第四讲 Verilog和FPGA

佟冬
tongdong@pku.edu.cn

微处理器研究开发中心（MPRC）
计算机科学技术系

北京大学

课程回顾：布尔函数

- 将一个开关函数 *f* 对于其变量每种可能取值的结果用表的形式表示。
  - 1对应逻辑“真”；0对应逻辑“假”
- 三个基本函数：与(AND)、或(OR)、非(NOT)的真值表

$a\ b$	$f(a,b)=a+b$
0 0	0
0 1	1
1 0	1
1 1	1

OR

$a\ b$	$f(a,b)=ab$
0 0	0
0 1	0
1 0	0
1 1	1

AND

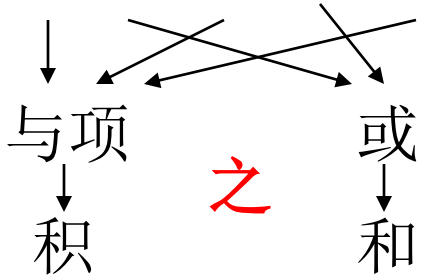
a	$f(a)=\bar{a}$
0	1
1	0

NOT

课程回顾：布尔函数范式

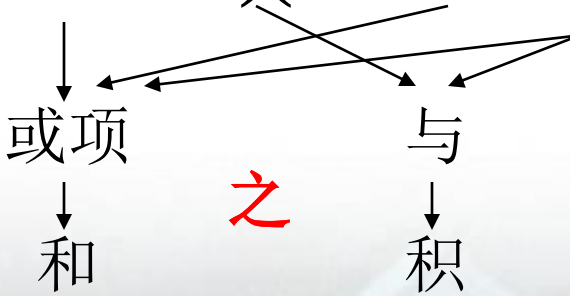
- 积之和(Sum of product, SOP)

