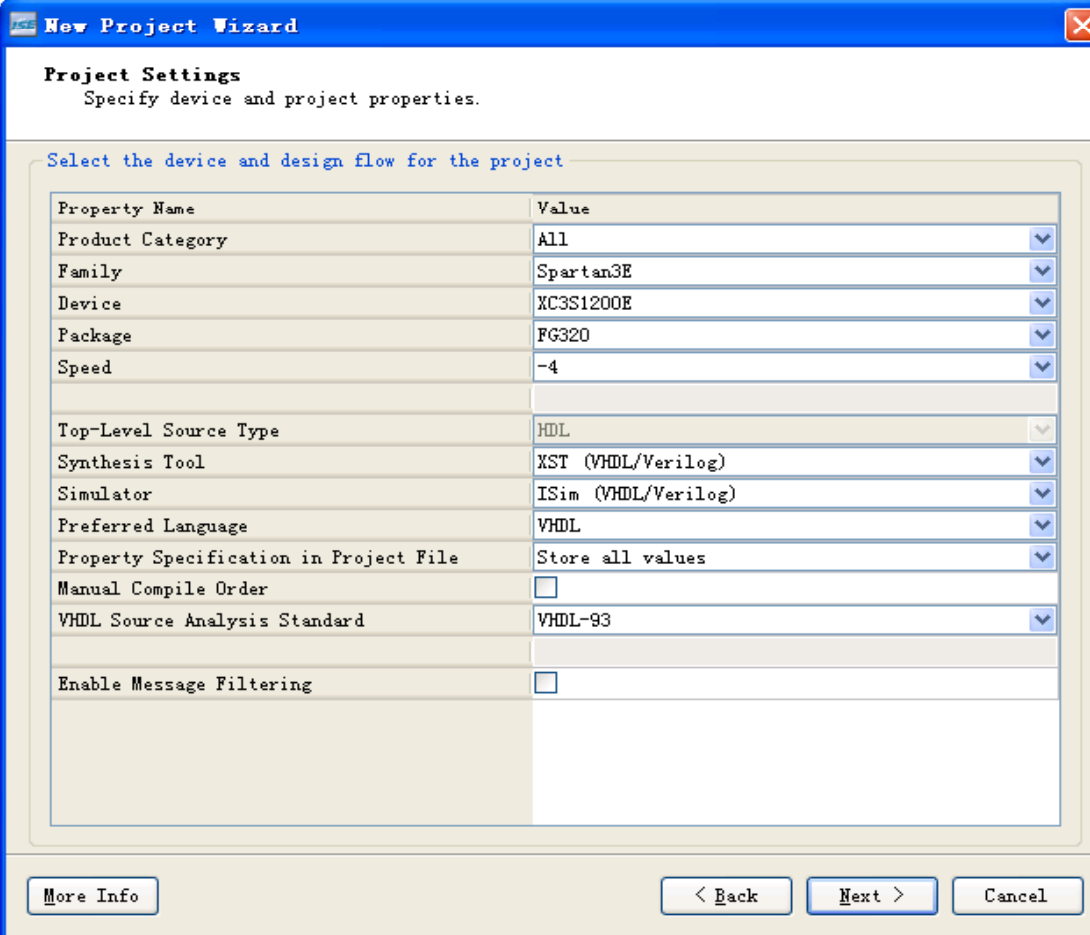
## ● 设计实例



- 新建一个工程（Project）



## 一 设置芯片，模拟器



**New Project Wizard**



**Project Settings**  
Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S1200E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

[More Info](#)      [< Back](#)      [Next >](#)      [Cancel](#)



 **New Project Wizard** 

**Project Summary**  
Project Navigator will create a new project with the following specifications.

Project:

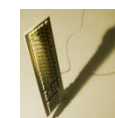
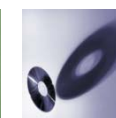
- Project Name: clock
- Project Path: D:\clock
- Working Directory: D:\clock
- Description:
- Top Level Source Type: HDL

Device:

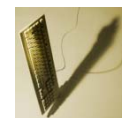
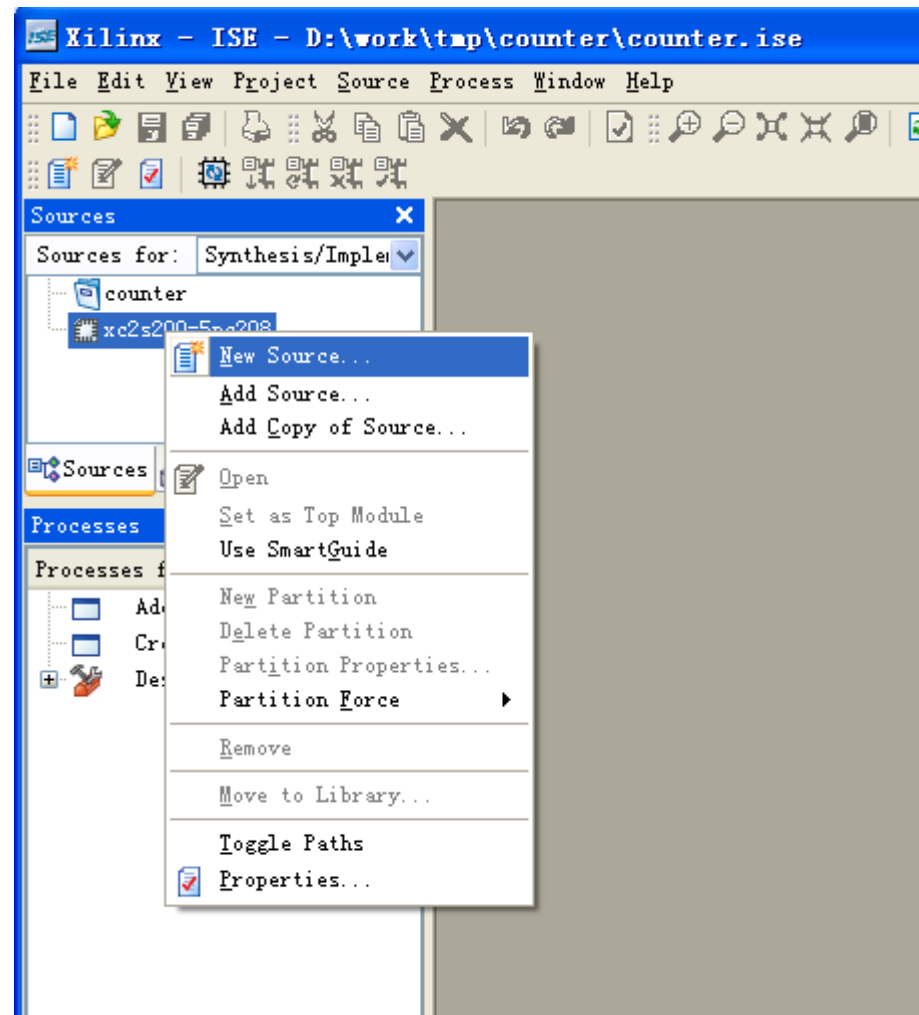
- Device Family: Spartan3E
- Device: xc3s1200e
- Package: fg320
- Speed: -4

