第二讲

一、可编程逻辑器件的发展历程

二、可编程逻辑器件的分类

按集成度可编程逻辑器件可分为：

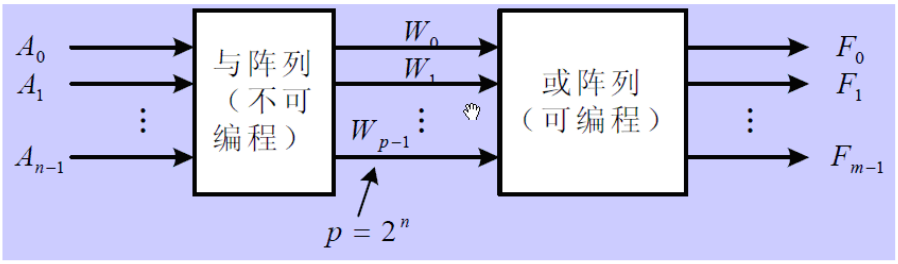
- 简单 PLD：PROM、PAL、PLA、GAL
- 复杂 PLD：CPLD、FPGA

三、简单PLD原理

3.1 PROM

缺点：与阵列固定，造成资源浪费。

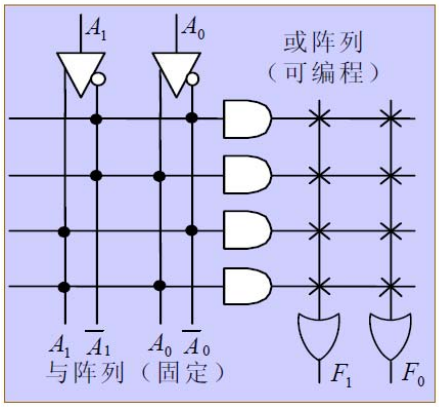
PROM的逻辑阵列结构



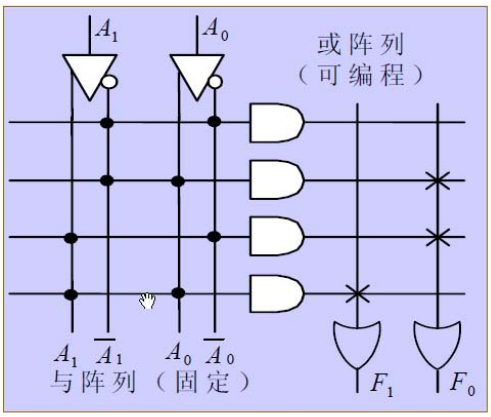
逻辑函数表示：

$$F_0 = M_{p-1,0}W_{p-1} + \cdots + M_{1,0}W_1 + M_{0,0}W_0$$
$$F_1 = M_{p-1,1}W_{p-1} + \cdots + M_{1,1}W_1 + M_{0,1}W_0$$
$$\vdots$$
$$F_{m-1} = M_{p-1,m-1}W_{p-1} + \cdots + M_{1,m-1}W_1 + M_{0,m-1}W_0$$

PROM表达的PLD图阵列



用PROM完成半加器逻辑阵列



$F0 = A0\overline{A1} + \overline{A0}A1$

3.2 PLA

1、PLA 器件除了实现组合逻辑外，还可实现实现逻辑

2、PLA 器件对于逻辑功能的处理比较灵活，但处理逻辑功能较简单的电路是比较浪费，因此在此基础上发展了 GAL 和 PAL 等 PLD 器。

3.3 PAL

3.4 GAL

GAL 有如下优点：

采用 CMOS 的浮栅工艺，器件速度提高，功耗下降

可以重复编程

总结

	阵列		输出
	AND	OR	
PROM	固定的	可编程的	TS、OC
FPLA	可编程的	可编程的	TS、OC、H、L
PAL	可编程的	可编程的	TS、I/O、寄存器型
GAL	可编程的	固定的	由用户定义

3.5 CPLD

3.6 FPGA

3.6.1 PGA可编程逻辑块

CLB： configuration Logic Block

FPGA 通常包括三种基本资源：

- ◆ 可编程逻辑块
- ◆ 可编程输入输出
- ◆ 可编程内部互连资源

除了上述资源外，还包括以下可选资源：

- ◆ 存储器资源
- ◆ 数字时钟管理单元（分频/倍频、数字延迟、时钟锁定）
- ◆ 算法运算单元（高速硬件乘法器、加法器）
- ◆ 多电平标准兼容的 I/O 接口
- ◆ 高速串行 I/O 接口
- ◆ 特殊功能模块（以太网 MAC 等硬件 IP 核）
- ◆ 微处理器（PowerPC405 等硬件处理器 IP 核）