$$f(A,B,C)=AB+A'C+AC'$$



- 和之积(Product of sum, POS）

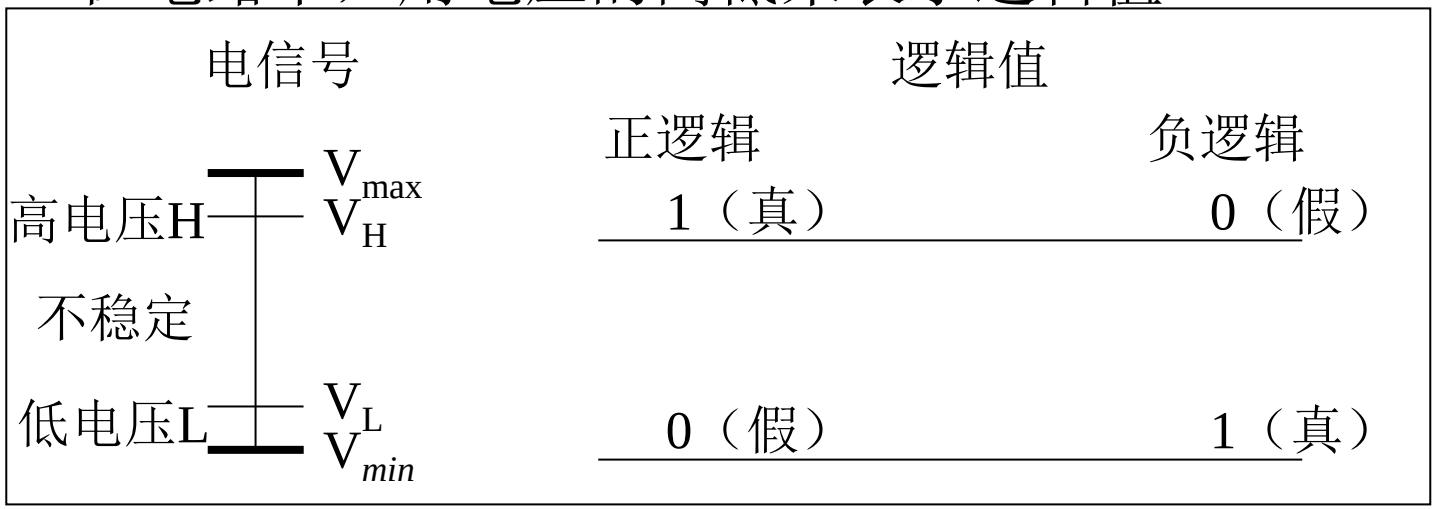
$$f(A,B,C)=(A'+B+C)(B'+C+D')(A+C'+D)$$



课程回顾：二进制在电路中的对应

电信号和逻辑值

– 在电路中，用电压的高低来表示逻辑值



– 一个信号被置为逻辑1称为**有效的**或者**真**。

– 一个信号被清为逻辑0称为**无效的**或者**假**

– 高有效信号（正逻辑）和低有效信号（负逻辑）

– 信号的极性(Polarity)表示信号是高有效或是低有效

课程回顾：基本逻辑门

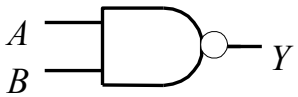
与非门

a	b	$f_{NAND}(a, b) = ab$
0	0	1
0	1	1
1	0	1
1	1	0

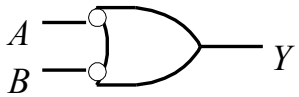
(a)

A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

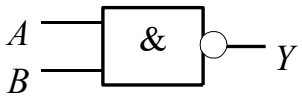
(b)



(c)



(d)

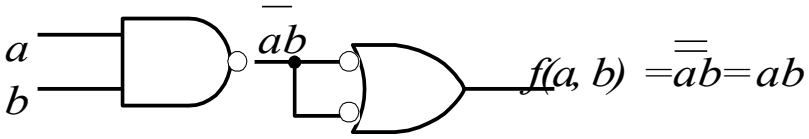


(e)

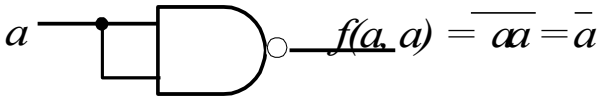
(a) 与非门的逻辑功能
(b) 与非门的电子功能
(c) 标准符号表示
(d) 替代符号（负逻辑）
(e)IEEE 块符号表示

课程回顾：与非门的特性

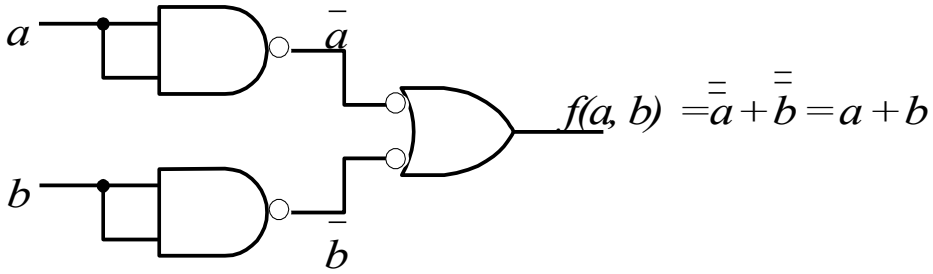
- AND, OR, NOT门可以仅用NAND门来构造



AND gate



NOT gate



OR gate

课程回顾：与非门的开关模型

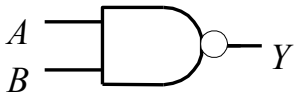
- 与非门(NAND)

a	b	$f_{NAND}(a, b) = ab$
0	0	1
0	1	1
1	0	1
1	1	0

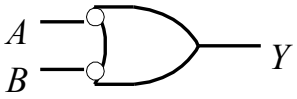
(a)

A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

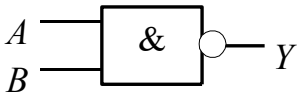
(b)



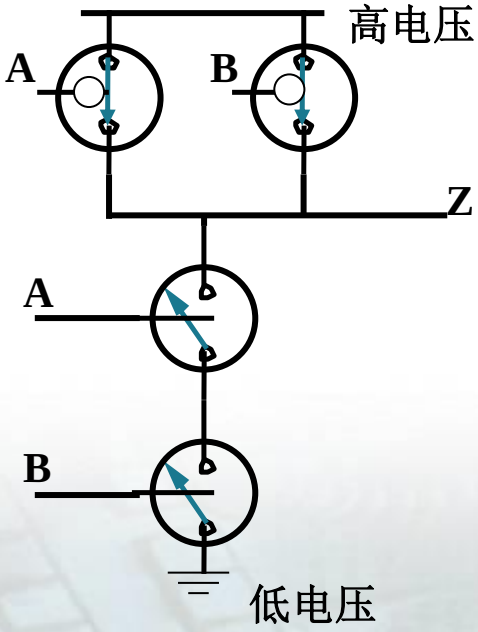
(c)

