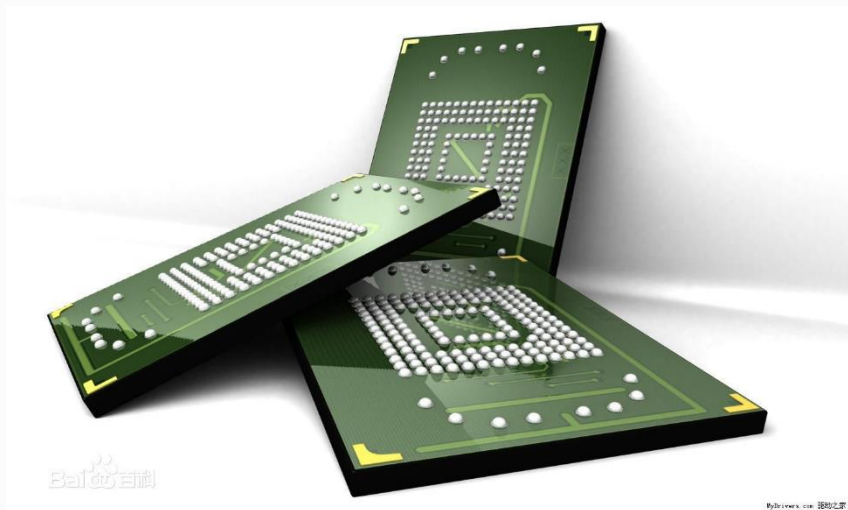




基于Vivado的FPGA开发





专业课程体系

专业/工程基础课

通识教育课
学科教育课
...



路

场

信号



专业方向课
专业选修课



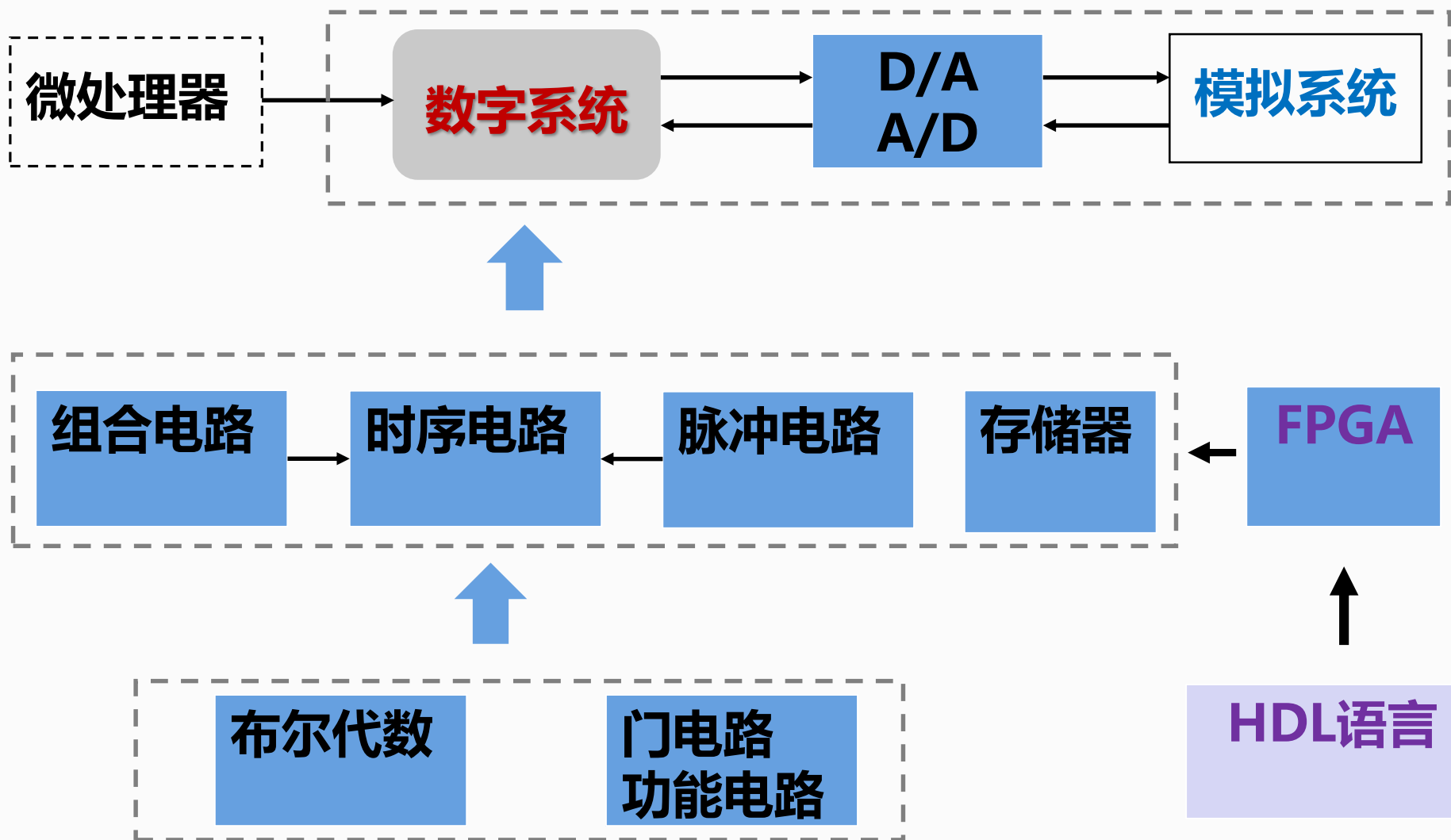
- ✓ 思想政治类
- ✓ 军事体育类
- ✓ 外国语言类
- ✓ 计算机类
- ✓ 数理基础类
- ✓

- ✓ 电路
- ✓ 数字逻辑电路
- ✓ 模拟电子线路
- ✓ 模拟电路EDA
- ✓ 数字系统综合设计
- ✓ 通信电子线路
- ✓

- ✓ 通信工程
- ✓ 电子信息工程
- ✓ 电子科学与技术
- ✓ 光电信息科学与工程
- ✓ 微电子科学与工程
- ✓



电子系统

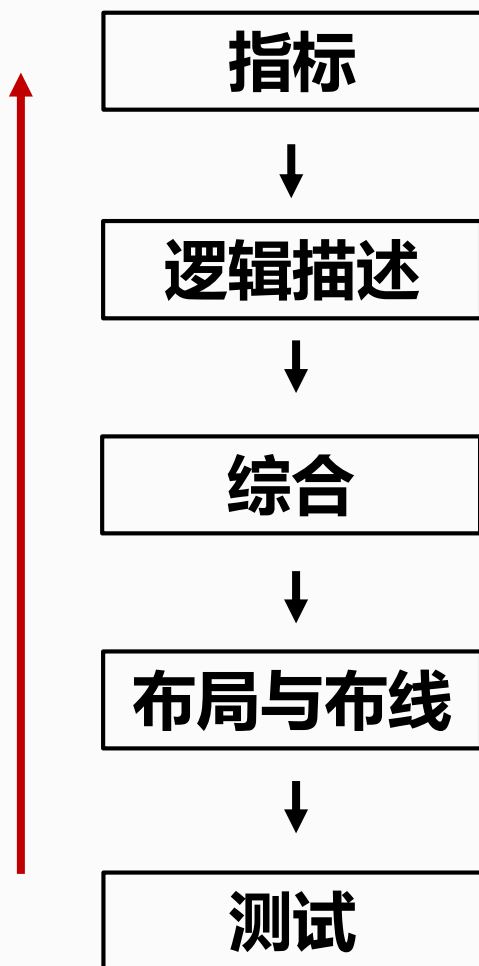




数字电路设计流程

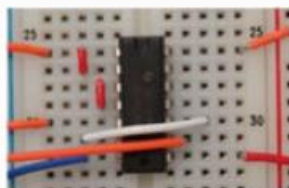
■ 传统数字电路设计流程

抽象



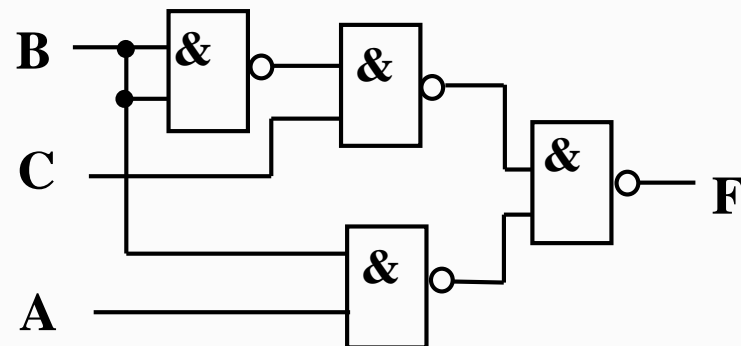
设计一个素数检测器 (0~7)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



