可编程逻辑电路设计

Digital Design Using PLD

duan@pku.edu.cn

二〇〇七年

PLD设计的VHDL语言实现

——时序逻辑电路设计

时序电路设计

- 时序电路设计
- 设计优化
- PLD器件的使用
  - FlEX10K10
  - Cyclone 1C6

1.时序电路设计

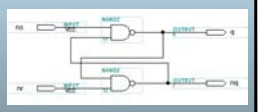
时序逻辑电路特征

- 等效于组合逻辑电路＋记忆性元件
- 输出不仅与当前的输入有关，而且与内部记忆性元件的当前状态有关
- 记忆性元件一般是各种触发器如RS触发器、D触发器以及锁存器
- PLD器件中一般都集成D触发器，便于设计各种时序逻辑电路

RS触发器的实现

- VHDL语言实现：

```
Entity rs is
    port( nr, ns: in std_logic;
          q, nq: out std_logic);
End rs;
Architecture rtl of rs is
    signal tq,tnq: std_logic;
Begin
    tq<=not (tnq and ns);
    tnq<=not (tq and nr);
    q<=tq;
    nq<=tnq;
End rtl;
```



锁存器的实现

❑ **VHDL语言实现：**

```
Entity latch is
    port( clk, d: in std_logic;
          q: out std_logic);
End latch;
Architecture rtl of latch is
begin
    process(clk, d)
    begin
      if (clk='1' ) then
          q<=d;
      end if;
    end process;
End rtl;
```

注意敏感信号列表包括clk和d

D触发器实现

❑ **VHDL语言实现：**

```
Entity dff is
    port( clk, d: in std_logic;
          q: out std_logic);
End dff;
Architecture rtl of dff is
begin
    process(clk)
    begin
      if (clk'event and clk='1') then
          q<=d;
      end if;
    end process;
End rtl;
```

注意敏感信号列表只需包括clk

时钟沿判断

❑ **上升沿判断条件：**
- ❖ clk'event and clk='1'
- ❖ Rising_edge(clk) ——定义于 std_logic_1164

❑ **下降沿判断条件：**
- ❖ clk'event and clk='0'
- ❖ Falling_edge(clk) ——定义于 std_logic_1164

❑ **clk'event**
- ❖ 'event是属性语句， clk'event表示clk信号发生变化

触发器复位/置位

❑ **异步复位：复位优先级高**

```
Entity a_dff is
    port( clk, d: in std_logic;
          a_reset: in std_logic;
          q: out std_logic);
End a_dff;
Architecture rtl of a_dff is
begin
    process(clk, a_reset)
    begin
      if (a_reset='1') then
          q<='0';
      elsif (clk'event and clk='1') then
          q<=d;
      end if;
    end process;
End rtl;
```

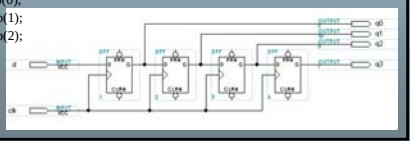
❑ **同步置位：时钟优先级高**

```
Entity s_dff is
    port( clk, d: in std_logic;
          s_preset: in std_logic;
          q: out std_logic);
End s_dff;
Architecture rtl of s_dff is
begin
    process(clk)
    begin
      if (clk'event and clk='1') then
        if (s_preset='1')  then
          q<='1';
        else
          q<=d;
        end if;
      end if;
    end process;
End rtl;
```

常用时序逻辑电路设计(1)

❑ **移位寄存器、延时电路、FIFO**

```
Entity shift4 is
    port( clk, d: in std_logic;
          q: out std_logic_vector(3 downto 0));
End shift4;
Architecture rtl of shift4 is
    signal temp:std_logic_vector(3 downto 0);
begin
    process(clk)
    begin
      if ( clk'event and clk='1') then
          temp(0)<=d;
          temp(1)<=temp(0);
          temp(2)<=temp(1);
          temp(3)<=temp(2);
      end if;
    end process;
    q<=temp;
End rtl;
```



常用时序电路逻辑设计(2)

❑ **加减计数器**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
Entity count4 is
  port( clk: in std_logic;
        reset: in std_logic;
        ena: in std_logic;
        inc: in std_logic;
        q: out std_logic_vector(3 downto 0));
End count4;