与 PLD 的主要区别：

- ◆ PLD 修改具有固定内联线电路的逻辑功能来进行编程
- ◆ FPGA 修改内连线的布线，更适合实现多级的逻辑功能。

1、找表的可编程结构

2、于反熔丝的可编程结构

是一次性可编程和非丢失性的器件

Actel 的多路开关型结构

3、于 Flash 的可编程结构

4、种可编程单元结构 FPGA 比较

	优点	缺点	典型产品	适用于
SRAM FPGA	可重编程； 技术成熟； 可选产品多； 已广泛使用；	配置时间长； 需片外配置芯片； 功耗较高； 安全性差；	Xilinx 公司的 Spartan、Virtex 系列 FPGA； Altera 公司的 Cyclone、Stratix 系列 FPGA；	一般商用数字系统
反熔丝 FPGA	工作频率高； 上电即运行； 安全性高； 不需外部配置芯片； 抗干扰能力强； 功耗低；	一次性编程；	Actel 公司的 Axcelerator、SX 系列 FPGA； QuickLogic 公司的 Eclipse 系列 FPGA；	国防、航空航天方面的应用
Flash FPGA	可重编程； 上电配置时间极短； 安全性高； 不需外部配置芯片； 功耗较低；	尚未广泛使用；	Lattice 公司的 ispXPGA、LatticeXP 系列 FPGA； Actel 公司的 ProASIC 系列 FPGA；	一般商用，[illegible]保证设计安全性的应用

3.6.2 可编程IOB

- 能兼容 TTL 和 CMOS 多种接口和电压标准
- 可配置为输入、输出、双向、集电极开路和三态等形式
- 能提供适当的驱动电流
- 降低功耗，防止过冲和减少电源噪声
- 支持多种接口电压（降低功耗）

3.6.3 可编程的内部连线资源

3.6.4 器件举例

1、Altra　EX10K 系列

- 逻辑阵列 LAB 是由一系列的相邻 LE 构成的
- 逻辑单元 LE---查找表
- 快速通道(FastTrack)
- I/O 单元与专用输入端口
- 嵌入式阵列块 EAB 是在输入、输出口上带有寄存器的 RAM 块，是由一系列的嵌入式 RAM 单元构成。

2、Xilinx　Virtex II

（1）内部结构概述

FPGA 内部比较复杂，根据 Datasheet 上的分类，主要包括以下几个部分：

- 输入/输出模块 Input/Output Blocks (IOB)
- 可配置逻辑单元 Configurable Logic Blocks (CLB)
- Bram ---- Block SelectRAM
- 18 x 18 乘法器（18-Bit x 18-Bit Multipliers）
- 全局时钟网络（Global Clock Mux）
- 数字时钟管理模块（DCM）
- 布线资源 Routing Resources

（2）全局时钟网络

- 采用原语 BUFGMUX：千万不要将两个时钟通过一个与门或者或门，这样很可能会产生毛刺，影响系统稳定性，如果要对时钟进行操作，例如切换时钟等，请使用 FPGA 内部的专用器件“BUFGMUX”。
- DCM（Digital Clock Manager）:DCM 是 FPGA 内部处理时钟的重要器件，他的左右主要有三个：消除时钟偏斜（Clock De-skew）、频率合成（Frequency Synthesis）和相位调整（Phase Shifting）

3、Altera 与 Xilinx 比较

(1) 结构不同

- A：嵌入式阵列块 EAB、逻辑阵列块 LAB、FastTrack、I/O 单元
- X：LCB(ConfigurableLogic Block)、IOB、布线资源(ICR)。

(2) RAM的使用不同

- A：EAB 可用做大型的 RAM，在 LE 中的查找表也可构成 RAM。
- X：没有专门的 RAM，需要时可由 CLB 中的 LUT 构建。

(3) 基本单元不同

- A:LE(8 个 LE 组成一个 LAB，一个 LE 带有一个触发器)。
- X:CLB(一个 CLB 带有两个触发器)。

(4) 布线不同

- A：FastTRACK，局部互连、进位链和级联链。
- X：单长线、双长线、长线。

(5) 时钟同步方式不同

- A：使用时钟锁定和时钟自举方法。
- X：有一个时钟网络用于时钟的同步，在 V 器件中使用 DLL(Delay-Locked-Loop)技术，缩短了信号输出时间，提高了 I/O 的速度。