BC	00	01	11	10
A=0		1		
A=1		1	1	1

$$F = \overline{A}B\overline{C}$$





数字电路设计流程

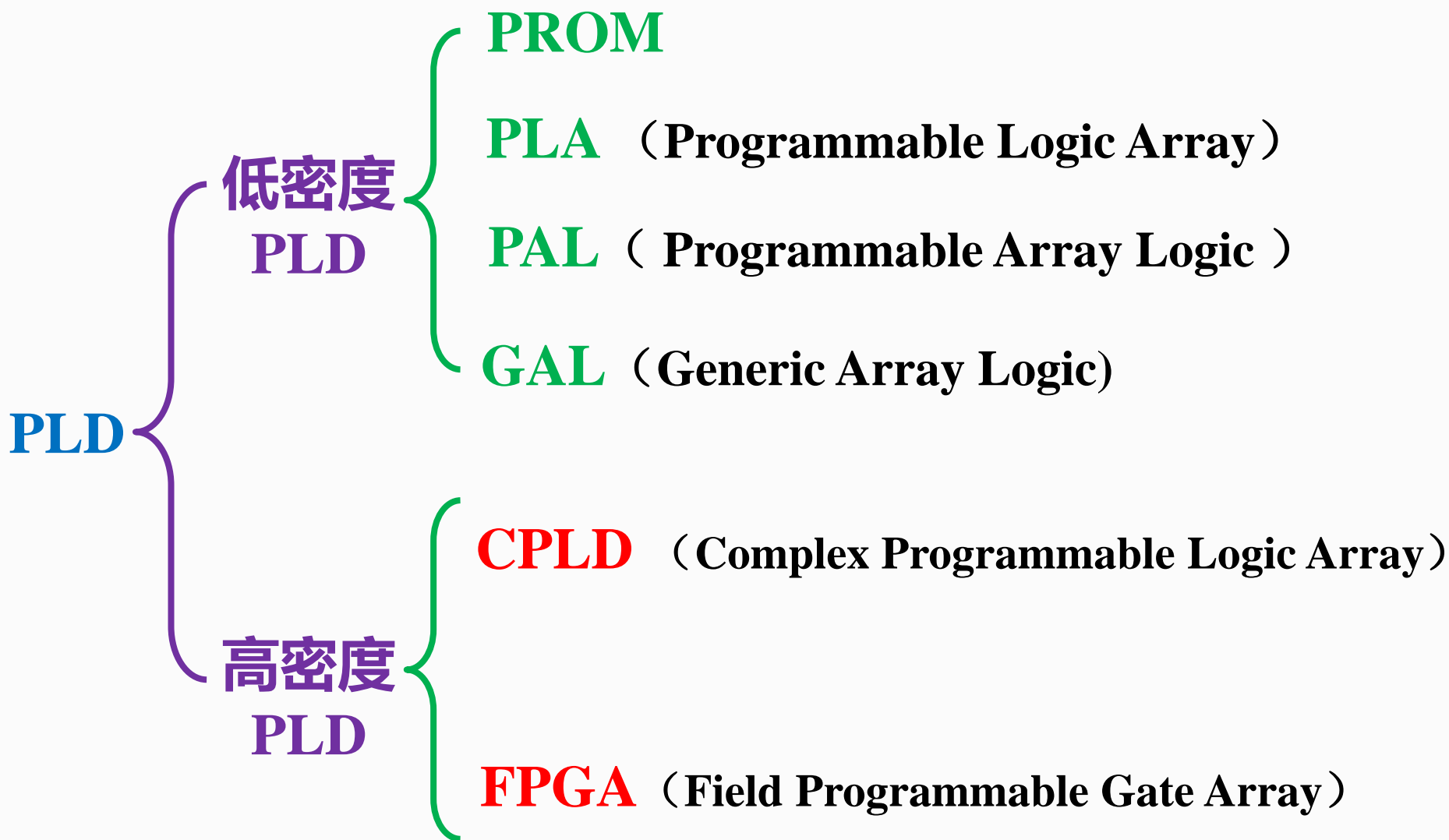
■ 现代数字电路设计流程

- ✓ 抽象思维
- ✓ 计算思维





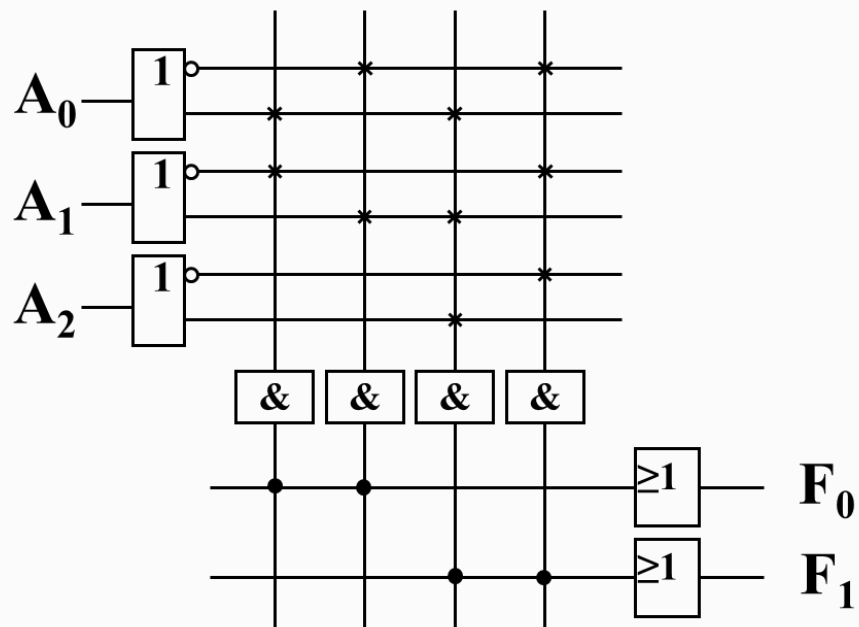
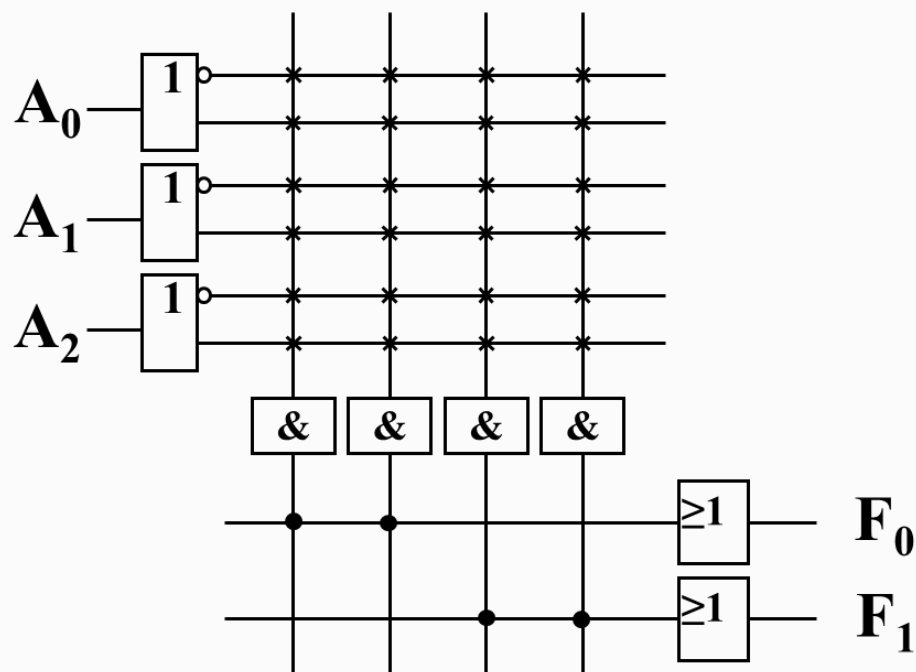
可编程逻辑器件(PLD)





CPLD的基本结构

与阵列可编程，或阵列固定

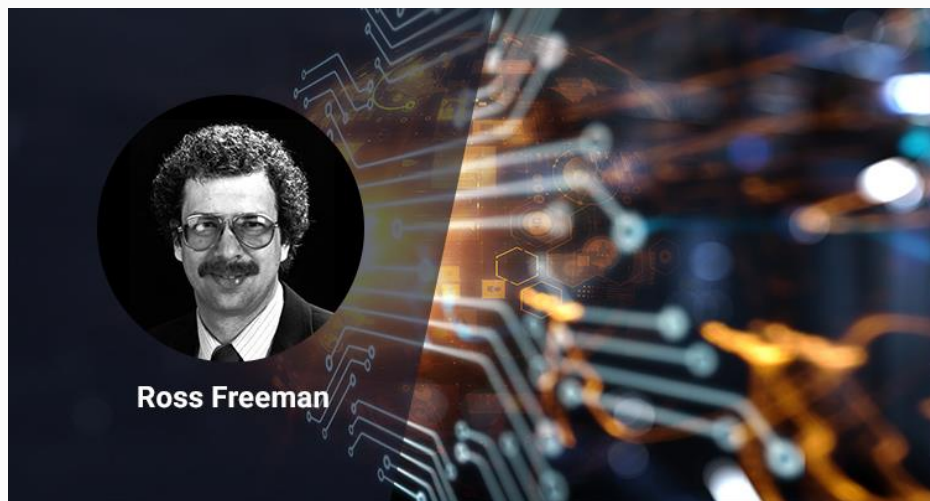


$$F_0 = A_0 \bar{A}_1 + \bar{A}_0 A_1$$

$$F_1 = A_0 A_1 A_2 + \bar{A}_0 \bar{A}_1 \bar{A}_2$$



FPGA发展



Ross Freeman, was an American electrical engineer and inventor, and co-founder of the leading FPGA developer Xilinx.



1985年，全球第一款FPGA产品XC2064——采用 $2\mu\text{m}$ 工艺，包含64个逻辑模块和85000个晶体管，门数量不超过1000个。



FPGA发展

■ Xilinx公司

开发工具： ISE、Vivado

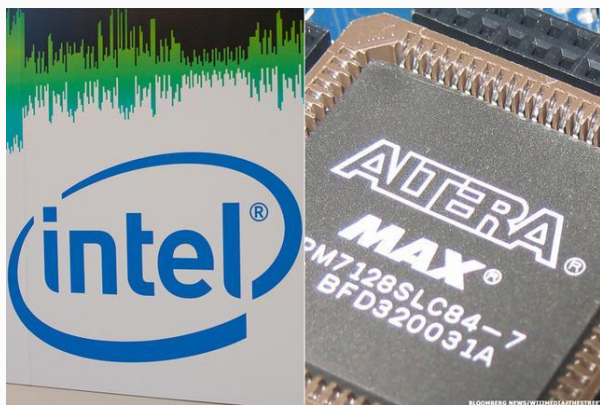
■ Altera公司

开发工具： Quartus



FPGA发展

- 2015 年 6 月，Intel 宣布以 167 亿美元的价格，收购全球第二大 FPGA 厂商 Altera。

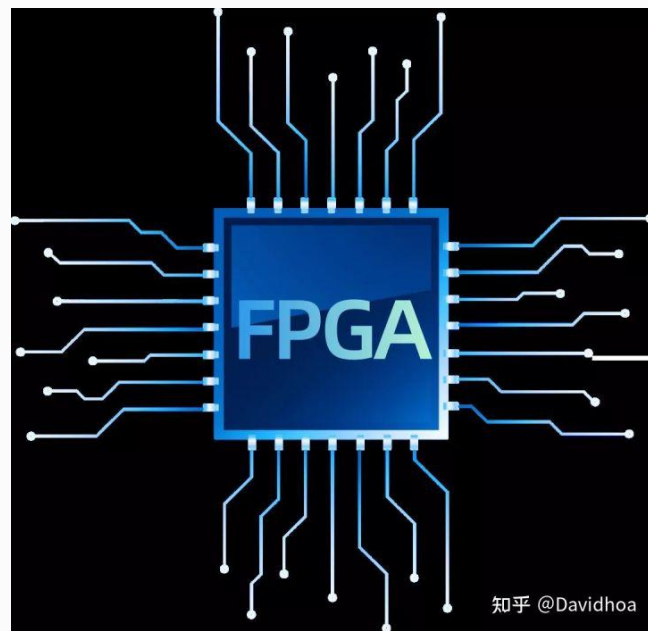


- 2022年2月14日，美国半导体芯片设计巨头AMD宣布以全股份交易方式，完成了对可编程芯片（FPGA）企业赛灵思（Xilinx）的收购。



FPGA发展

- ✓ 5G通信
- ✓ 人工智能
- ✓ 数据中心
- ✓ 航空航天
- ✓ 雷达
- ✓ 导航
- ✓ IC设计





FPGA发展

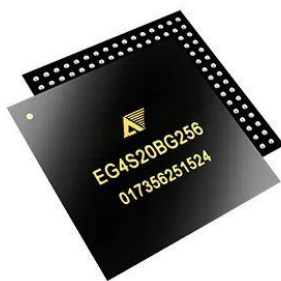
■ 自主知识产权FPGA

✓ 上海复旦微



2018年，28nm工艺制程，亿门级FPGA，SerDes速率13.1Gbps。

✓ 安路科技



✓ 航锦科技

✓ 紫光国微



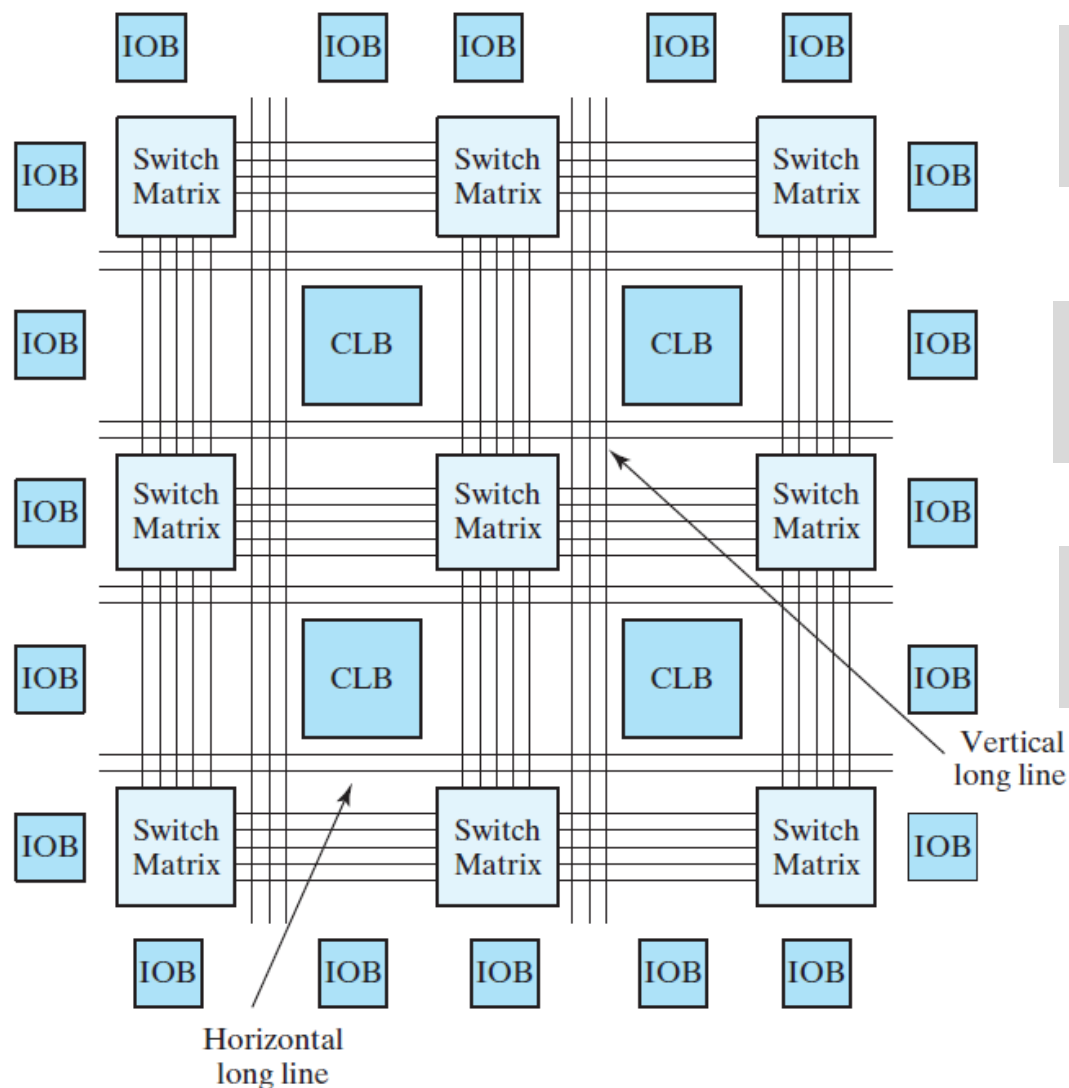
✓ 紫光同创、高云半导体、智多晶、京微齐力



■ 任重道远，每一个为此努力的人都值得尊敬



FPGA的基本结构



➤ **CLB: 可配置逻辑模块**
(Configurable Logic Block)