Architecture rtl of count4 is
  signal temp:std_logic_vector(3 downto 0);
begin
  process(clk)
  begin
    if (clk'event and clk='1') then
      if (reset='1') then
        temp<="0000";
      elsif (ena='1') then
        if (inc='1') then
          temp<=temp+'1';
        else
          temp<=temp-'1';
        end if;
      end if;
    end if;
  end process;
  q<=temp;
end rtl;
```

常用时序电路逻辑设计(3)

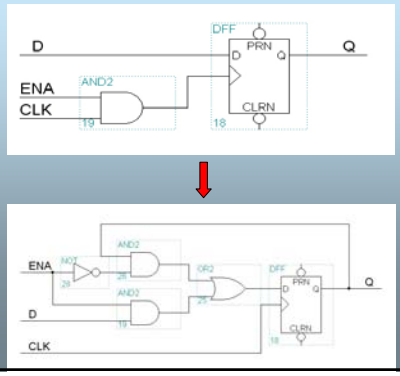
- **n进制减计数器、分频电路**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
Entity freq_divd is
  port( clk: in std_logic;
        reset: in std_logic;
        base: in std_logic_vector(3 downto 0);
        q: out std_logic_vector(3 downto 0));
End freq_divd;

Architecture rtl of freq_divd is
  signal temp:std_logic_vector(3 downto 0);
begin
  process(clk)
  begin
     if (clk'event and clk='1') then
          if (reset='1') then
               temp<="0000";
          elsif (temp= "0000" ) then
               temp<=base;
          else
               temp<=temp-'1';
          end if;
     end if;
  end process;
  q<=temp;
end rtl;
```

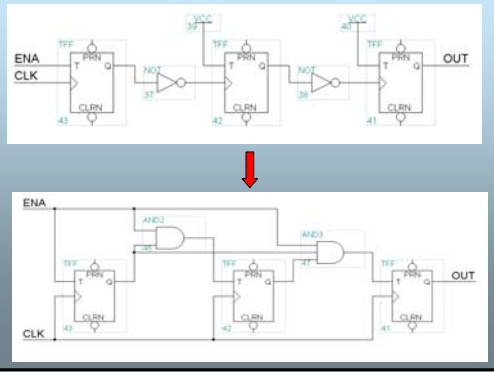
PLD设计若干问题(1)——时钟问题

- 全局时钟是最简单和最可预测的时钟。
- FPGA或CPLD芯片一般提供专用的全局时钟输入引脚以及相应的全局时钟网络，它能够提供器件中最短的时钟到输出的延时以及最小的器件不同部分时钟沿的差异。
- 不推荐在时钟上加控制或者用组合逻辑电路产生时钟——可以将之转换成同步使能信号
- 异步电路→同步电路
- 多个非同源时钟→用高速时钟同步

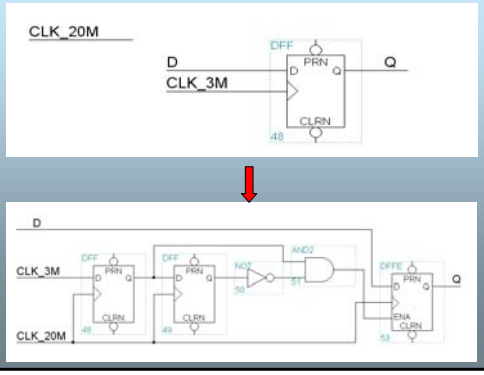
避免直接控制时钟



异步逻辑转换成同步逻辑

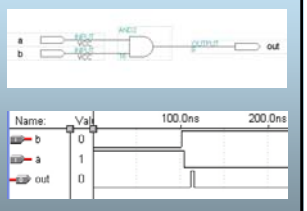


非同源时钟的同步化



PLD设计若干问题(2)——冒险问题

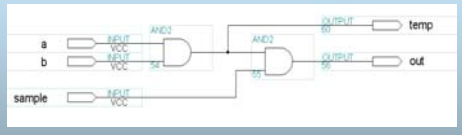
- 组合逻辑电路中，如果若干个输入“同时”发生变化，则有可能产生冒险
- PLD器件内部的组合逻辑电路发生的冒险通常是难以观察的
- 通过加入多余的逻辑门（如两个串连的反门）试图增加局部延迟改善冒险在PLD设计中是不可取的
- 异步复位或置位信号对冒险尤为敏感

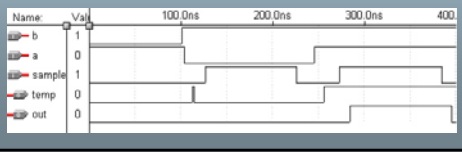


PLD设计若干问题(2)——冒险问题

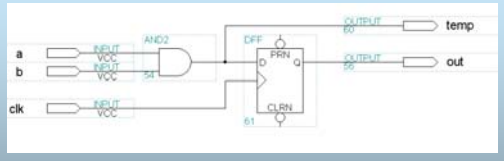
- **消除冒险的方法**
  - 采用适当的数字电路，如同步电路、格雷码计数器等——减小冒险产生的机会
  - 利用时序电路本身固有的抗冒险能力——只有冒险发生在时钟有效沿附近且满足所需的建立与保持时间时，才会发生效应
  - 在组合逻辑信号稳定的时候对其进行采样
    - 用与门采样
    - 用D触发器采样