Synthesis Tool: XST (VHDL/Verilog)  
Simulator: ISim (VHDL/Verilog)  
Preferred Language: VHDL  
Property Specification in Project File: Store all values  
Manual Compile Order: false  
VHDL Source Analysis Standard: VHDL-93

Message Filtering: disabled



## 一 建立和编辑VHDL源文件





**New Source Wizard**

**Select Source Type**  
Select source type, file name and its location.

- IP (CORE Generator & Architecture Wizard)
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name:  
clock

Location:  
D:\clock

☒ Add to project

More Info      Next >      Cancel



## 一 设置端口

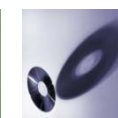
**New Source Wizard**

**Define Module**  
Specify ports for module.

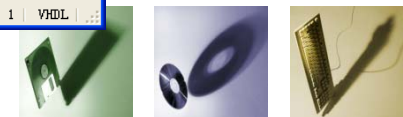
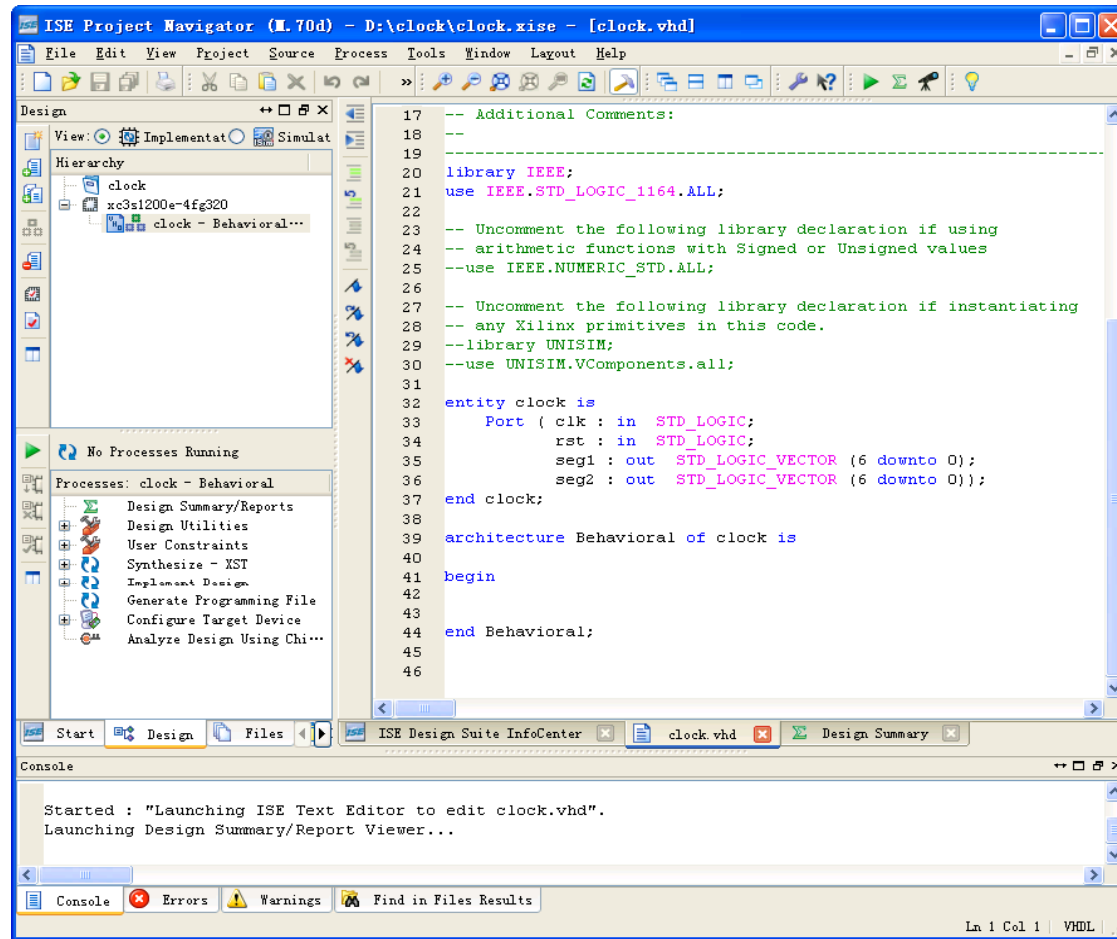
Entity name:

Architecture name:

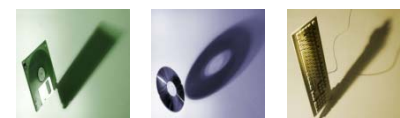
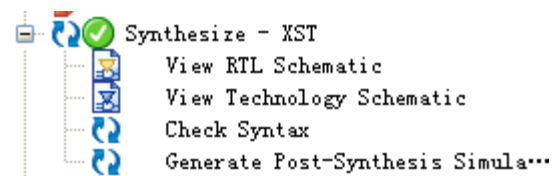
Port Name	Direction	Bus	MSB	LSB
clk	in	<input type="checkbox"/>		
rst	in	<input type="checkbox"/>	0	0
seg1	out	<input checked="" type="checkbox"/>	6	0
seg2	out	<input checked="" type="checkbox"/>	6	0
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		



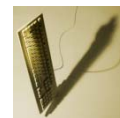
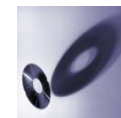
# 一 编辑代码



# 一 逻辑综合

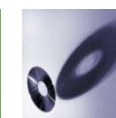
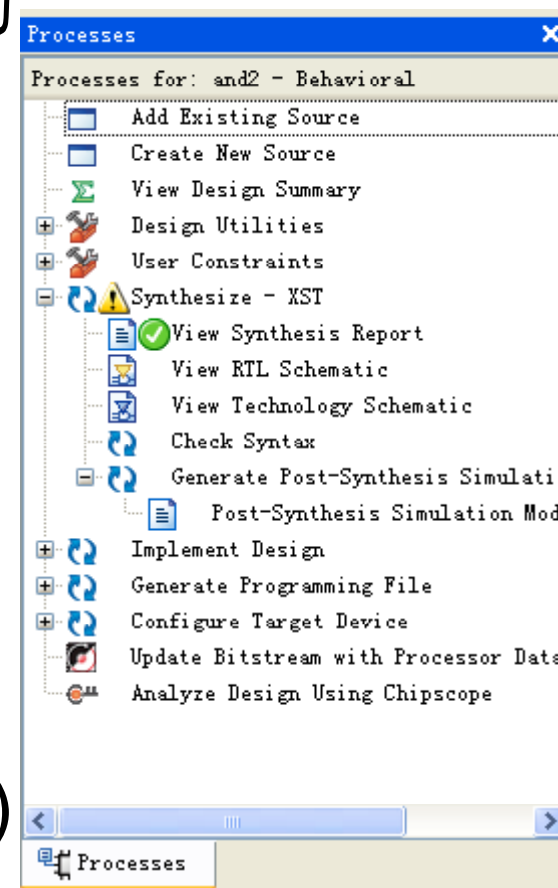


- 综合过程中，主要执行以下三个步骤
  - 语法检查过程，检查设计文件语法是否有错误
  - 编译过程，翻译和优化**HDL**代码，将其转换为综合工具可以识别的元件序列
  - 映射过程，将这些可识别的元件序列转换为可识别的目标技术的基本元件





- synthesis工具可以完成下面的任务：
  - 查看综合报告(view Synthesis Report)
  - 查看RTL原理图 (View RTL schematic)
  - 查看技术原理图 (View Technology Schematic)
  - 源代码语法检查(Check Syntax)
  - 生成综合后仿真模块及报告





## • 查看综合报告

### TABLE OF CONTENTS

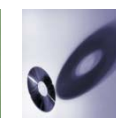
- 1) Synthesis Options Summary
- 2) HDL Compilation
- 3) Design Hierarchy Analysis
- 4) HDL Analysis
- 5) HDL Synthesis
  - 5.1) HDL Synthesis Report
- 6) Advanced HDL Synthesis
  - 6.1) Advanced HDL Synthesis Report
- 7) Low Level Synthesis
- 8) Partition Report
- 9) Final Report
  - 9.1) Device utilization summary
  - 9.2) Partition Resource Summary
  - 9.3) TIMING REPORT

Device utilization summary:

-----

Selected Device : 3s500epq208-4

Number of Slices:	1	out of	4656	0%
Number of 4 input LUTs:	1	out of	9312	0%
Number of IOs:	3			
Number of bonded IOBs:	3	out of	158	1%



Timing Summary:

Speed Grade: -4

Minimum period: No path found  
Minimum input arrival time before clock: No path found  
Maximum output required time after clock: No path found  
Maximum combinational path delay: 6.209ns

Timing Detail:

All values displayed in nanoseconds (ns)

Timing constraint: Default path analysis

Total number of paths / destination ports: 2 / 1

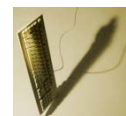
Delay: 6.209ns (Levels of Logic = 3)

Source: a (PAD)

Destination: c (PAD)