➤ **IOB: 输入/输出模块**
(Input Output Block)

➤ **ICR: 互连资源**
(Inter-Connection Resource)

**配置数据存放在内部
静态RAM内。**



FPGA的基本结构

■ 可配置逻辑模块 (CLB)

Xilinx Artix-7 系列器件

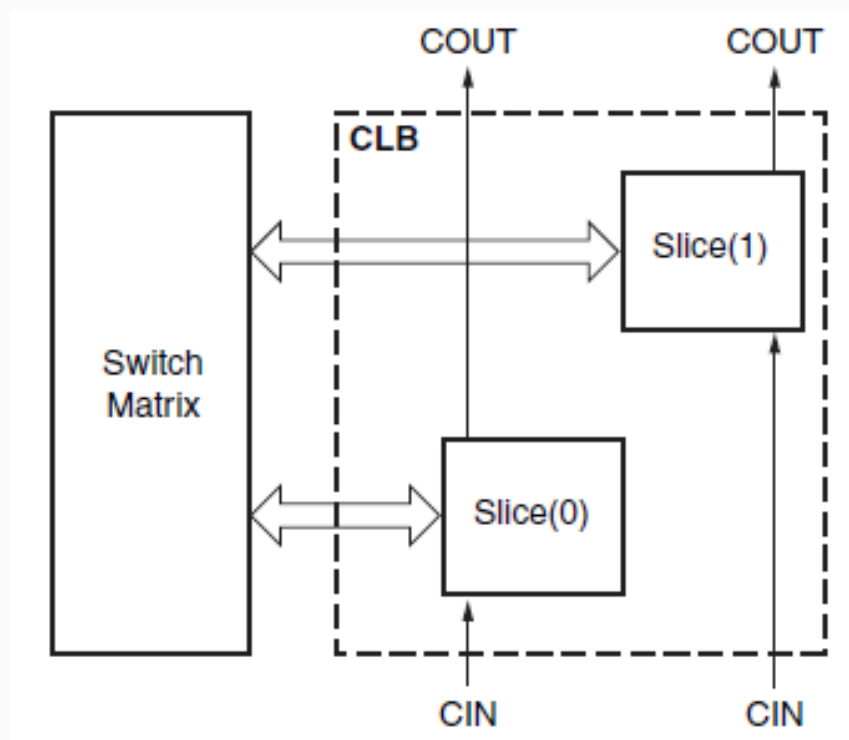
➤ 1个CLB: 2 Slice

➤ 1个Slice包含:

- ✓ 4个6-input LUT
- ✓ 8个触发器
- ✓ 多个数据选择器
- ✓ 算术进位逻辑

➤ Slice分两种: SliceL和SliceM

- ✓ SliceM中的LUT可以配置为分布式RAM或者移位寄存器。
- ✓ 1个CLB包含2个SliceL或者1个SliceL和1个SliceM。





1/4 Slice



触发器

触发器/锁存器

➤ 存储元件

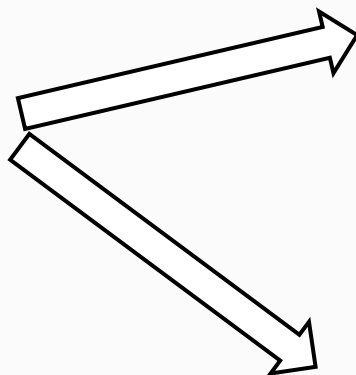


FPGA的基本结构

逻辑函数: $F = AB + AC$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

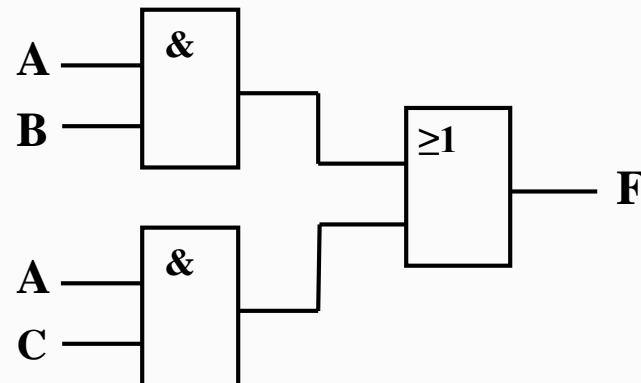
门电路实现



LUT实现
 8×1 RAM

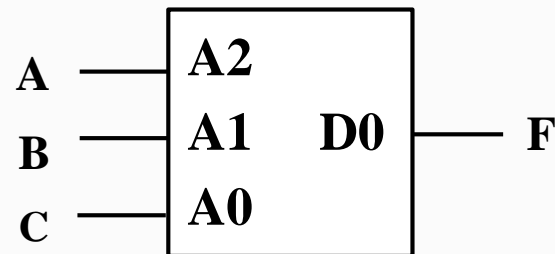
变量输入

逻辑输出



地址输入

数据输出





FPGA的基本结构

Device	Slices ⁽¹⁾	SLICEL	SLICEM	6-input LUTs	Distributed RAM (Kb)	Shift Register (Kb)	Flip-Flops
7A12T	2,000 ⁽²⁾	1,316	684	8,000	171	86	16,000
7A15T	2,600 ⁽²⁾	1,800	800	10,400	200	100	20,800
7A25T	3,650	2,400	1,250	14,600	313	156	29,200
7A35T ✓	5,200 ⁽²⁾	3,600	1,600	20,800	400	200	41,600
7A50T	8,150	5,750	2,400	32,600	600	300	65,200
7A75T	11,800 ⁽²⁾	8,232	3,568	47,200	892	446	94,400
7A100T	15,850	11,100	4,750	63,400	1,188	594	126,800
7A200T	33,650	22,100	11,550	134,600	2,888	1,444	269,200

Notes:

1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only SLICEMs can use their LUTs as distributed RAM or SRLs.
2. Number of slices corresponding to the number of LUTs and flip-flops supported in the device.

➤ **Distributed RAM = 1600 SliceM * 4 * 64bit = 409600bits = 400 Kb**

➤ **Shift Register = 1600 SliceM * 4 * 32bit = 204800bits = 200 Kb**

➤ **Flip-Flops = 5200 Slice * 8个 = 41600个**



FPGA的基本结构

■ Block RAM 和 DSP Slice

Device	DSP48E1 Slices ⁽²⁾	Block RAM Blocks ⁽³⁾		
		18 Kb	36 Kb	Max (Kb)
XC7A12T	40	40	20	720
XC7A15T	45	50	25	900
XC7A25T	80	90	45	1,620
XC7A35T	90	100	50	1,800
XC7A50T	120	150	75	2,700
XC7A75T	180	210	105	3,780
XC7A100T	240	270	135	4,860
XC7A200T	740	730	365	13,140

- Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
- Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.

下面关于FPGA说法正确的是：

A

FPGA是基于与或阵列的；

B

FPGA是基于查找表（LUT）的；

C

FPGA中文是“现场可编程门阵列”；

D

FPGA的密度没有CPLD大；

E

FPGA内部集成有触发器和RAM。

提交



Verilog HDL语言

■ HDL

HDL (Hardware Description Language) 是以文本形式描述数字系统硬件的计算机语言，专门用来描述硬件结构和逻辑电路的行为。

➤ VHDL

➤ Verilog HDL

- ✓ **1995年：批准为一种标准的HDL, IEEE Std 1364-1995**
- ✓ **2001年：修订版, IEEE Std 1364-2001**
- ✓ **2005年：增强版, IEEE Std 1364-2005**



Verilog HDL语言

■ Verilog HDL基本结构—模块

模块是Verilog语言的基本单元，模块可以用下面一种建模方法（或一个组合）来描述。

- 用实例化（instantiation）预定义基本逻辑门（primitive gates）或用户自定义的门电路进行门级建模（gate level modeling）
- 用关键字为assign的连续赋值（continuous level modeling）语句进行数据流建模（dataflow modeling）
- 用关键字为always的过程赋值（procedural assign）语句进行行为建模（behavioral modeling）



Verilog HDL语言

■ 门级建模

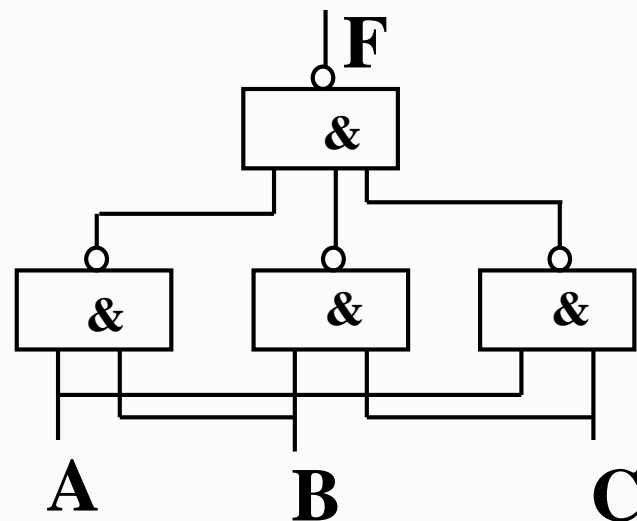
例1：3变量表决器

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= BC + AC + AB$$

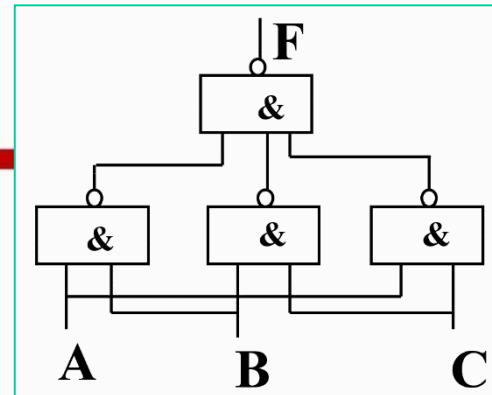
$$F = AB + AC + BC$$

$$= \overline{\overline{AB} \cdot \overline{AC} \cdot \overline{BC}}$$





Verilog HDL语言



门级建模 例1：3变量表决器

➤ 设计源文件

```
module voting3(  
    input a,  
    input b,  
    input c,  
    output y  
);  
    wire y1,y2,y3;  
    nand  
        G1 (y1,a,b),  
        G2 (y2,a,c),  
        G3 (y3,b,c),  
        G4 (y,y1,y2,y3);  
endmodule
```

- ✓ **module:** 关键词，定义了一个模块
- ✓ **voting3:** 模块名
- ✓ **a, b, c:** 模块的输入端口声明
- ✓ **y:** 模块的输出端口声明
- ✓ **wire:** 关键词，声明线网型变量
- ✓ **y1,y2,y3:** 线网型变量，物理连接
- ✓ **nand:** 实例化预定义的基本门（与非门）
- ✓ **G1,G2,G3,G4:** 实例名
- ✓ **Endmodule:** 关键词