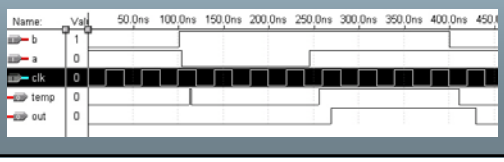
消除冒险——用与门采样





消除冒险——用D触发器采样





2.设计优化

设计的优化

- **基于面积的优化**
  - 资源共享
  - 逻辑优化
  - 串行化(时分复用)
- **基于速度的优化**
  - 并行化
  - 流水线
  - 寄存器配平
  - 改善关键路径
- **工具软件优化**
  - Global Project Logic Synthesis
  - Global Project Timing Requirements

面积优化----资源共享

- **选择乘法器:**
  - 输入: 4位无符号整数multa,multb,multc,multd, 选择信号sel
  - 输出: 8位乘积结果multout
  - 功能: sel为高电平时,multout=multa*multb;否则为multc*multd

- **实现方法一**

```
architecture rtl1 of mult_4_4 is
    signal temp1: std_logic_vector(7 downto 0);
    signal temp2: std_logic_vector(7 downto 0);
begin
    temp1<=multa*multb;
    temp2<=multc*multd;
    multout<=temp1 when sel='1' else temp2;
end rtl;
```

- **MAX+PLUSII综合结果:**
  占用70个LCs

- **实现方法二:共享乘法器**

```
architecture rtl2 of mult_4_4 is
    signal tempa: std_logic_vector(3 downto 0);
    signal tempb: std_logic_vector(3 downto 0);
begin
    multout<=tempa*tempb;
    tempa<=multa when sel='1' else multc;
    tempb<=multb when sel='1' else multd;
end rtl;
```

- **MAX+PLUSII综合结果:**
  占用38个LCs

面积优化----逻辑优化

- **乘法器:**
  - 输入: 4位无符号整数multa,时钟信号clk
  - 疏出: 8位乘积结果multout
  - 功能: 在clk上升沿输出multout<=multa* "1111"

- **实现方法一:**

```
architecture rtl1 of mult_4 is
    signal multb: std_logic_vector(3 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            multb<="1111";
            multout<=multb*multa;
        end if;
    end process;
end rtl;
```

- **综合结果: 占用21个LCs**

- **实现方法二:逻辑优化**

```
architecture rtl2 of mult_4 is
    constant multb: std_logic_vector(3 downto 0):= "1111";
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            multout<=multb*multa;
        end if;
    end process;
    end rtl;
```

- **综合结果: 占用8个LCs**

面积优化----串行化(时分复用)

- **例1:**

```
entity multadd_1 is
port(
    multa0,multa1,multa2: in std_logic_vector(3 downto 0);
    multb0,multb1,multb2: in std_logic_vector(3 downto 0);
    clk: in std_logic;
    multout: out std_logic_vector(9 downto 0));
end multadd_1;

architecture rtl of multadd_1 is
begin

    process(clk)
    begin
        if clk'event and clk='1' then
            multout<=multb0*multa0+multa1*multb1+multa2*multb2;
        end if;
    end process;
end rtl;
```

- **综合结果: 124LCs**

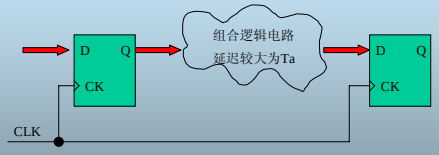
- **例2:**

```
entity multadd_2 is
port(
    multa0,multb0: in std_logic_vector(3 downto 0);
    multa1,multb1: in std_logic_vector(3 downto 0);
    multa2,multb2: in std_logic_vector(3 downto 0);
    start: in std_logic;
    clk: in std_logic;
    multout: out std_logic_vector(9 downto 0));
end multadd_2;

architecture rtl of multadd_2 is
    signal cnt: std_logic_vector(2 downto 0);
    signal temp: std_logic_vector(7 downto 0);
    signal a,b: std_logic_vector(3 downto 0);
    signal multsum: std_logic_vector(9 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if start='1' then
                cnt<="000";
            elsif cnt(2)='0' then
                cnt<=cnt+'1';
            end if;
        end if;
    end process;
    temp<=a*b;
    a<=    multa0 when cnt="000" else
           multa1 when cnt="001" else
           multa2;
    b<=    multb0 when cnt="000" else
           multb1 when cnt="001" else
           multb2;
    process(clk)
    begin
        if clk'event and clk='1' then
            if start='1' then
                multsum<="0000000000";
            else
                multsum<=temp+multsum;
            end if;

            if cnt="011" then
                multout<=multsum;
            end if;
        end if;
    end process;
end rtl;
```

- **综合结果: 81LCs**

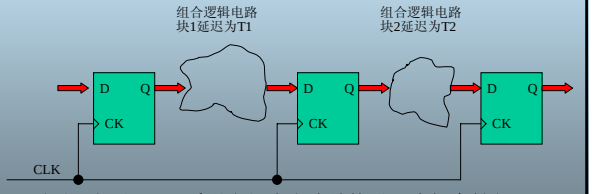
速度优化----流水线

- **未使用流水线: CLK周期应不小于Ta**



使用流水线

- **将组合逻辑部分分成两部分:每块的延时分别为T1,T2,满足T1<Ta,T2<Ta. 此时最小时钟周期为max(T1,T2)<Ta, 系统工作的最大时钟频率有所提高.**