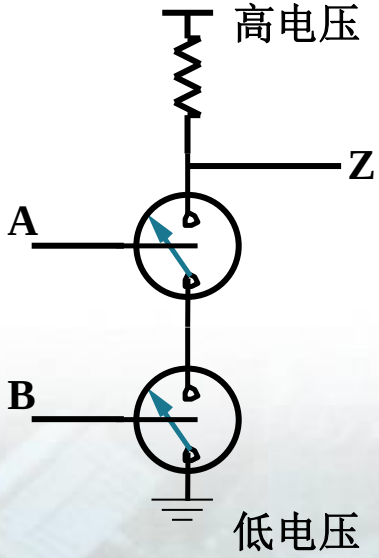


(d)



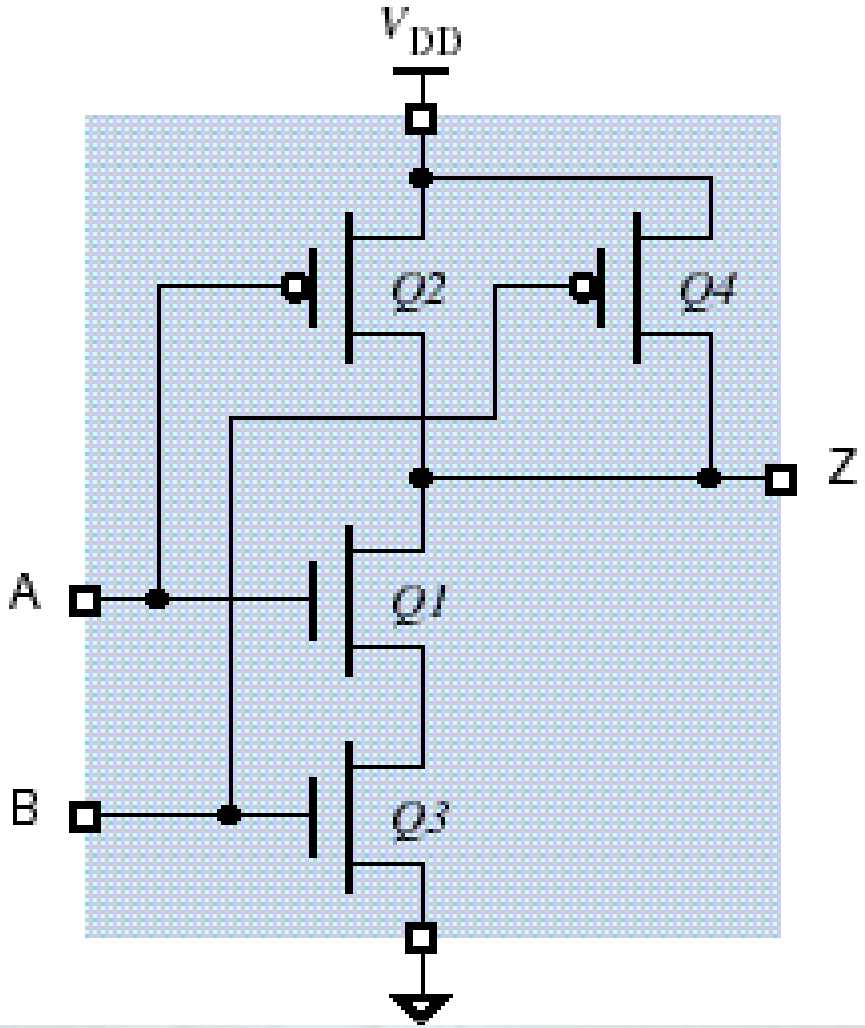
(e)





课程回顾：CMOS与非门(NAND)

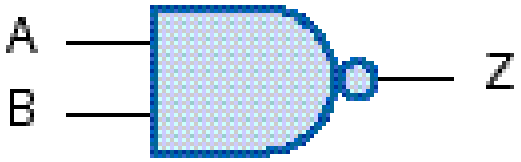
(a)



(b)