Verilog HDL语言

➤ 仿真源文件 (例1: 3变量表决器)

```
module voting3_test( );
```

```
  reg x, y, z;
```

```
  wire F;
```

```
  voting3 t0(
```

```
    .a(x),
```

```
    .b(y),
```

```
    .c(z),
```

```
    .f(F)
```

```
  );
```

```
  initial
```

```
  begin
```

```
    x = 0; y = 0; z = 0;
```

```
    repeat(8)
```

```
      #10 {x, y, z} = {x, y, z} + 1;
```

```
  end
```

```
endmodule
```

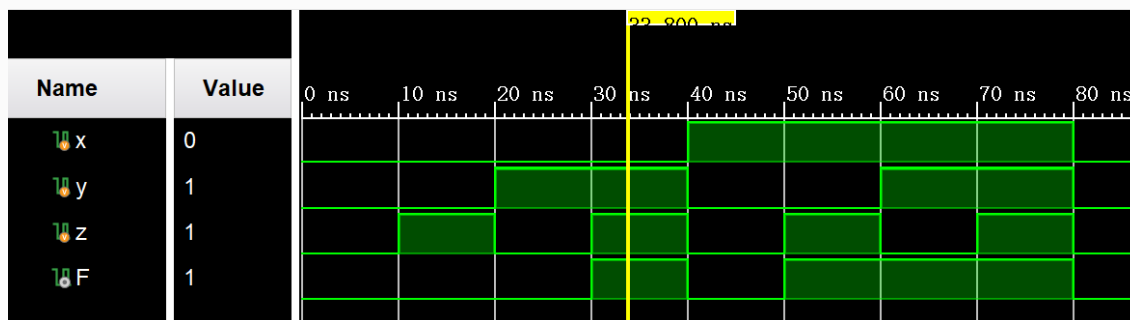
✓ 仿真文件模块名 (没有端口)

✓ 输入/输出变量定义

✓ 待测试模块例化 (调用)

信号名关联方式

✓ 产生激励





Verilog HDL语言

■ 数据流建模 例1：3变量表决器

使用运算符和assign关键词

Some Verilog HDL Operators

Symbol	Operation	Symbol	Operation
+	binary addition		
-	binary subtraction		
&	bitwise AND	&&	logical AND
	bitwise OR		logical OR
^	bitwise XOR		
~	bitwise NOT	!	logical NOT
= =	equality		
>	greater than		
<	less than		
{ }	concatenation		
?:	conditional		

assign: 关键词，连续赋值语句

```
module voting3(
```

```
    input a,
```

```
    input b,
```

```
    input c,
```

```
    output y
```

```
);
```

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

```
endmodule
```



Verilog HDL语言

■ 行为建模

- 行为描述采用always关键词构成一个always块
- 在always块内包含可选的事件控制表达式（也叫做敏感表）以及过程赋值语句

always @ (事件控制表达式/敏感表)

begin

过程赋值语句

end



Verilog HDL语言

■ 行为建模

例1：3变量表决器

设计源文件

```
module voting3(  
    input a,  
    input b,  
    input c,  
    output reg y  
);  
  
    always @ *  
    begin  
        case ({a, b, c})  
            3'b000 : y = 0;  
            3'b001 : y = 0;  
            3'b010 : y = 0;  
            3'b011 : y = 1;  
            3'b100 : y = 0;  
            3'b101 : y = 1;  
            3'b110 : y = 1;  
            3'b111 : y = 1;  
        endcase  
    end  
endmodule
```



Verilog HDL语言

■ 数据类型

- 线网型变量 wire
- 寄存器型变量 reg
- 存储器型变量 reg
- 符号常量 parameter



Verilog HDL语言

■ 运算符

➤ 算术运算符

算术运算符	说明
+	加
-	减
*	乘
/	除
%	取模

➤ 逻辑运算符

逻辑运算符	说明
&&（双目）	逻辑与
（双目）	逻辑或
！（单目）	逻辑非



Verilog HDL语言

➤ 按位运算符

按位运算符	说明
~	按位取反
&	按位与
	按位或
^	按位异或
^~, ~^	按位同或

➤ 关系运算符

关系运算符	说明
<	小于
<=	小于或等于
>	大于
>=	大于或等于



Verilog HDL语言

➤ 等式运算符

等式运算符	说明
<code>==</code>	等于
<code>!=</code>	不等于
<code>===</code>	全等
<code>!==</code>	不全等

➤ 缩减运算符

缩减运算符	说明
<code>&</code>	与
<code>~ &</code>	与非
<code> </code>	或
<code>~ </code>	或非
<code>^</code>	异或
<code>^~, ~^</code>	同或

➤ 移位运算符

移位运算符	说明
<code>>></code>	右移
<code><<</code>	左移

$4'b0011 + 4'b0011 = 4'[\text{填空1}] ;$

$4'b0010 \& 4'b0011 = 4'[\text{填空2}] ;$

$4'b0010 | 4'b0110 = 4'[\text{填空3}] ;$

$\sim 4'b0011 = 4'[\text{填空4}] ;$

$4'b0110 >> 1 = 4'[\text{填空5}] ;$

$4'b0110 << 1 = 4'[\text{填空6}] ;$

正常使用填空题需3.0以上版本雨课堂

作答



Verilog HDL语言

➤ 条件运算符和拼接运算符

1. 条件运算符: ?

实现的是组合电路

例如: `assign f = sel ? d1 : d0`

当 `sel = 1` 时, `f = d1`;

当 `sel = 0` 时, `f = d0`。

2. 拼接运算符: { }

将两个或者两个以上个信号的某些位拼接起来。

例如: `{a, b[3:0], 3'b101}`



Verilog HDL语言

■ 语句

➤ 赋值语句

1. 连续赋值语句 assign

用于对wire型变量赋值，是描述组合逻辑最常用的方法之一。例如， **assign f = a&b;**

2. 过程赋值语句 “=” 和 “<=”

用于对reg型变量赋值，在过程块中使用。有非阻塞和阻塞两种方式。

- ✓ 阻塞赋值方式：赋值符号为 =，例如， **b=a;**
- ✓ 非阻塞赋值方式：赋值符号为 <=，例如， **b<=a。**



Verilog HDL语言

➤ 过程说明语句 `always`

`always` 块包含一个或一个以上的语句，在`always`块中被赋值的只能是寄存器`reg`型变量。

`always @ (敏感信号表达式)`

例如，

`always @ (A, B, C)` //电平敏感事件，A、B或者C变化将启动`always`块的过程语句。

`always @ (posedge clk)` //clk上升沿触发

`always @ (negedge clk)` //clk下降沿触发

`always @ (posedge clk1, negedge clk2)`

`always @ (*)` //所在模块的任何输入信号变化都触发

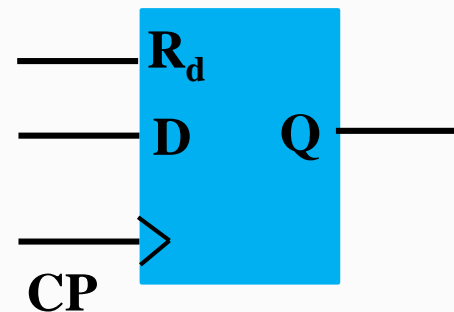


Verilog HDL语言

例：D 触发器

设计源文件

```
module FF_D(  
    input clk,  
    input reset,  
    input d,  
    output reg q  
);  
    always @ (posedge clk, posedge reset)  
        if (reset)  
            q <= 0;  
        else  
            q <= d;  
endmodule
```



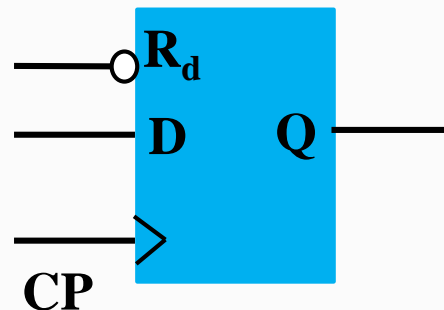
异步、高电平复位！



Verilog HDL语言

异步、低电平复位!

```
always @ (posedge clk, negedge reset)
    if (~reset)
        q <= 0;
    else
        q <= d;
endmodule
```



同步、低电平复位!

```
always @ (posedge clk)
    if (~reset)
        q <= 0;
    else
        q <= d;
endmodule
```

同步、高电平复位!

```
always @ (posedge clk)
    if (reset)
        q <= 0;
    else
        q <= d;
endmodule
```



Verilog HDL语言

例：D 触发器

仿真源文件

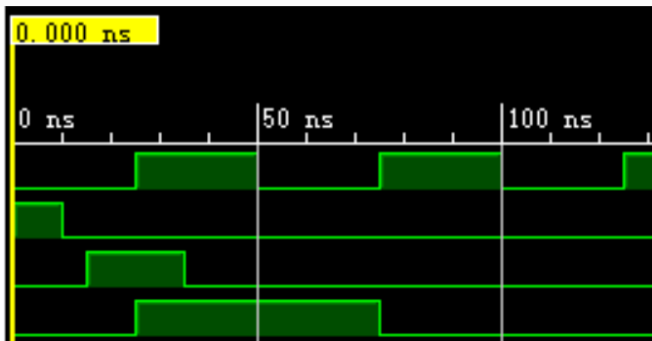
```
module test_DFF( );  
    reg cp;  
    reg clr;  
    reg d;  
    wire q;  
    FF_D u0 (  
        .clk(cp),  
        .reset(clr),  
        .d(d),  
        .q(q)  
    );  
    parameter PERIOD = 50;
```

```
initial  
begin  
    clr = 1;  
    cp = 0;  
    d = 0  
    #10 clr = 0;  
    #5 d = 1;  
    #20 d = 0;  
end
```

```
initial  
begin  
    repeat (8);  
        #(PERIOD/2) cp = ~cp;  
    end  
endmodule
```

仿真结果

Name	Value
cp	0
clr	1
d	0
q	0



```
always  
begin  
    #(PERIOD/2) cp = ~cp;  
end
```

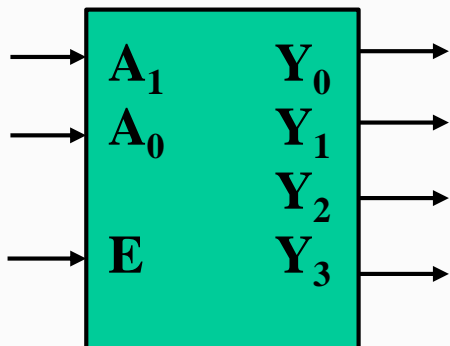


Verilog HDL语言

➤ 条件语句

1. if-else 语句

2-4译码器



```
module Decoder_2_4_if (  
    input [1:0] a,  
    input en,  
    output reg [3:0] y  
);  
    always @ *  
        if (en==1'b0)  
            y = 4'b0000;  
        else if (a == 2'b00)  
            y = 4'b0001;  
        else if (a == 2'b01)  
            y = 4'b0010;  
        else if (a == 2'b10)  
            y = 4'b0100;  
        else  
            y = 4'b1000;  
endmodule
```



Verilog HDL语言

2. case 语句

2-4 译码器

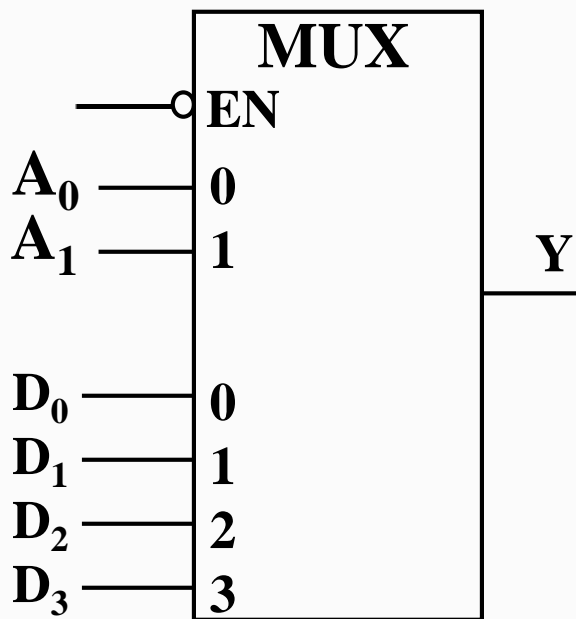
```
module Decoder_2_4_case (  
    input [1:0] a,  
    input en,  
    output reg [3:0] y  
);  
    always @ *  
    case ({en, a})  
        3'b000, 3'b001, 3'b010, 3'b011: y = 4'b0000;  
        3'b100: y = 4'b0001;  
        3'b101: y = 4'b0010;  
        3'b110: y = 4'b0100;  
        3'b111: y = 4'b1000;  
    endcase  
endmodule
```