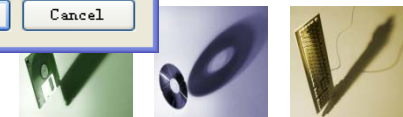
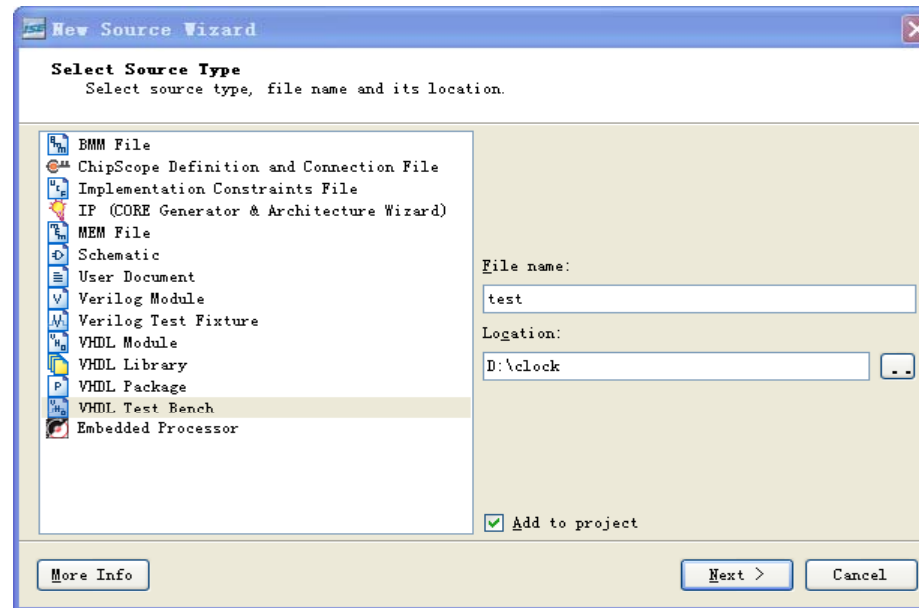
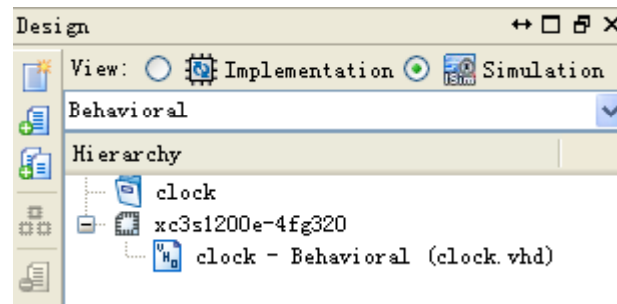
Data Path: a to c

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
IBUF:I->O	1	1.218	0.595	a_IBUF (a_IBUF)
LUT2:IO->O	1	0.704	0.420	c1 (c_OBUF)
OBUF:I->O		3.272		c_OBUF (c)
Total		6.209ns (5.194ns logic, 1.015ns route) (83.7% logic, 16.3% route)		

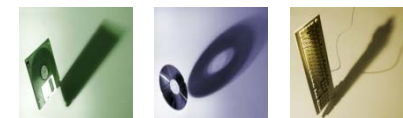
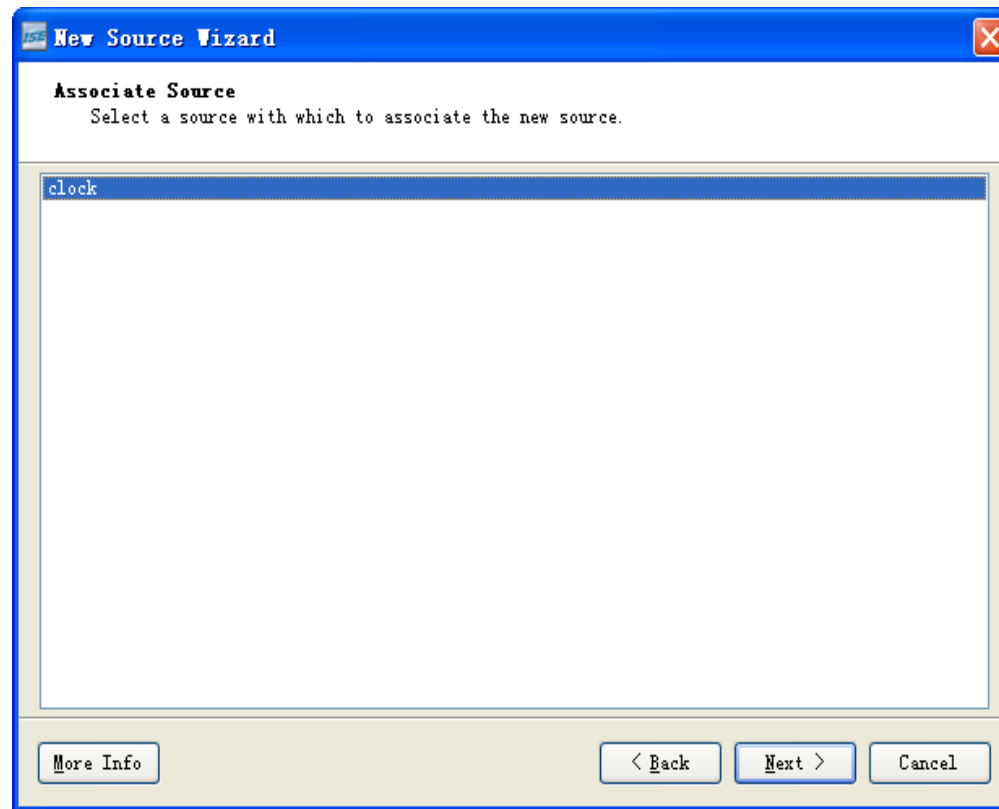