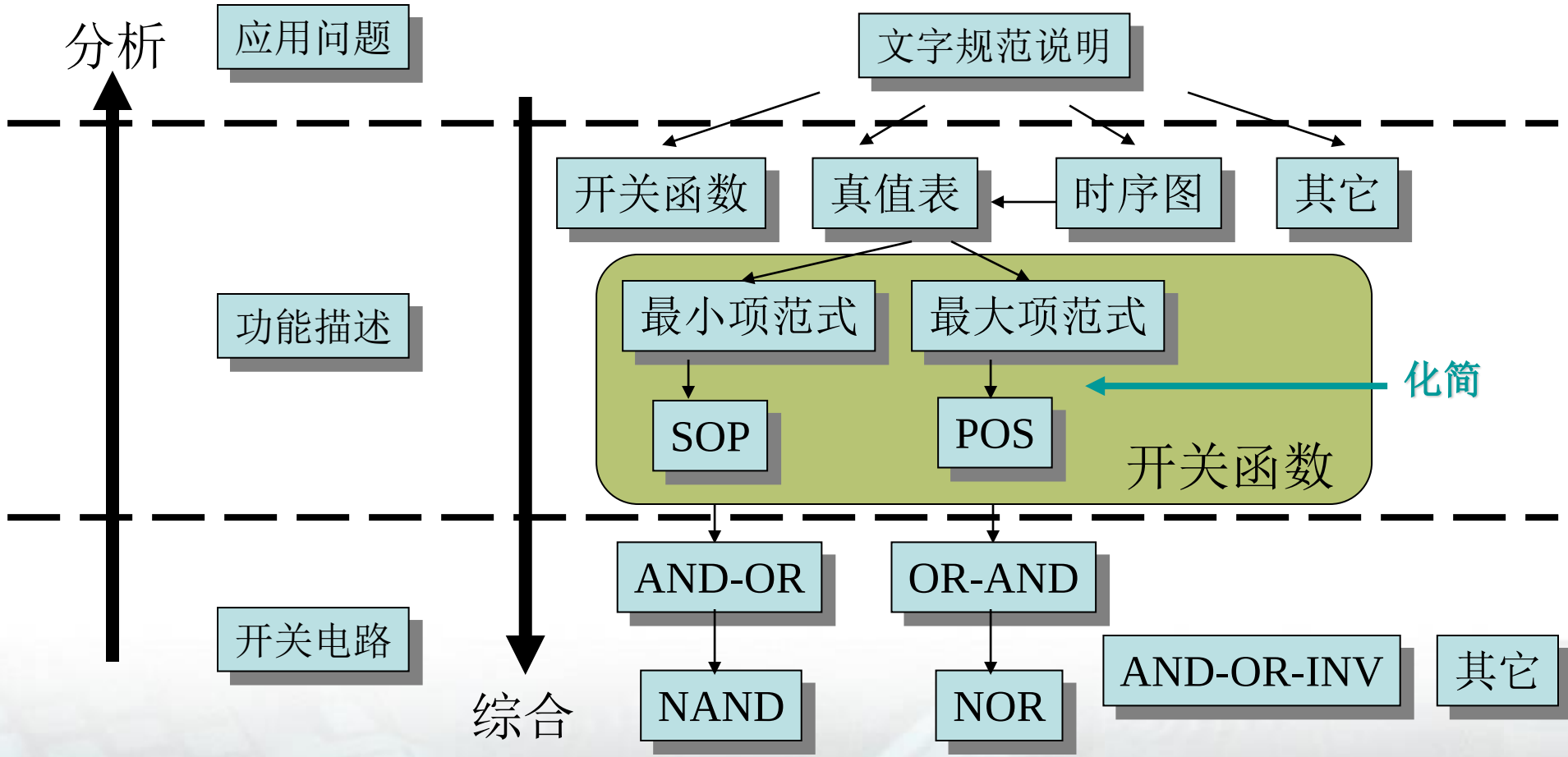
A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L

(c)



组合电路分析与设计

- 组合电路的分析与综合



北京大学

Verilog 硬件描述语言

数字电路的表示法

- 物理器件 Physical devices (transistors，relays)
- 开关 **Switches**
- 真值表 **Truth tables**
- 布尔代数 **Boolean algebra**
- 逻辑门 **Logic Gates**
- 波形图 **Waveform**
- 有穷状态行为 **Finite state behavior**
- 寄存器传输行为 **Register-transfer behavior**
- 并发抽象描述Concurrent abstract specification