(6) 另外 XilinFPGA 有专门的乘法器

3.7 CPLD与FPGA的区别

CPLD 适用于逻辑密集型，FPGA 适用于数据密集型

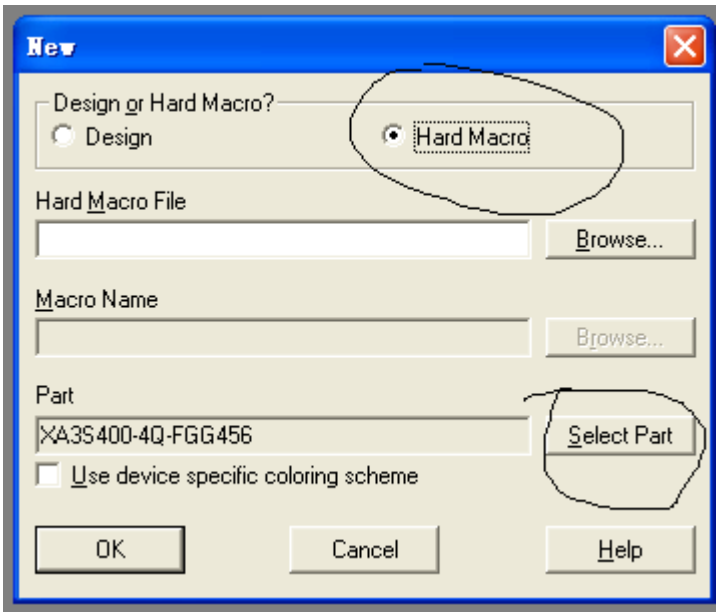
1、逻辑单元
2、互连
3、编程工艺

第三讲 FPGA Editor 操作

1、 打开 FPGA Editor

程序 → Xilinx ISE souite 10.1 → ISE → Accessories → FPGA Editor

2 新建硬核，点击 File → new



3、点击右侧“add”，

第四讲 编码器与译码器

一、编码

1、2^n需要n位二进制码表示

2、目前使用的编码器有普通编码器和优先编码器两种。

- ◆ 普通编码器：任何时刻只允许输入一个有效编码请求信号，否则输出将发生混乱。
- ◆ 优先编码器：允许同时输入两个以上的有效编码请求信号。

二、练习

ISE 使用技巧：

1、出现下图的情况的原因：

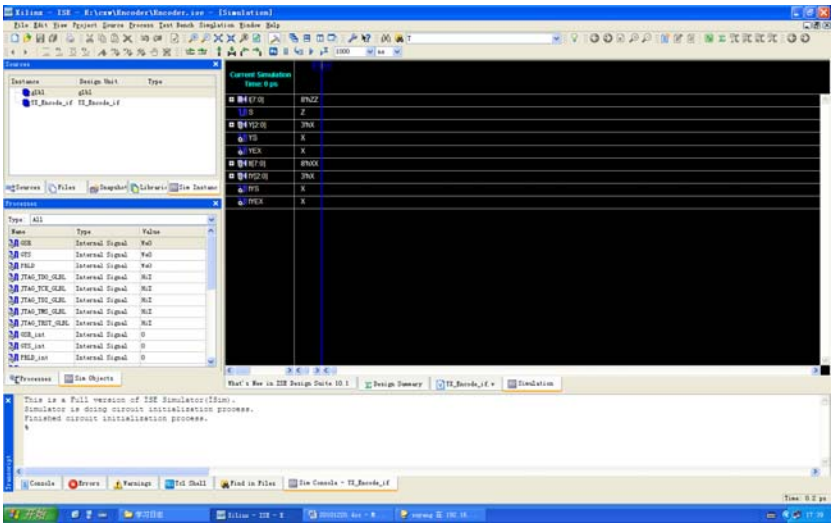
（1） 事先没有选择正确的 verilog test fixture 文件

（2） always 块没有触发条件，例如

```
always
    begin
        Z[3] = ~ Y1[4];
        Z[2] = ~(Y1[2]&Y0[2]);
        Z[1] = ~(Y1[1]&Y0[1]);
        Z[0] = ~(Y1[0]&Y0[0]);
    end
```

应改为：

```
always @(*)
    begin
        Z[3] = ~ Y1[4];
        Z[2] = ~(Y1[2]&Y0[2]);
        Z[1] = ~(Y1[1]&Y0[1]);
        Z[0] = ~(Y1[0]&Y0[0]);
    end
```



2、如下代码综合时出现如下错误：

代码：

```
module Encoder(I, S, Z);
    input  [15:0] I;
```

```
    input         S;
    output [3:0]  Z;
    reg    [3:0]  Z;
    reg    [4:0]  Y0;
    wire   [4:0]  Y1;

    YX_Encode_case  m1(.I(I[15:8]) , .S(S)    , .Y(Y1)),
                       m0(.I(I[7:0])  , .S(Y1[3]), .Y(Y0));

    always @(*)
        begin
            Z[3] = ~ Y1[4];
            Z[2] = ~(Y1[2]&Y0[2]);
            Z[1] = ~(Y1[1]&Y0[1]);
            Z[0] = ~(Y1[0]&Y0[0]);
        end
endmodule
```

错误：

ERROR:HDLCompilers:246 - "Encoder.v" line 32 Reference to vector reg 'Y0' is not a legal net lvalue

原因：Y0 的类型错误，reg 应改为 wire 。