Verilog HDL语言

例：“四选一”数据选择器

设计源文件



```
module mux_4x1_beh(
    input [0:3] D,
    input [1:0] A,
    input _EN,
    output reg Y
);
always @ (D, A, _EN)
begin
    if ( _EN )
        Y = 0;
    else
        case (A)
            2'b00: Y = D[0];
            2'b01: Y = D[1];
            2'b10: Y = D[2];
            2'b11: Y = D[3];
        endcase
    end
endmodule
```



Verilog HDL语言

例：“四选一”数据选择器

```
module sim_mux_4x1 ();  
    reg [0:3] D;  
    reg [1:0] A;  
    reg _EN;  
    wire Y;  
    mux_4x1_beh u0(  
        D, A, _EN, Y  
    );  
    initial  
    begin  
        A = 2'b00; D = 4'b1000; _EN=0;  
        #50  
        A = 2'b01; D = 4'b1000; _EN=0;  
        #50  
        A = 2'b10; D = 4'b0010; _EN=0;  
        #50  
        A = 2'b11; D = 4'b1001; _EN=0;  
        #50  
        _EN = 1;  
    end  
endmodule
```

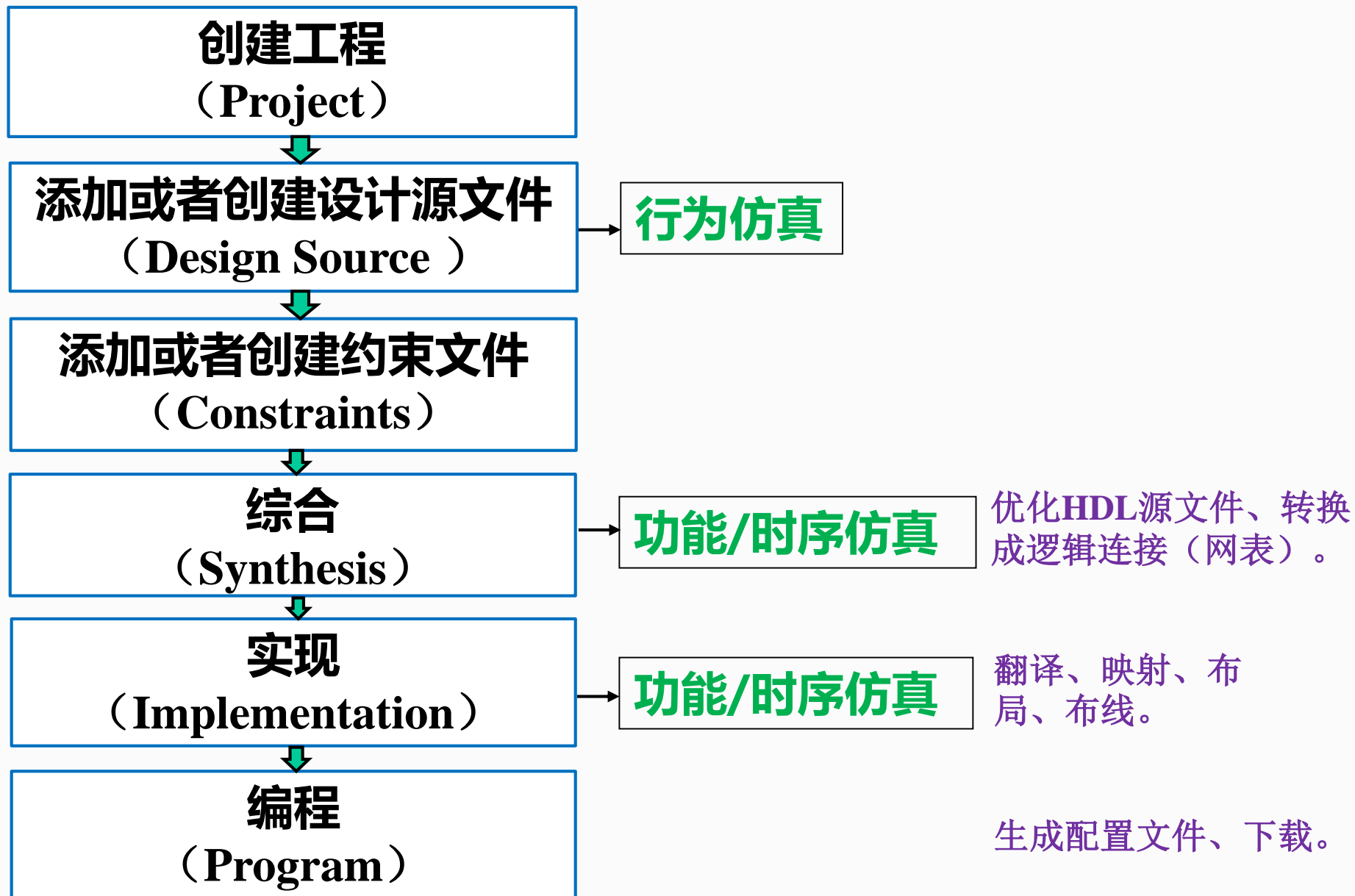
仿真源文件

仿真结果





基于Vivado的FPGA开发流程





基于Vivado的FPGA开发流程

例子：三变量表决器电路设计

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\begin{aligned} F &= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC \\ &= BC + AC + AB \end{aligned}$$

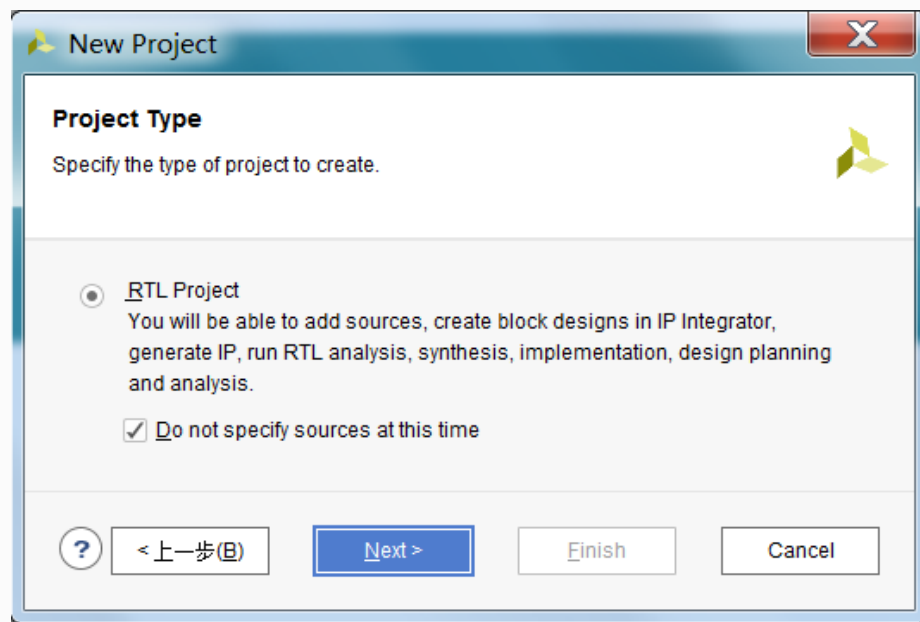
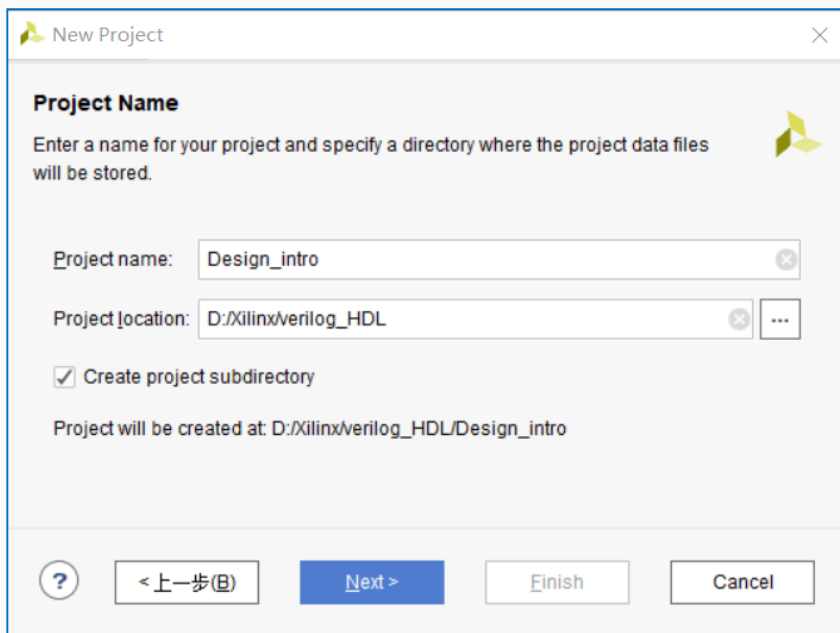