- **根据需要还可以采用多级流水线结构进一步提高性能.**

流水线范例-4个16进制数加法

- 未采用流水线

```
architecture rtl of add4_41 is
    signal temp0,temp1: std_logic_vector(3 downto 0);
    signal temp2,temp3: std_logic_vector(3 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            temp0<=adda0;
            temp1<=adda1;
            temp2<=adda2;
            temp3<=adda3;
            sum<=temp0+temp1+temp2+temp3;
        end if;
    end process;
end rtl;
```

- 编译结果:
  EPF8282ALC84-2芯片上,Fmax=33.78MHz

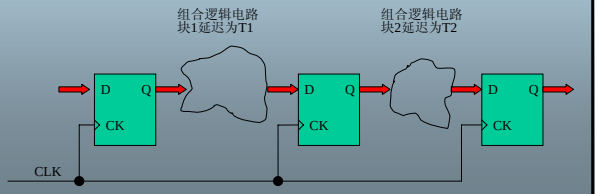
- 采用1级流水线

```
architecture rtl of add4_41 is
    signal temp0,temp1: std_logic_vector(3 downto 0);
    signal temp2,temp3: std_logic_vector(3 downto 0);
    signal sumtemp0: std_logic_vector(4 downto 0)
    signal sumtemp1: std_logic_vector(4 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            temp0<=adda0;
            temp1<=adda1;
            temp2<=adda2;
            temp3<=adda3;
            sumtemp0<=temp0+temp1;
            sumtemp1<=temp2+temp3;
            sum<=sumtemp0+sumtemp1;
        end if;
    end process;
end rtl;
```

- 编译结果:
  EPF8282ALC84-2芯片上,Fmax=59.88MHz

寄存器配平

- 不合理的结构:
  - T1>>T2,则时钟最高频率由组合逻辑块1延迟决定,约为1/T1.
- 合理调整:
  - 如果不关心中间寄存器结果,则调整前后组合逻辑块功能,将块1部分功能调整到组合逻辑块2中,减小T1,增加T2,使二者大致相等,则时钟工作频率可以提高.



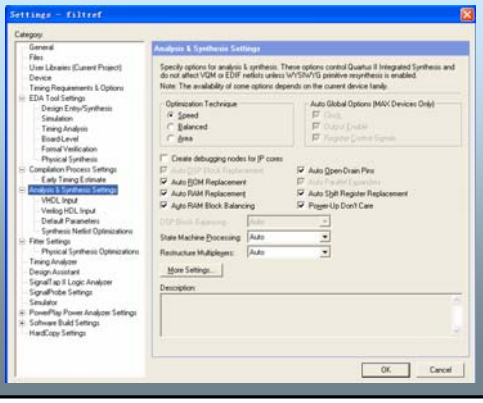
关键路径

- 寻找制约时钟频率的最长延迟的路径,然后采用方法减小该路径的延迟,达到提高最高时钟工作频率的目的.
- MAX+PLUS II软件的定时分析功能为寻找关键路径提供了帮助.

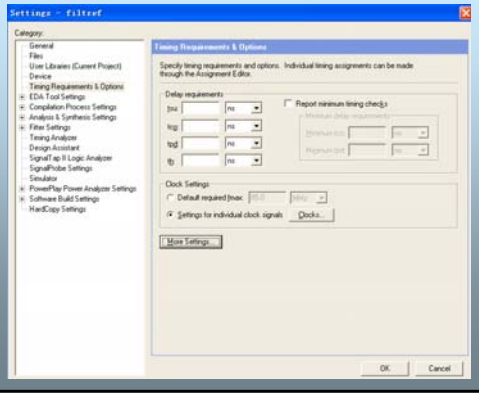
工具软件优化

- MAX+PLUS II优化选项
  - Global Project Logic Synthesis
    - Synthesis Style: Fast, Normal, WYS|WYG
    - Optimize: Area→Speed
    - Carry Chain
    - Cascade Chain
  - Global Project Timing Requirements
    - Tpd
    - Tco
    - Tsu
    - Fmax