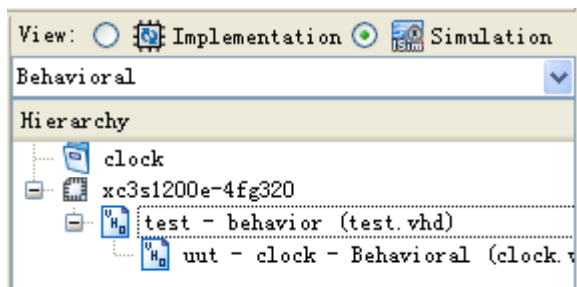


# ● 仿真

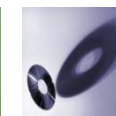


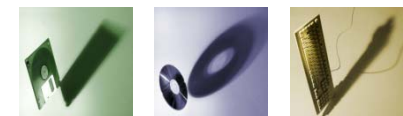
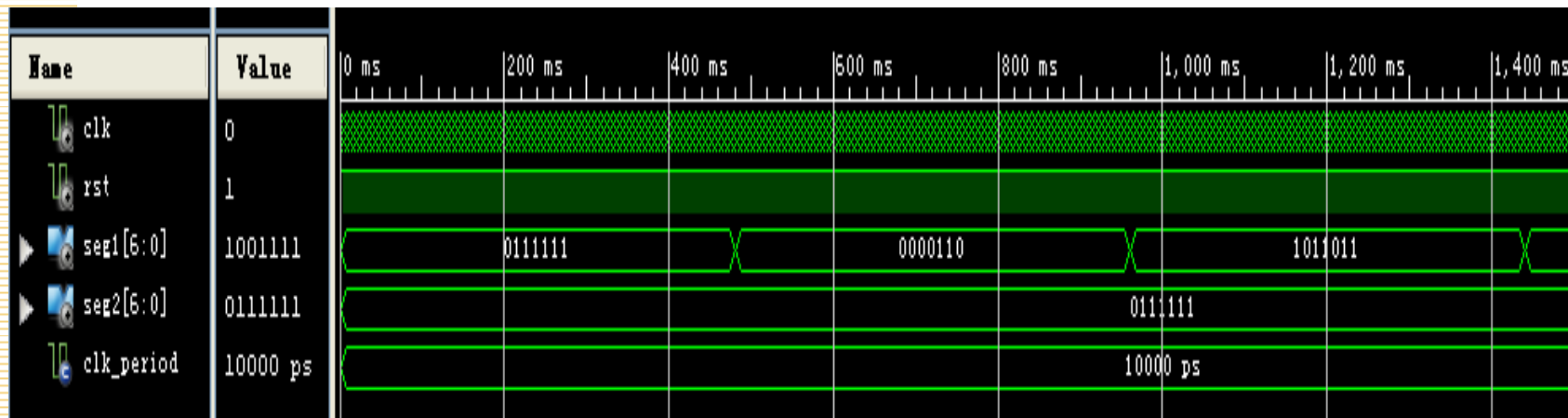
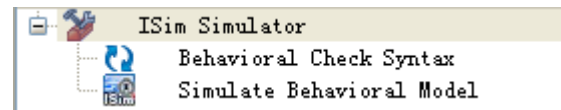
- 出现关联源文件对话框，这里将test bench 关联到clock上，即对clock进行仿真



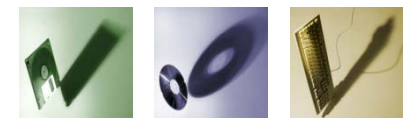
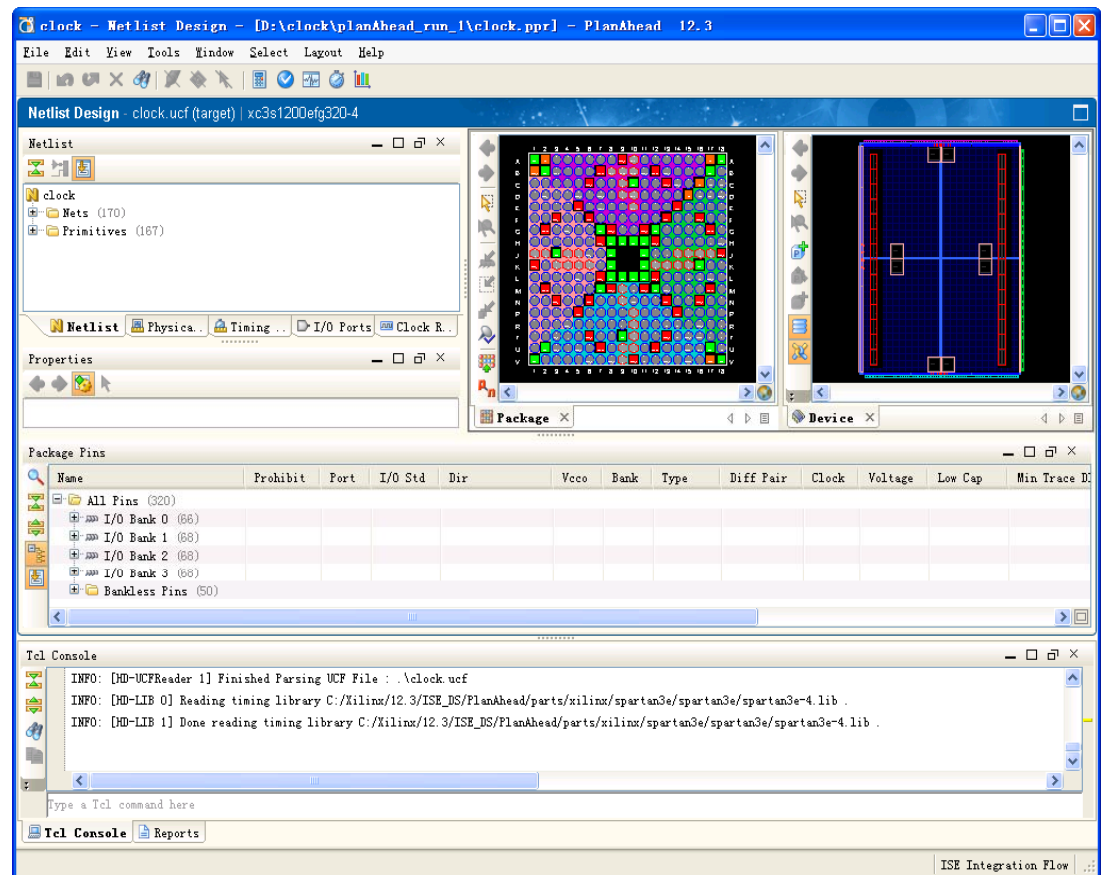
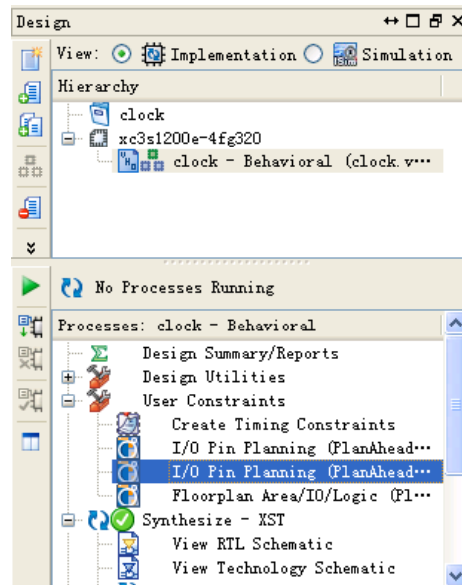


```
63 BEGIN
64
65 -- Instantiate the Unit Under Test (UUT)
66 uut: clock PORT MAP (
67     clk => clk,
68     rst => rst,
69     seg1 => seg1,
70     seg2 => seg2
71 );
72
73 -- Clock process definitions
74 clk_process :process
75 begin
76     clk <= '0';
77     wait for clk_period/2;
78     clk <= '1';
79     wait for clk_period/2;
80 end process;
81
82
83 -- Stimulus process
84 stim_proc: process
85 begin
86     -- hold reset state for 100 ns.
87     wait for 100 ns;
88
89     wait for clk_period*10;
90
91     -- insert stimulus here
92
93     wait;
94 end process;
95
96 END;
```