基于Vivado的FPGA开发流程

例子：三变量表决器电路设计

1. 创建工程

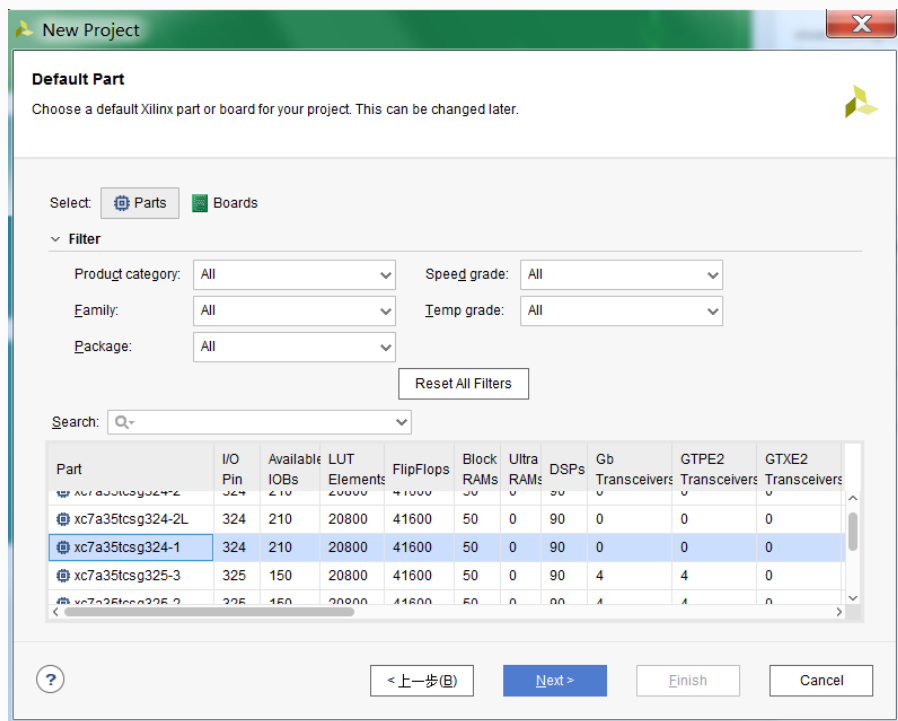
(1) 打开Vivado，单击“Create Project”，输入工程名称。





基于Vivado的FPGA开发流程

(2) 选择器件为Artix-7 系列的FPGA: xc7a35tcsg324-1



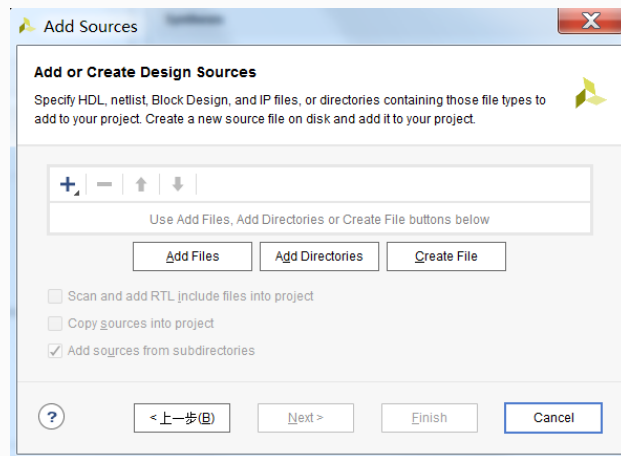
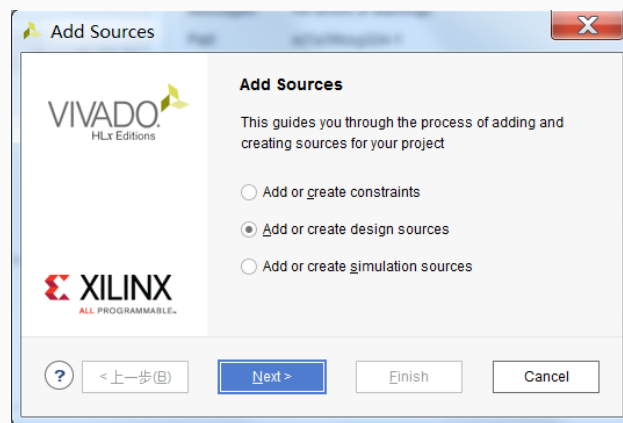
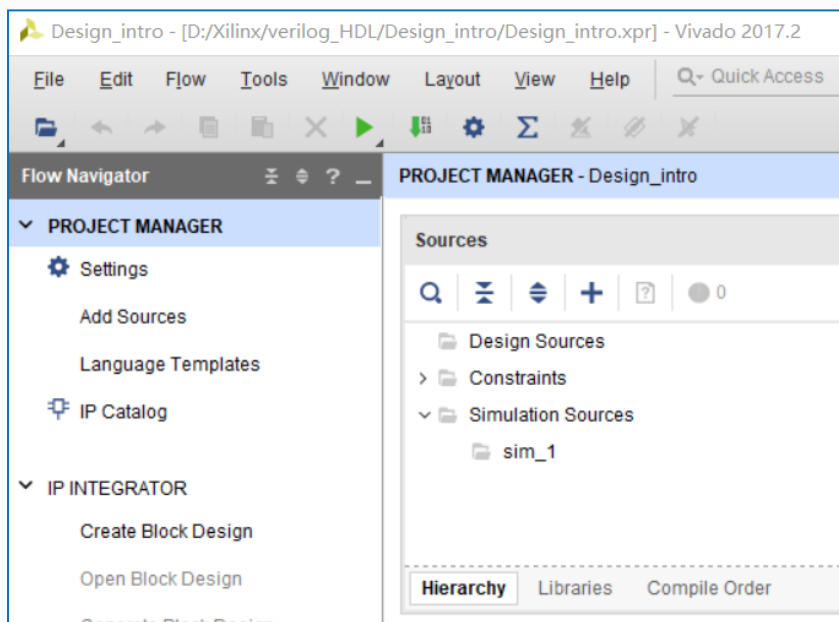
(3) 单击 “Finish” 完成工程创建



基于Vivado的FPGA开发流程

2. 创建 Verilog HDL 源文件

(1) 在左侧Flow Navigator窗口的PROJECT MANAGER菜单下点击“Add Sources”，选中“Add or create design sources”，单击“Next”进入下一步，单击“Create File”，输入文件名，点击“Finish”。





基于Vivado的FPGA开发流程

(2) 弹出如下窗口，输入模块名。下面是端口定义，建议这里不定义。点击“OK”。

Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name:

I/O Port Definitions

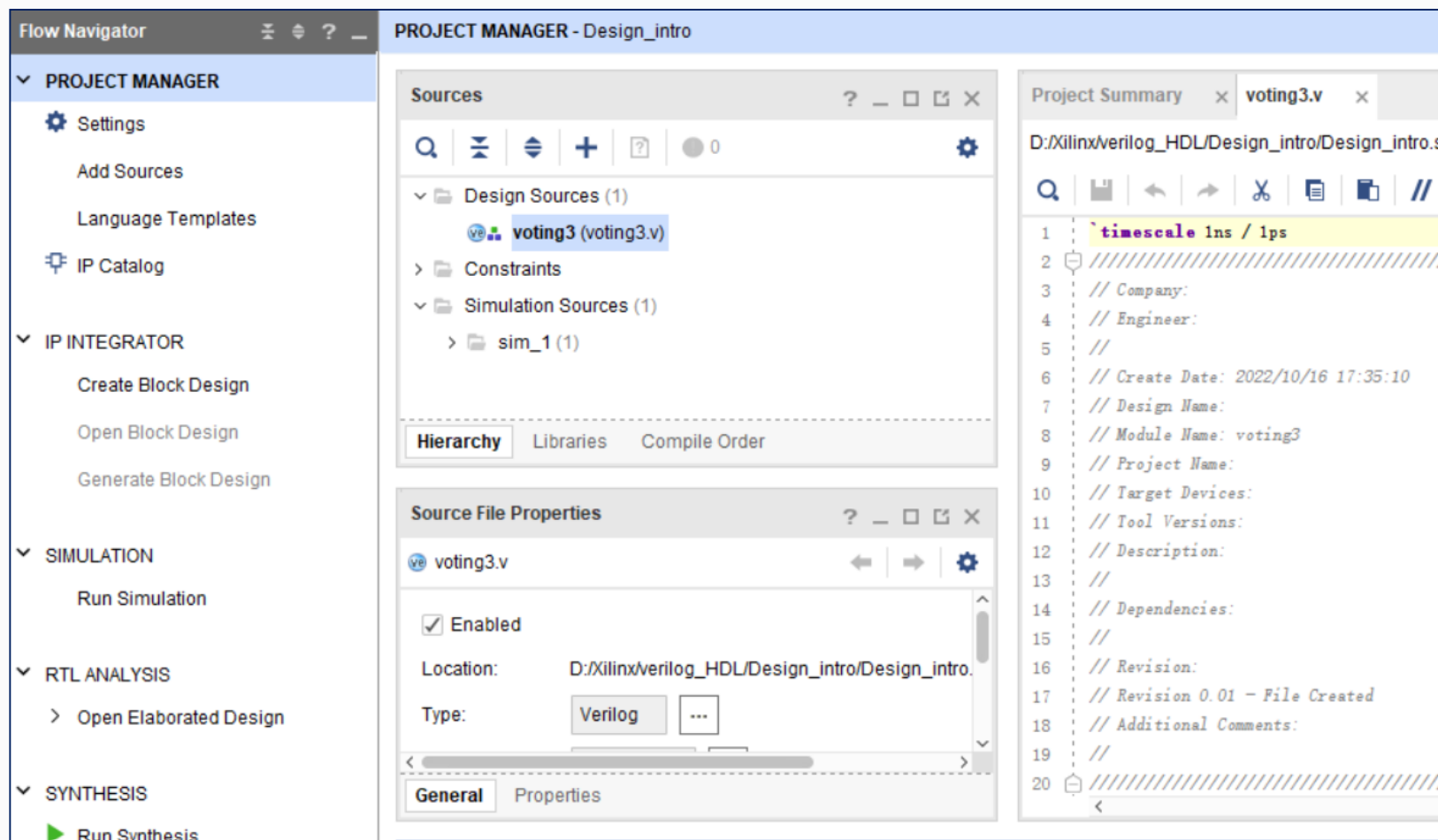
Port Name	Direction	Bus	MSB	LSB
	input	<input type="checkbox"/>	0	0

Buttons: ? OK Cancel



基于Vivado的FPGA开发流程

(3) 在中部Sources窗口的Design Sources下单击文件voting3，在右边窗口即可编辑voting3.v文件。





基于Vivado的FPGA开发流程

```
module voting3(  
    input a,  
    input b,  
    input c,  
    output f  
);  
    assign f = a&b | a&c | b&c; //f = ab + ac + bc;  
endmodule
```

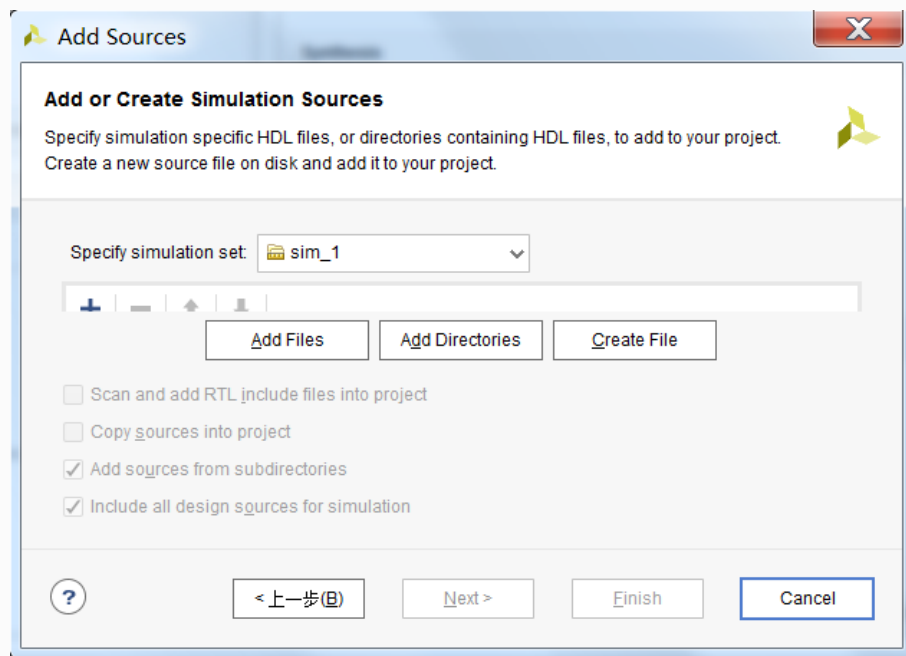
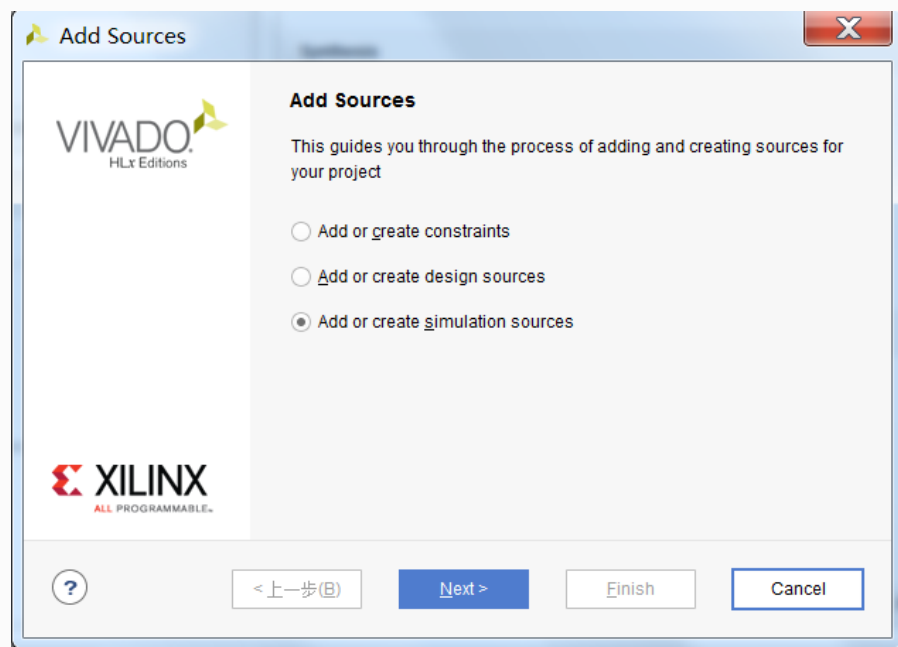
(4) 点击顶部保存按钮。