Synthesis Options



Timing Requirement



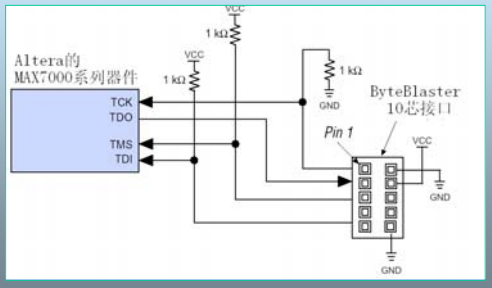
3. PLD器件的使用

- 器件选择
- CPLD器件配置
- FPGA器件配置
- FPGA配置举例
- 电路设计要点

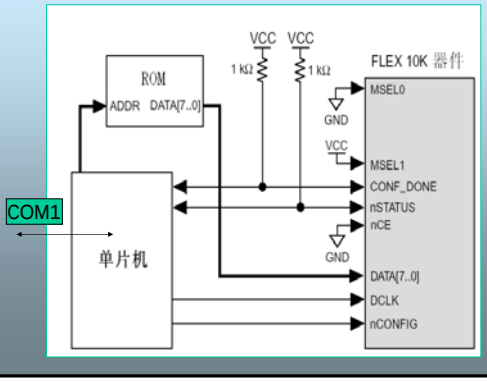
器件选择

- **类型**
  - CPLD：控制
  - FPGA：信号处理运算，缓存器，
- **容量**
  - 宏单元数量
- **管脚**
  - IO数量，电平（3.3/2.5)
- **速度**
- **价格**
- **采购**
- **…**

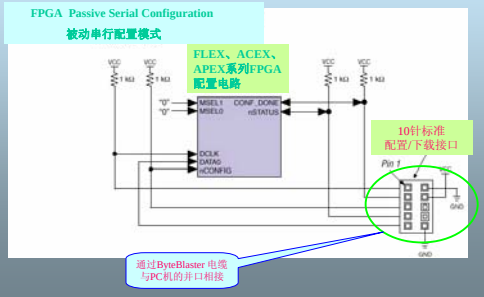
CPLD的ISP方式编程电路



FPGA配置电路 1 —利用单片机



FPGA下载调试电路—ByteBlaster电缆



FPGA配置电路 2 —利用配置器件

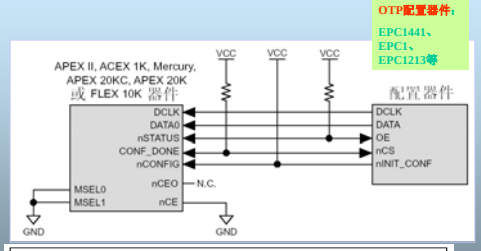
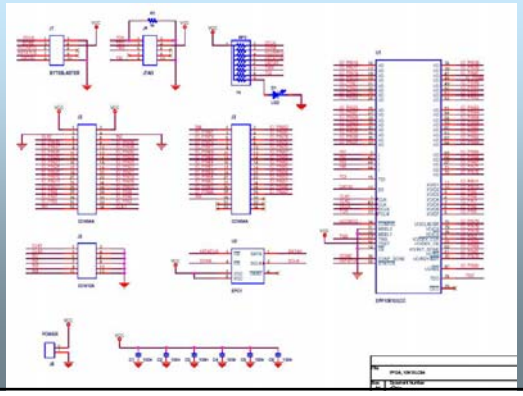


Table 5. Configuration Devices

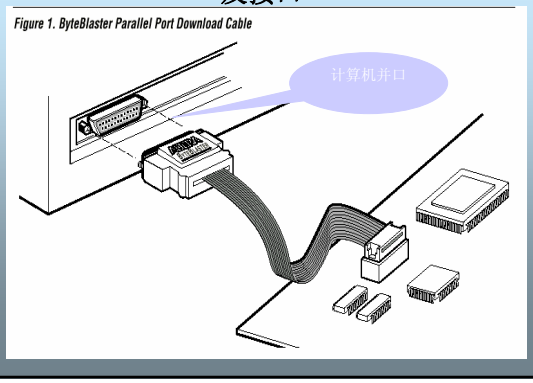
Device	Description
EPC4E	4,194,176 x 1-bit device with 2.5-V or 1.8-V operation
EPC2	1,695,680 × 1-bit device with 5.0-V or 3.3-V operation
EPC1	1,046,496 × 1-bit device with 5.0-V or 3.3-V operation
EPC1441	440,800 × 1-bit device with 5.0-V or 3.3-V operation