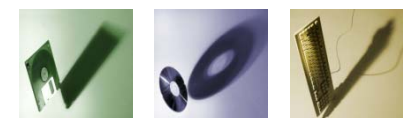
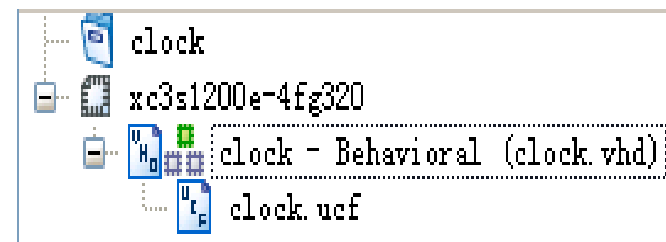




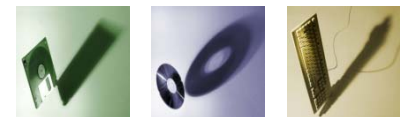
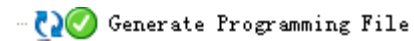
# • 用户约束



I/O Ports				
Name	Dir	Neg	Site	B
All ports (16)				
seg1 (7)	Output			
seg1[0]	Output		C9	
seg1[1]	Output		D9	
seg1[2]	Output		E9	
seg1[3]	Output		F9	
seg1[4]	Output		G9	
seg1[5]	Output		A10	
seg1[6]	Output		B10	
seg2 (7)	Output			
seg2[0]	Output		C7	
seg2[1]	Output		D7	
seg2[2]	Output		E7	
seg2[3]	Output		F7	
seg2[4]	Output		A8	
seg2[5]	Output		E8	
seg2[6]	Output		F8	
Scalar ports (2)				
clk	Input		B9	
rst	Input		V10	

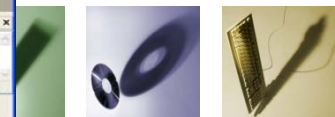
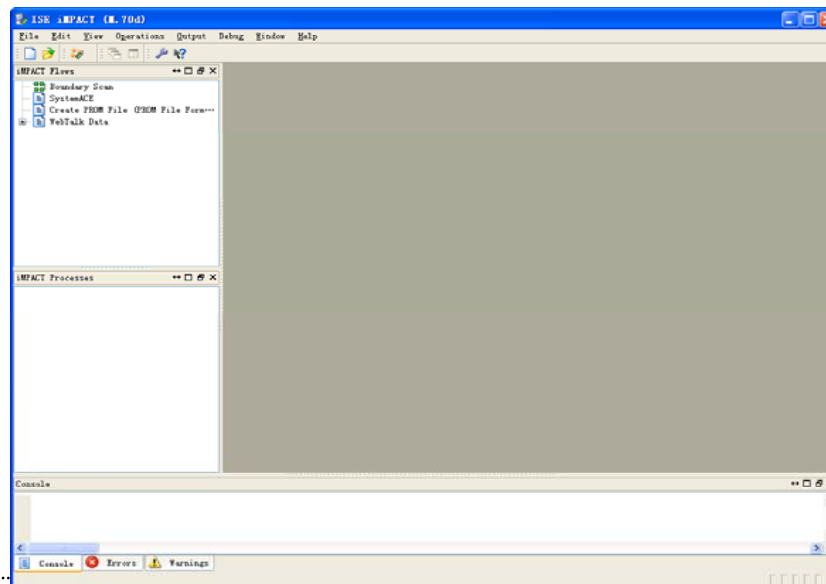