基于Vivado的FPGA开发流程

3. 仿真

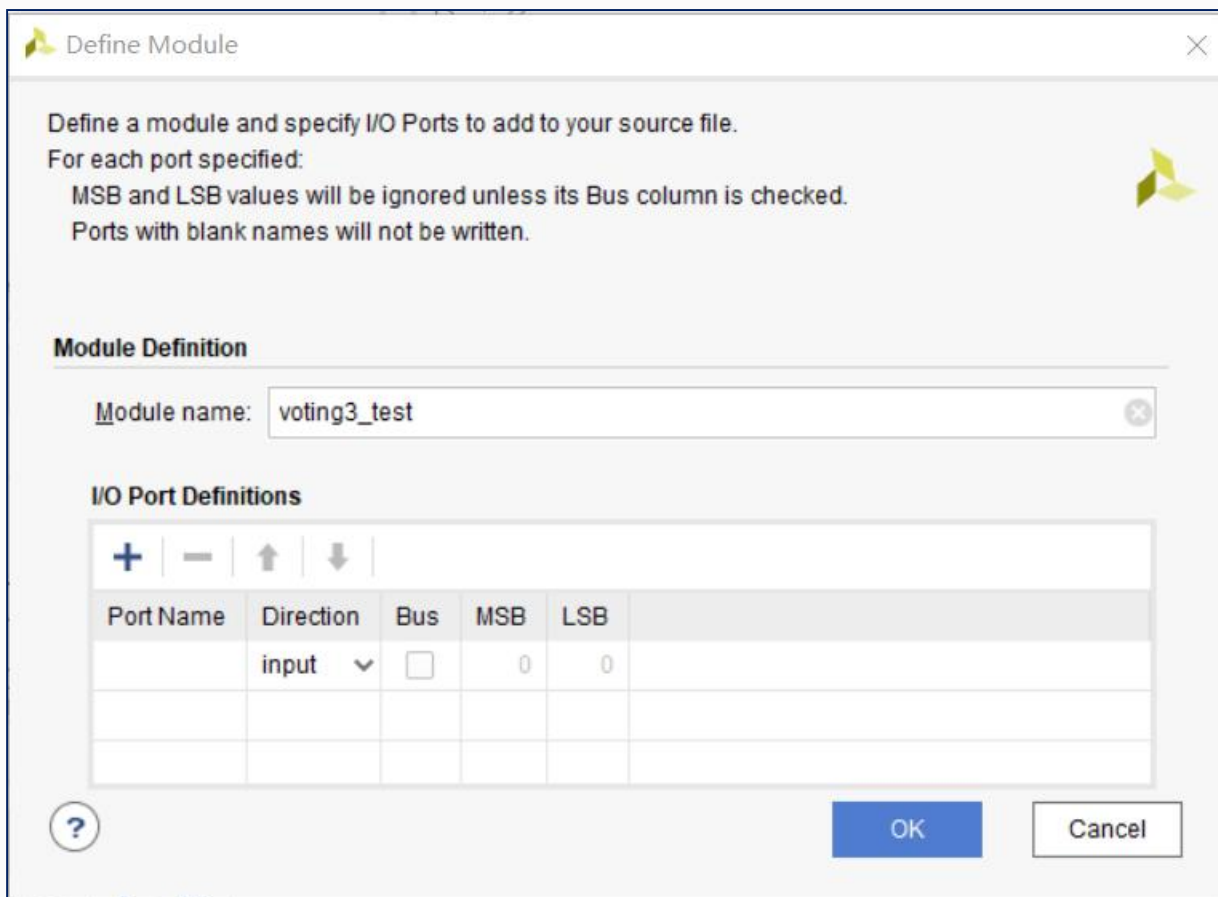
(1) 在左侧Flow Navigator窗口的PROJECT MANAGER菜单下点击“Add Sources”，选中“Add or create simulation sources”，单击“Next”进入下一步，单击“Create File”，输入文件名（voting3_test），点击“OK”，再在弹出窗上点击“Finish”。





基于Vivado的FPGA开发流程

(2) 弹出如下窗口，输入模块名，点击“OK”。



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name:

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	<input type="checkbox"/>	0	0

Buttons: ? OK Cancel



基于Vivado的FPGA开发流程

(3) 在中部Sources窗口的Simulation Sources下单击文件voting3_test，在右边窗口即可编辑voting3_test.v文件。

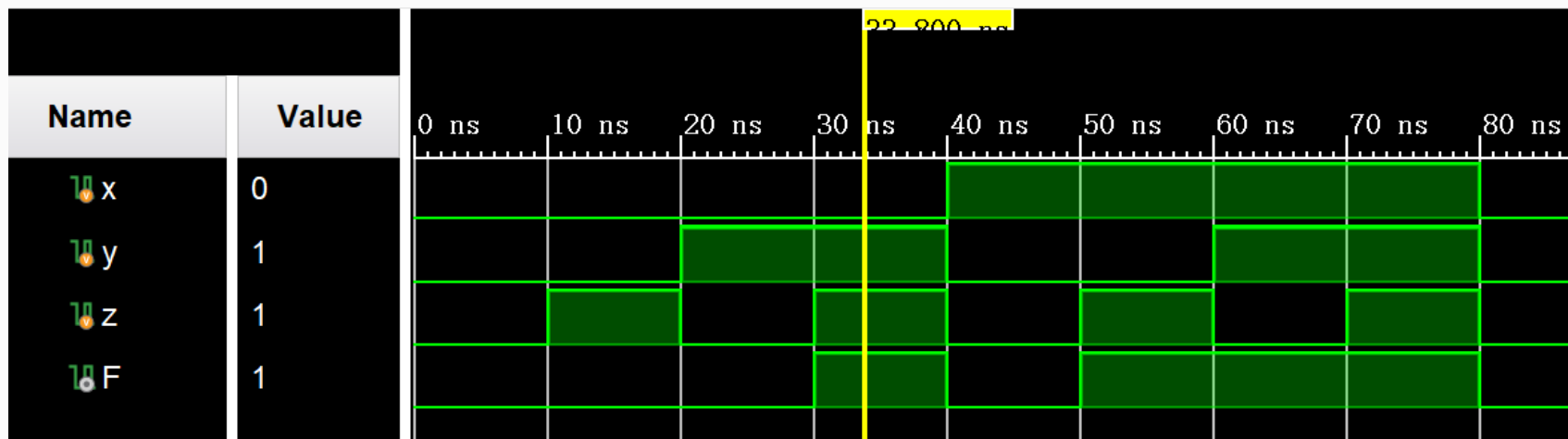
(4) 仿真文件编辑完成后，点击顶部保存按钮。

```
module voting3_test();  
    reg x, y, z;  
    wire F;  
  
    voting3 t0(  
        .a(x),  
        .b(y),  
        .c(z),  
        .f(F)  
    );  
  
    initial  
    begin  
        x = 0; y = 0; z = 0;  
        repeat(8)  
            #10 {x, y, z} = {x, y, z} + 1;  
        end  
    endmodule
```



基于Vivado的FPGA开发流程

(5) 在左侧Flow Navigator窗口的SIMULATION菜单下点击“Run Simulation”，再单击“Run Behavioral Simulation”，稍等片刻后将出现仿真结果。

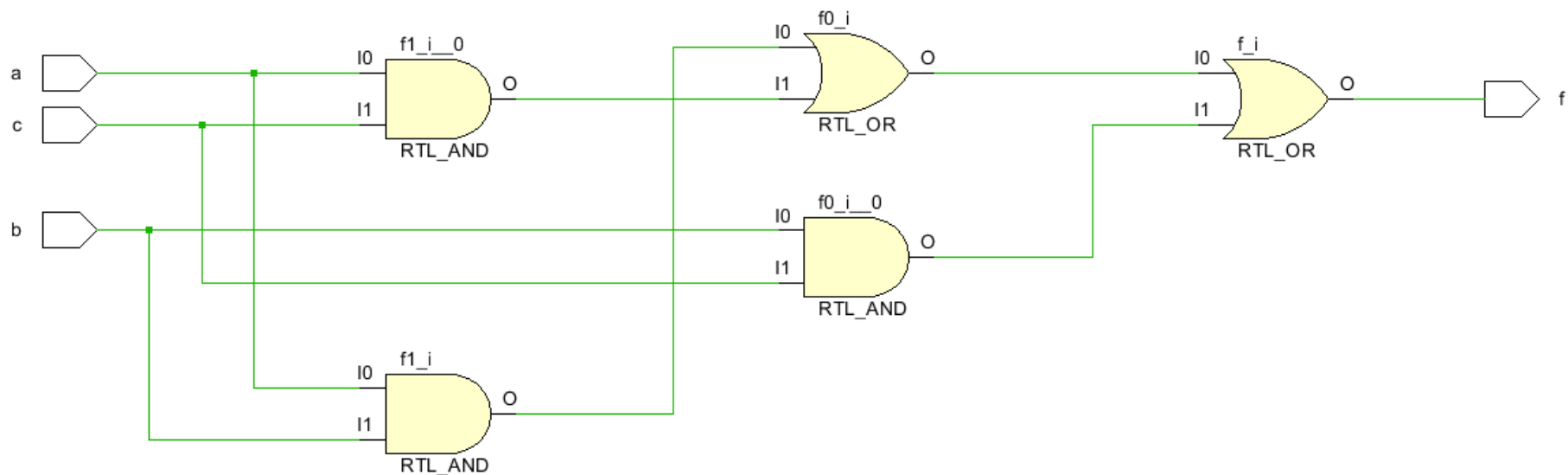




基于Vivado的FPGA开发流程

4. RTL分析

在左侧Flow Navigator窗口的RTL ANALYSIS菜单下单击“Open Elaborated Design”。这一步不是必须的。

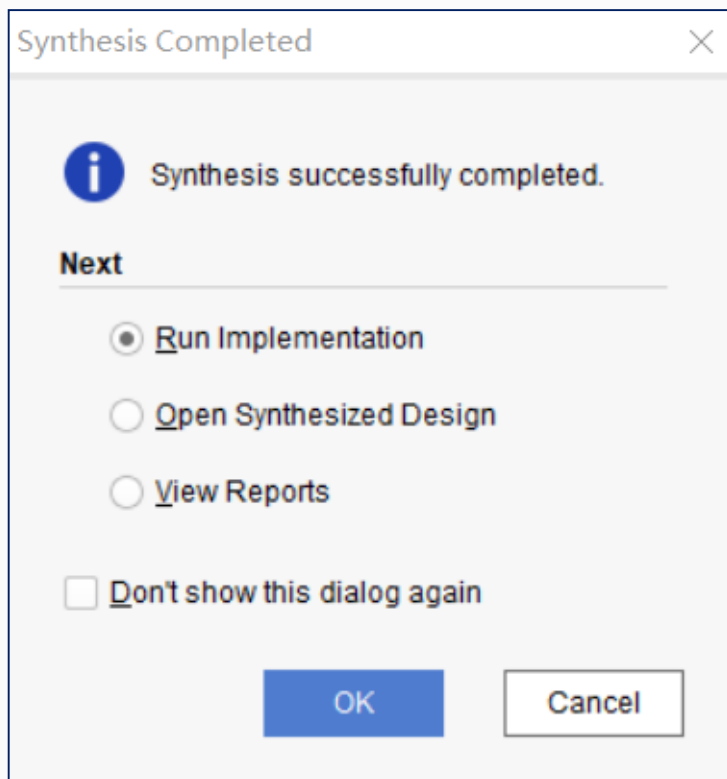




基于Vivado的FPGA开发流程

5. 综合

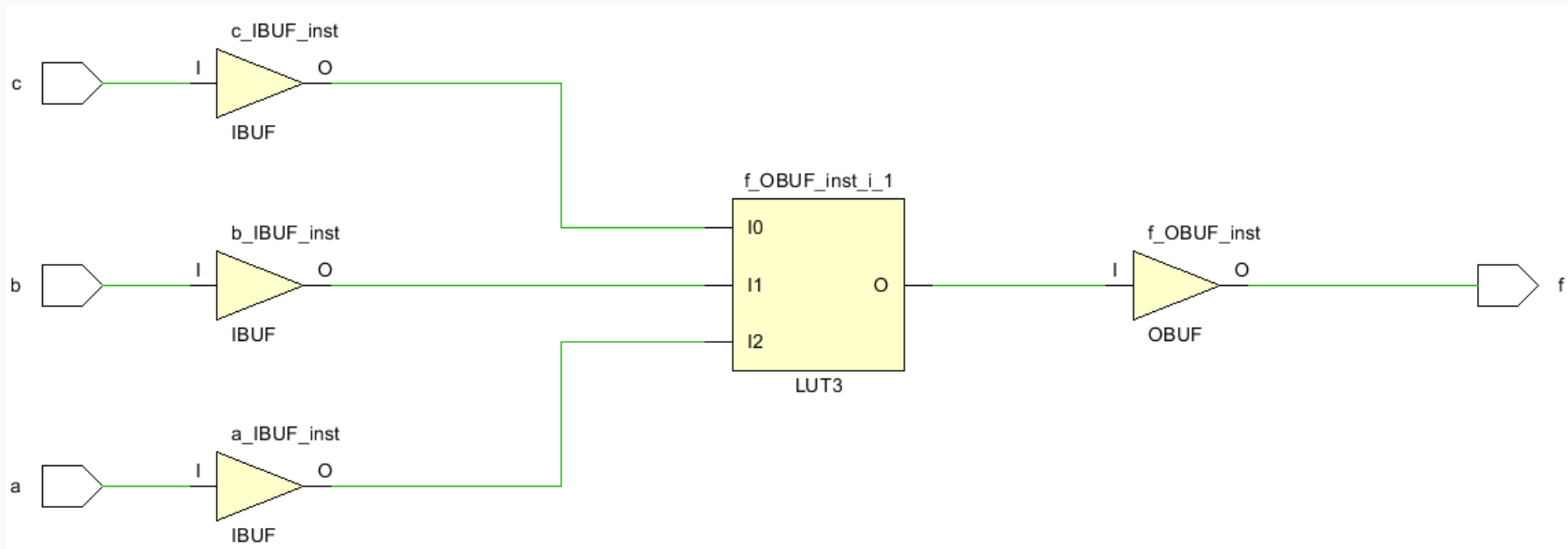
(1) 在左侧Flow Navigator窗口的SYNTHESIS菜单下单击“Run Synthesis”。在弹出窗上点击“OK”后，右上角指示正在进行综合，等待。