FPGA子板电原理图

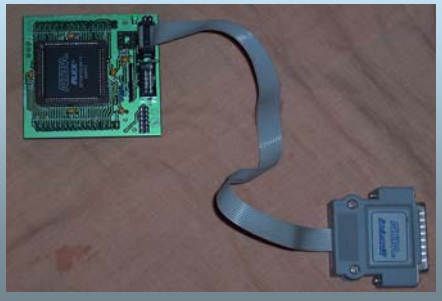


实物图

Altera ByteBlaster(MV)下载调试编程电缆及接口



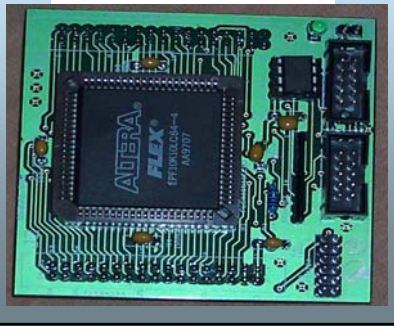
实物图—下载 sof 文件调试



配置器件编程

- 配置文件
  - pof
- 配置器件
  - EPC1
  - EPC1441
- 编程器:
  - SuperPro GX
  - ALL11

实物图—运行



低成本FPGA

Low-Cost FPGAs		
Cyclone II	第二代低成本Cyclone™ FPGA系列用于那些考虑成本多于性能或其他功能的设计。	• Nios® II嵌入式处理器支持 • 嵌入式18x18数字信号处理(DSP)乘法器 • 中等容量的片内存储器 • 中等速度的I/O和存储器接口 • 广泛的IP核支持
Cyclone	第一代低密度、低成本Cyclone FPGA系列	• Nios II嵌入式处理器支持 • 中等容量的片内存储器 • 从低到中等速度的I/O和存储器接口 • 广泛的IP核支持

Features

- **Technology**
  - 1.5V, 0.13-um, Copper Layer, SRAM
- **Density**
  - 20,060 LEs /288Kbits RAM
- **IO Standards**
  - LVTTL,LVCMOS,SSTL-2,SSTL-3
  - 32/64b,33/66M, PCI
  - 640Mbps LVDS I/O
  - DDR-SDRAM(133MHz),SDRAM
- **Clock**
  - PLL:Multiplication/Phaseshift
  - Global Clock
- **IP Core**
  - MegaCore, AMPP

Devices

Table 1–1. Cyclone Device Features

Feature	EP1C3	EP1C4	EP1C6	EP1C12	EP1C20
LEs	2,910	4,000	5,980	12,060	20,060
M4K RAM blocks (128 × 36 bits)	13	17	20	52	64
Total RAM bits	59,904	78,336	92,160	239,616	294,912
PLLs	1	2	2	2	2
Maximum user I/O pins (1)	104	301	185	249	301

Table 1–2. Cyclone Package Options & I/O Pin Counts

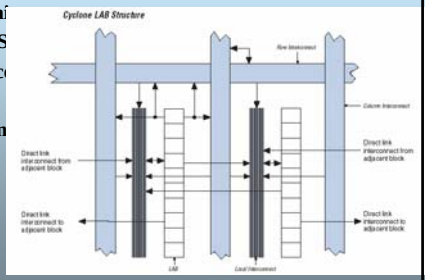
Device	100-Pin TQFP (1)	144-Pin TQFP (1), (2)	240-Pin PQFP (1)	256-Pin FineLine BGA	324-Pin FineLine BGA	400-Pin FineLine BGA
EP1C3	65	104				
EP1C4					249	301
EP1C6		98	185	185		
EP1C12			173	185	249	
EP1C20					233	301

Table 1–3. Cyclone QFP & FineLine BGA Package Sizes

Dimension	100-Pin TQFP	144-Pin TQFP	240-Pin PQFP	256-Pin FineLine BGA	324-Pin FineLine BGA	400-Pin FineLine BGA
Pitch (mm)	0.5	0.5	0.5	1.0	1.0	1.0
Area (mm²)	256	484	1,024	289	361	441
Length × width (mm × mm)	16 × 16	22 × 22	34.6 × 34.6	17 × 17	19 × 19	21 × 21

Logic Array Block

- **10 Les**
- **LE Carry Chai**
- **LAB Control S**
- **A Local Interc**
- **LUT Chain**
- **Register Chain**

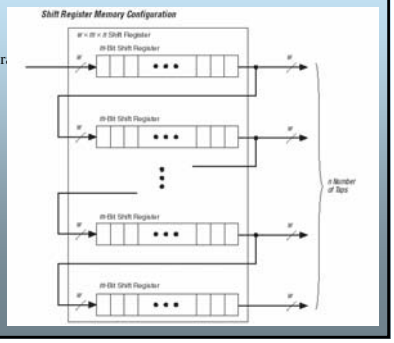


Embedded Memory

- 4,608 RAM bits
- 200 MHz performance
- True dual-port memory
- Simple dual-port memory
- Single-port memory
- Byte enable
- Parity bits
- Shift register
- FIFO buffer
- ROM
- Mixed clock mode

Shift Register Chain

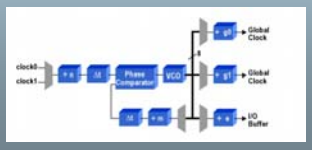
- **DSP**
  - Pseudo Random Gener
  - Filter
  - Correlation



PLL

Cyclone PLL Features

Feature	PLL Support
Clock multiplication and division	m/(n × post-scale counter) (1)
Phase shift	Down to 125-ps increments (2), (3)
Programmable duty cycle	Yes
Number of internal clock outputs	2
Number of external clock outputs	One differential or one single-ended (4)