- 设计实现
  - 运行Implement Design
- 配置生成
  - 运行Generate Programme File

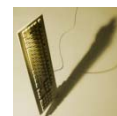
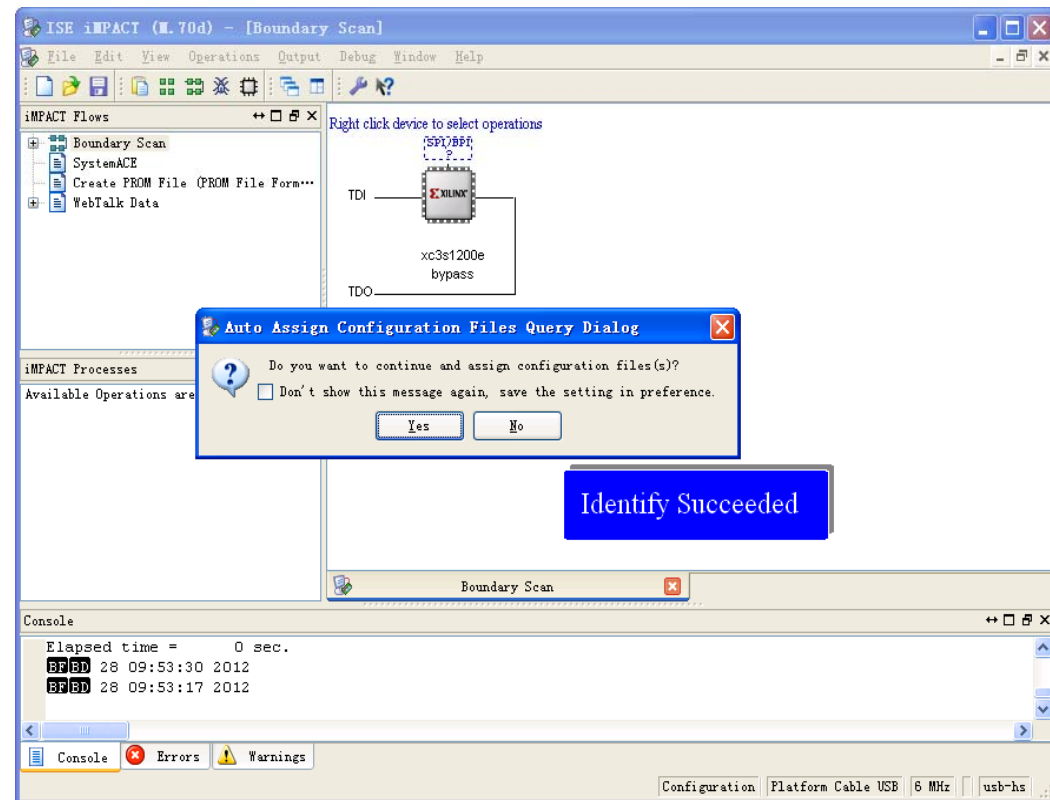


- 下载验证

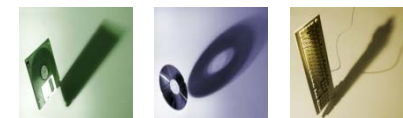
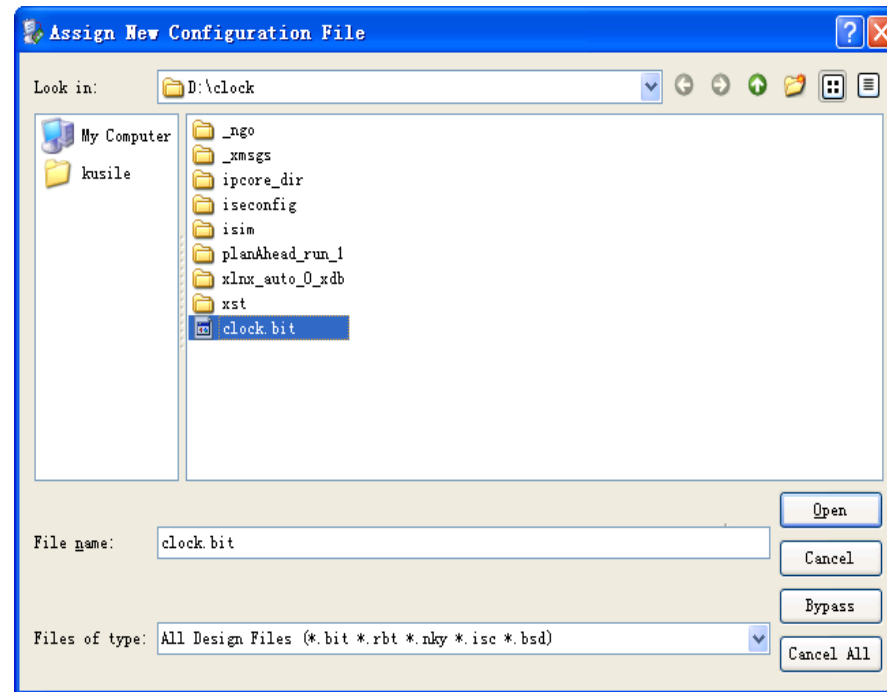
- 双击Configure Device (iMPACT), 启动iMPACT
- 双击左边窗口中的Boundary Scan, 在右边出现的空白窗口中右键单击选择Initialize Chain



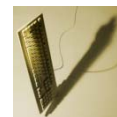
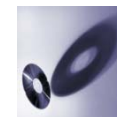
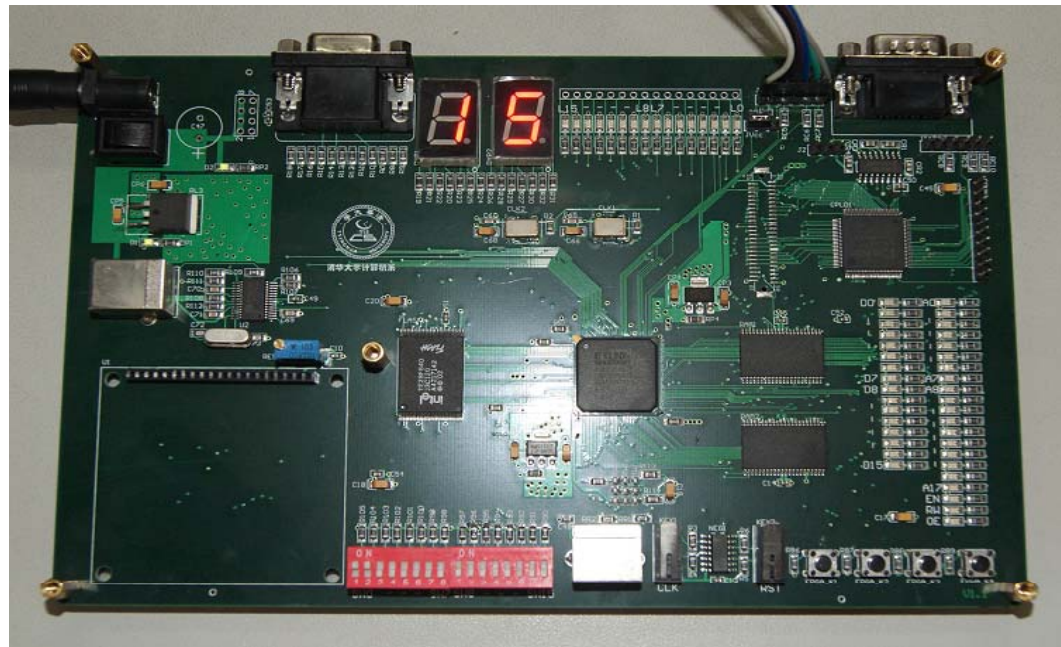
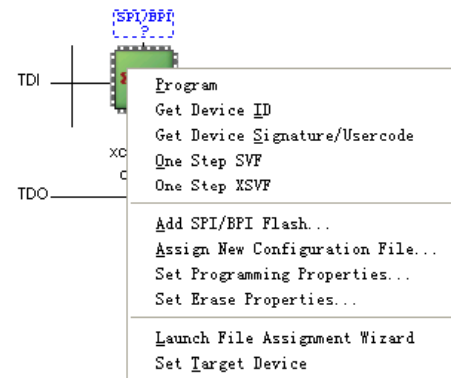