基于Vivado的FPGA开发流程

(2) 在左侧Flow Navigator窗口的SYNTHESIS菜单下单击“Open Synthesized Design”，再选“Schematic”。





基于Vivado的FPGA开发流程

6. 实现

(1) 在左侧Flow Navigator窗口的IMPLEMENTATION菜单下单击“run implementation”，单击“Yes”，单击“OK”。右上角指示正在进行实现，等待。

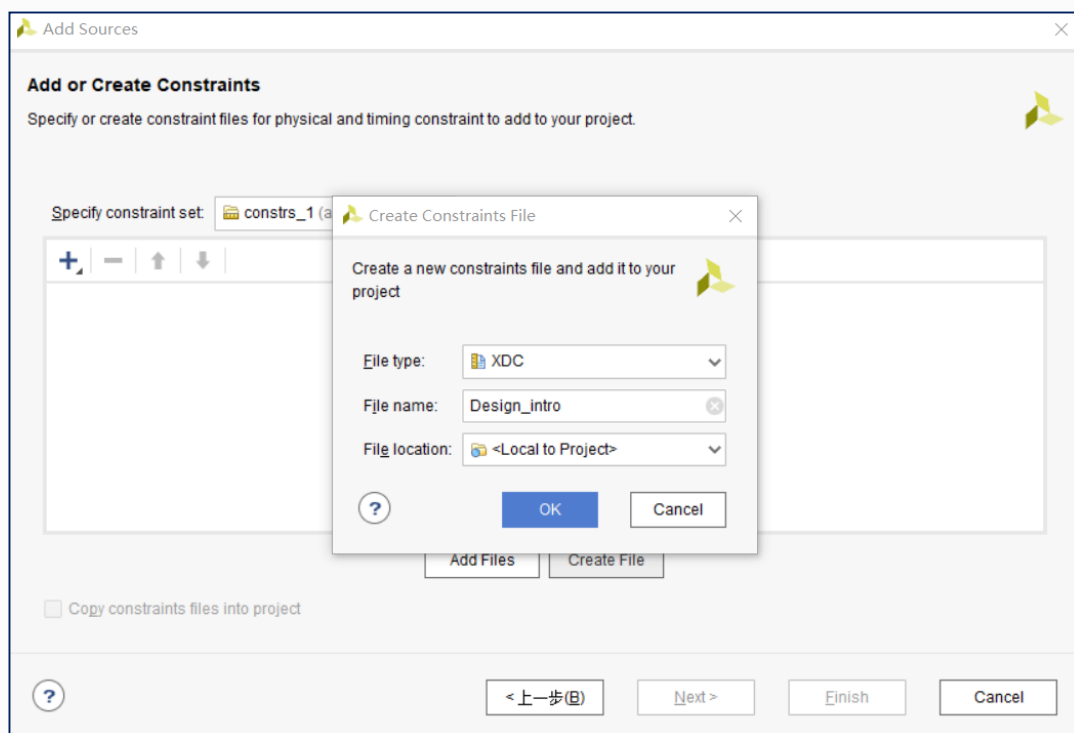


基于Vivado的FPGA开发流程

7. 约束

(1) 在左侧Flow Navigator窗口的PROJECT MANAGER菜单下点击“Add Sources”，选中“Add or create constraints”，单击“Next”进入下一步，单击“Create File”，输入文件名“Design_intro”，单击“Finish”。

(2) 弹出如下窗口，单击“OK”。





基于Vivado的FPGA开发流程

(3) 在约束文件Design_intro.xdc中输入FPGA管脚分配

#Switches

```
set_property -dict {PACKAGE_PIN R1 IOSTANDARD LVCMOS33} [get_ports a]  
set_property -dict {PACKAGE_PIN N4 IOSTANDARD LVCMOS33} [get_ports b]  
set_property -dict {PACKAGE_PIN M4 IOSTANDARD LVCMOS33} [get_ports c]
```

#LED

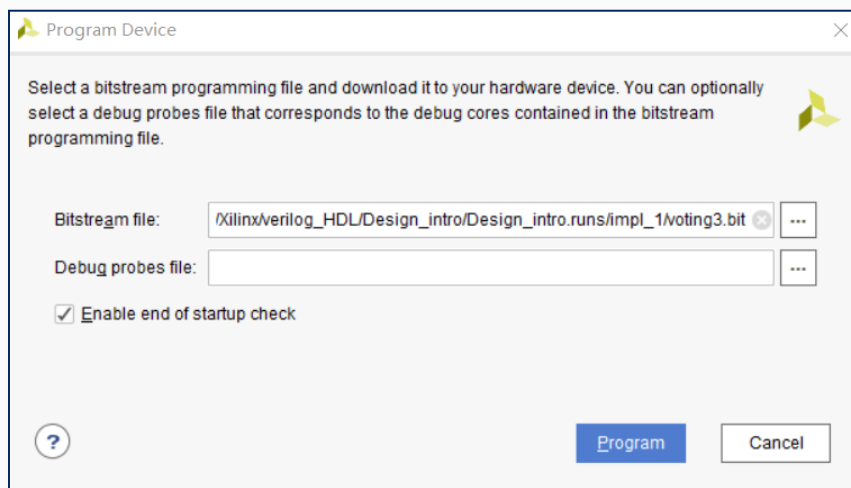
```
set_property -dict {PACKAGE_PIN K2 IOSTANDARD LVCMOS33} [get_ports f]
```



基于Vivado的FPGA开发流程

8. 编程与调试

- (1) 在左侧Flow Navigator窗口的PROGRAM AND DEBUG菜单下单击“Generate Bitstream”，单击“OK”。右上角指示正在进行产生比特流，等待。在成功完成后的弹出窗口，单击“Cancel”。
- (2) 用连接线连接电脑USB接口和电路板JTAG接口。
- (3) 单击“Open Hardware Manager”、“Open Target”、“Auto Connect”。
- (4) 单击“Program Device”、“xc7a35t_0”、“Program”。





基于Vivado的FPGA开发流程

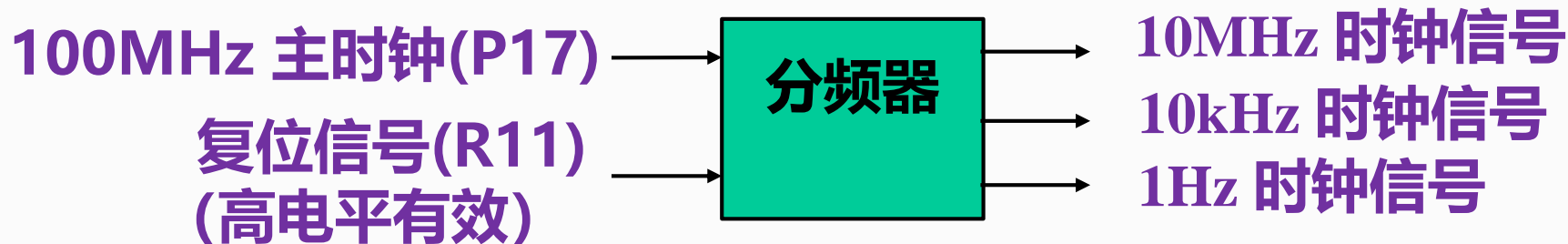
也可以用case语句来描述
三变量表决器。

```
module voting3(  
    input a,  
    input b,  
    input c,  
    output reg f  
);  
always @ *  
    case ({a,b,c})  
        3'b000: f = 0;  
        3'b001: f = 0;  
        3'b010: f = 0;  
        3'b011: f = 1;  
        3'b100: f = 0;  
        3'b101: f = 1;  
        3'b110: f = 1;  
        3'b111: f = 1;  
    endcase  
endmodule
```



基于Vivado的FPGA开发流程

■ 练习题



(1) 首先实现较小分频比的分频器, $D_1=10$, 进行仿真;

(2) 实现分频系数 $D=100\text{MHz}/10\text{kHz}=10000$;

(3) 输出至开发板上 (建议用LD2-0 (K2)), 用示波器观察10kHz时钟信号。

(4) 为了便于观察, 对10kHz时钟信号再分频10000 倍, 输出一个1Hz的信号 (也可以直接对100MHz主时钟分频 10^8) , 驱动LED (建议用LD2-0 (K2)) 。



基于Vivado的FPGA开发流程

■ 程序架构

工程名: ES_design

ES_design_top.v

freq_devision.v

LED_display.v

DDS.v

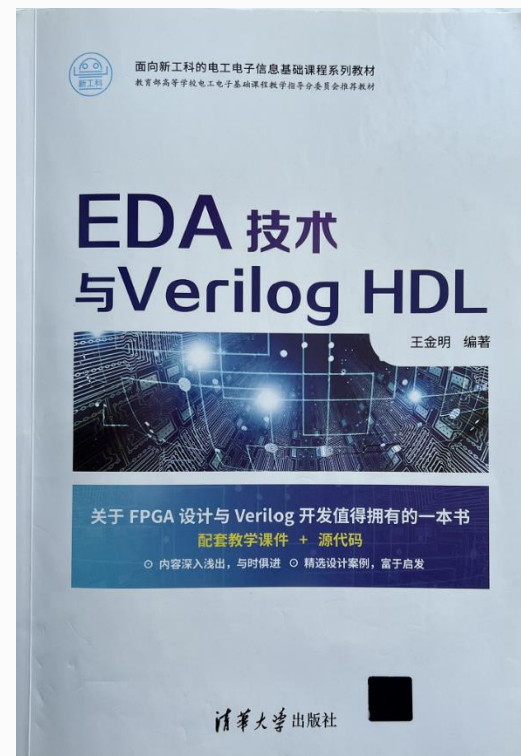
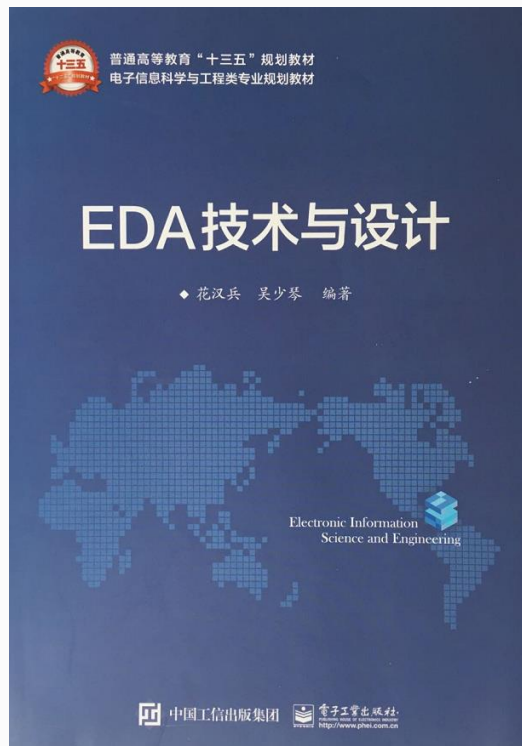
.....



基于Vivado的FPGA开发流程

■ 参考资料

➤ 实验教材



➤ MOOC

<https://www.icourse163.org/course/NJUST-1001753091>

补充单元：Verilog HDL和FPGA设计入门