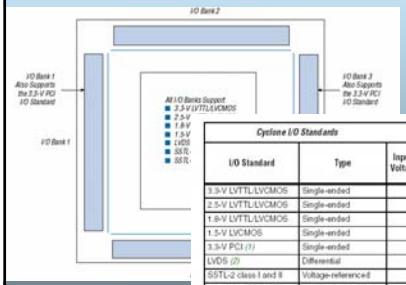
PLL Functions

- **Clock Multiplication**
  - State Machine, PipeLine,Multi Phase/TD
- **Phase Shift**
  - Setup/Hold Time
- **Duty**
  - DDR
- **Enable/Disable**
  - Low Power
- **External Output**

PLL I/O Standards

I/O Standard	CLK Input	EXTCLK Output
3.3-V LVTTL/LVCMOS	✓	✓
2.5-V LVTTL/LVCMOS	✓	✓
1.8-V LVTTL/LVCMOS	✓	✓
1.5-V LVCMOS	✓	✓
3.3-V PCI	✓	✓
LVDS	✓	✓
SSTL-2 class I	✓	✓
SSTL-2 class II	✓	✓
SSTL-3 class I	✓	✓
SSTL-3 class II	✓	✓
Differential SSTL-2		✓

IO Banks



Cyclone I/O Standards

I/O Standard	Type	Input Reference Voltage (V_{REF}) (V)	Output Supply Voltage (V_{CCIO}) (V)	Board Termination Voltage (V_{TT}) (V)
3.3-V LVTTL/LVCMOS	Single-ended	N/A	3.3	N/A
2.5-V LVTTL/LVCMOS	Single-ended	N/A	2.5	N/A
1.8-V LVTTL/LVCMOS	Single-ended	N/A	1.8	N/A
1.5-V LVCMOS	Single-ended	N/A	1.5	N/A
3.3-V PCI (1)	Single-ended	N/A	3.3	N/A
LVDS (2)	Differential	N/A	2.5	N/A
SSTL-2 class I and II	Voltage-referenced	1.25	2.5	1.25
SSTL-3 class I and II	Voltage-referenced	1.5	3.3	1.5
Differential SSTL-2 (3)	Differential	1.25	2.5	1.25

Configuration

Cyclone FPGA Configuration Schemes

Configuration Scheme	Description
Active serial (AS) configuration	Configuration using: • Serial configuration devices (EPCS1 or EPCS4)
Passive serial (PS) configuration	Configuration using: • Enhanced configuration devices (EPC4, EPC8, and EPC16) • EPC2, EPC1 configuration devices • Intelligent host (microprocessor) • Download cable
JTAG-based configuration	Configuration via JTAG pins using: • Download cable • Intelligent host (microprocessor) • Jam™ Standard Test and Programming Language (STAPL)

Selecting Cyclone Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	AS
0	1	PS
0	0 or 1 (1)	JTAG-based (2)

Configuration

Cyclone FPGA Configuration Schemes

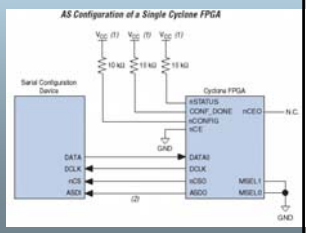
Configuration Scheme	Description
Active serial (AS) configuration	Configuration using: • Serial configuration devices (EPCS1 or EPCS4)
Passive serial (PS) configuration	Configuration using: • Enhanced configuration devices (EPC4, EPC8, and EPC16) • EPC2, EPC1 configuration devices • Intelligent host (microprocessor) • Download cable
JTAG-based configuration	Configuration via JTAG pins using: • Download cable • Intelligent host (microprocessor) • Jam™ Standard Test and Programming Language (STAPL)

Selecting Cyclone Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	AS
0	1	PS
0	0 or 1 (1)	JTAG-based (2)

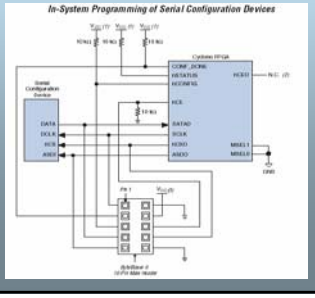
AS Configuration

- **DCLK**
  - 14M-17M-20M
- **Time**
  - EP1C3:0.628Mbit
  - DCLK=14M,71ns
  - 47ms
- **>100ms?**

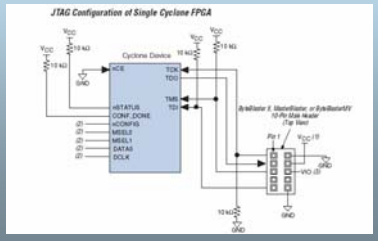


How to Program Conf Device?