## 一 选择要下载的文件



# 一下载

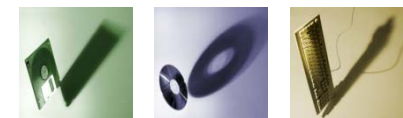


- 注意事项

- 严禁带电拔插芯片或连接线（串口/下载线）
- 不要拆卸实验平台
- 下载不成功，可以检查一下下载线连接是否牢固

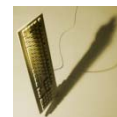
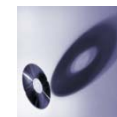


## • USB下载线安装



# 硬件描述语言与实验介绍

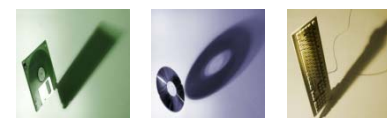
- 1、VHDL语言
- 2、实验环境
- 3、实验介绍

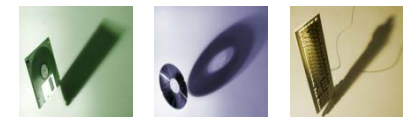
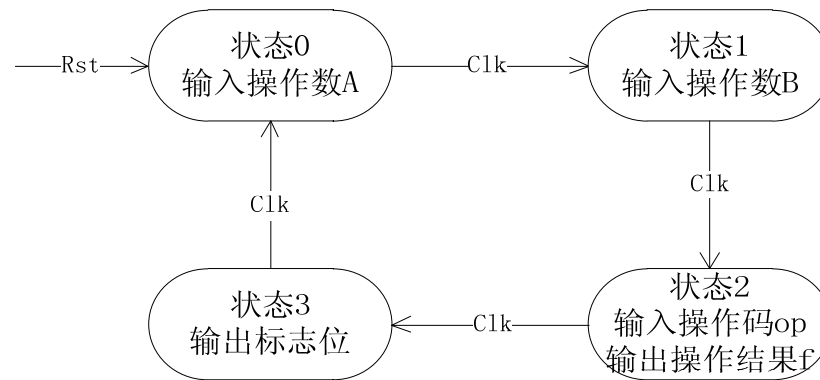
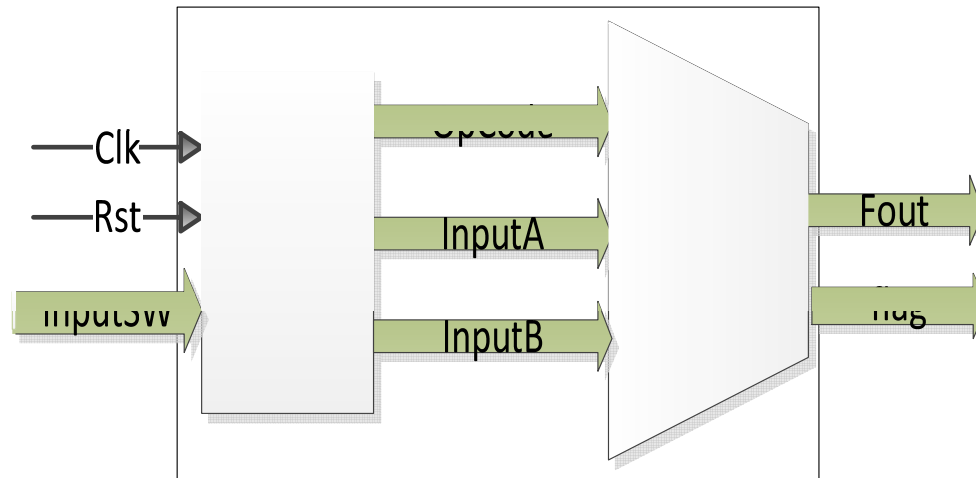


## • 实验一

- 实现ALU，并配合ALU完成一个状态机，按时钟顺序发送操作数、操作码给ALU，并将结果输出在发光二极管上；

操作码	功能	描述
ADD	$A + B$	加法
SUB	$A - B$	减法
AND	A and B	与
OR	A or B	或
XOR	A xor B	与或
NOT	not A	取非
SLL	A sll B	逻辑左移B位
	A sla B	算术左移B位
ROL	A rol B	循环左移B位







- 实验二(5.4/5.5)
  - 实现对内存的访问
  - 实现对串口的访问



- 实验三

- 使用提供的指令集实现一个CPU，运行监控程序
- 23+5
- 流水/多周期



- 实验分组

- 每组一个实验箱
- 最好在宿舍完成，直接可以来实验室检查
- 9区423 62773730-2
- 请班里协调好实验时间，课后与我联系