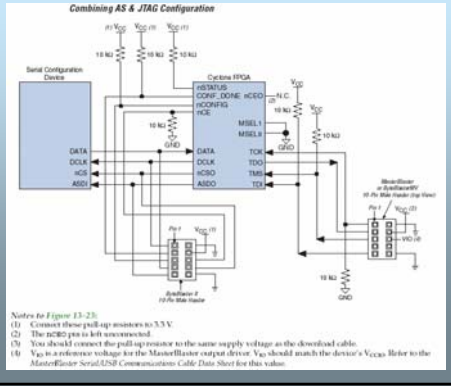
- **Byte Blaster II**



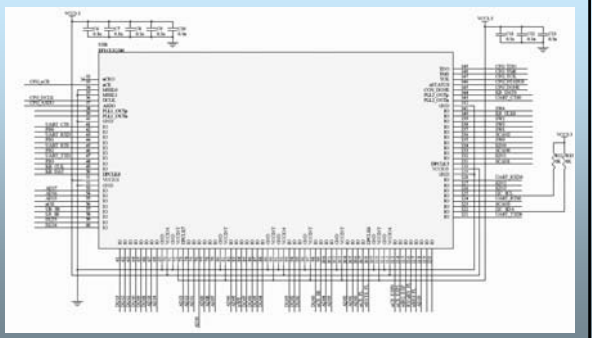
JTAG



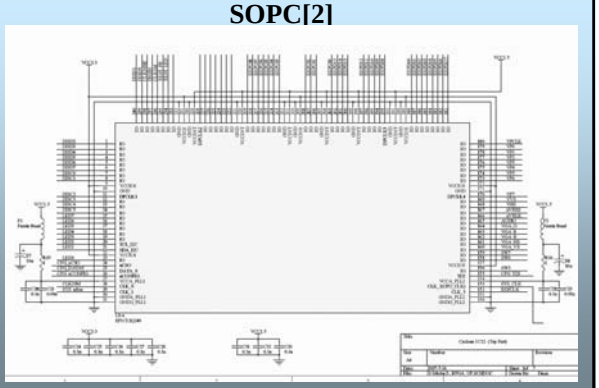
Combining AS&JTAG



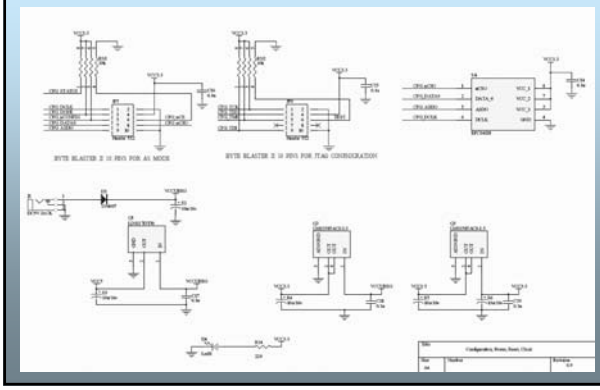
SOPC[1]



SOPC[2]



SOPC[3]



SOPC[4]

PLD电路设计注意事项

- **电源滤波**
- **调试输出管脚**
- **不用管脚的处理**
- **上电过程**
  - VCCINT,VCCIO
  - 配置时间 < 系统复位时间
- **设计注意事项**
  - 同步时序电路
  - 复位输入 Global Reset input
  - 时钟考虑

实验三

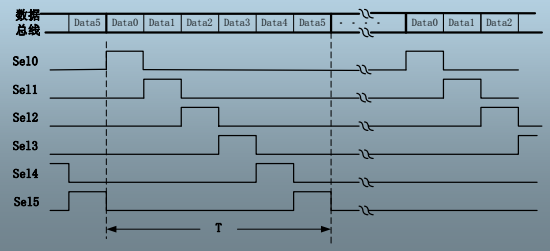
- **必做：**
  - 1、扫描显示电路
  - 2、4位十进制加/减计数器
- **选做：**
  - 1、脉宽测量电路
  - 2、用流水线设计一个8×8的高速乘法器
  - 3、倒计时装置
- **思考题**
  - 扫描显示的闪烁效果能否通过直接将扫描周期T增大来实现？
  - 矩阵键盘的扫描式输入原理及实现方法。

实验原理

- **显示器组的两种显示模式**
  - 独立显示模式：
    - 各个显示器独立、连续地显示
    - 优点：控制简单
    - 缺点：占用太多I/O
  - 扫描显示模式：
    - 所有显示器被看成一个整体，共用数据线，采用时分的方式不连续地显示
    - 优点：占用的I/O少
    - 缺点：控制复杂，需要一定的逻辑资源实现控制

实验原理

- **扫描显示——基本原理**
  - 将每个扫描周期分成若干时隙，每个时隙中选通一个显示器并在数据总线上传输相应的显示数据
  - 扫描周期要较短——避免人眼察觉



实验原理

- **扫描显示——闪烁**
  - 通过对数据总线或者选通信号加周期性的控制信号实现闪烁
  - 闪烁周期较长
  - 可控制闪烁时亮灭的时间比例——控制信号的占空比