硬件描述语言
Hardware description languages (HDL)

- 在多种抽象级别上描述硬件
- 结构级描述 (Structural description)
  - 原理图的文本替代
  - 功能模块(module)的模块层次化组合
- 行为级/功能级描述 (Behavioral/functional description)
  - 描写模块做什么？而不描述如何做
  - 可综合产生模块电路
- 模拟语义

硬件描述语言HDLs

- Abel (~1983) - Data-I/O开发
  - 面向可编程逻辑器件
  - 对状态机的支持不足够得好
- ISP (~1977) – CMU研究项目
  - 面向模拟，不能综合
- Verilog (~1985) - Gateway 开发 (被Cadence合并)
  - 类Pascal和C语言
  - 延迟delays只在模拟器中有效
  - 很容易有效地编程
- VHDL (~1987) – 美国国防部DoD发起的标准
  - 类Ada语言 (着重于可复用性和可维护性)
  - 显式的模拟语义
  - 很通用但也很繁琐

HDL和编程语言的关系

- 程序结构Program Structure
  - 多次例化相同类型的模块
  - 通过原理图表明模块之间的连接关系
  - 模块的层次化结构
- 赋值Assignment
  - 连续赋值continuous assignment (逻辑始终计算）
  - 传播延迟propagation delay (计算耗费时间)
  - 信号时序非常重要 (计算的结果何时产生效果)
- 数据结构Data structures
  - 位宽显式的拼写出来，不支持动态结构
  - 不支持指针
- 并行性Parallelism
  - 硬件是自然地并行运行（必须支持多线程）
  - 赋值并行的发生（不仅仅是串行执行的）

事件驱动的模拟：硬件是并行的！

U.S. Patent Dec. 9, 1997 Sheet 3 of 8 5,696,94

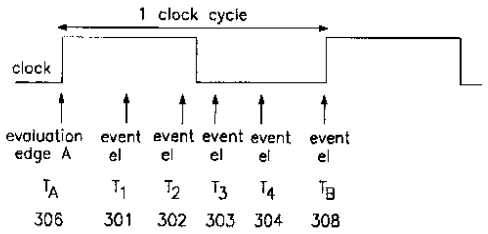


FIG. 3

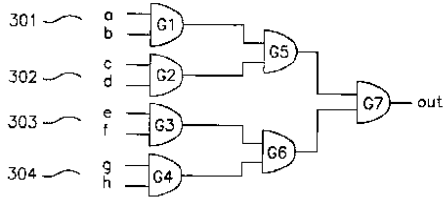


FIG. 4

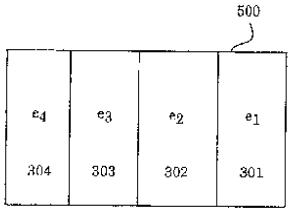


FIG. 5a

U.S. Patent Dec. 9, 1997 Sheet 5 of 8 5,696,942

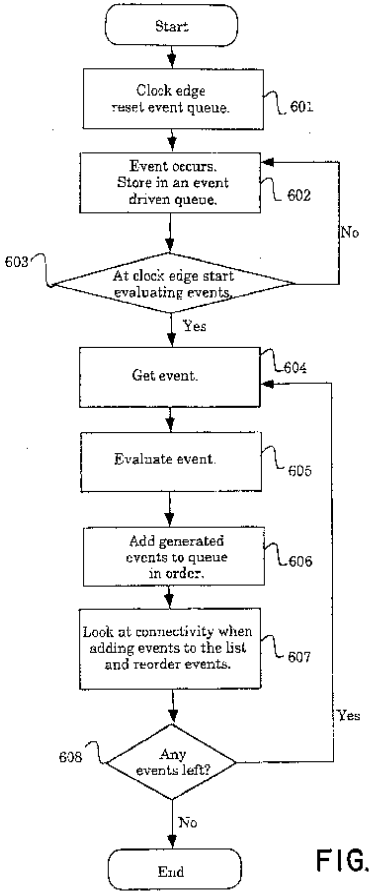


FIG. 6a

Patent Dec. 9, 1997 Sheet 6 of 8 5,696,942

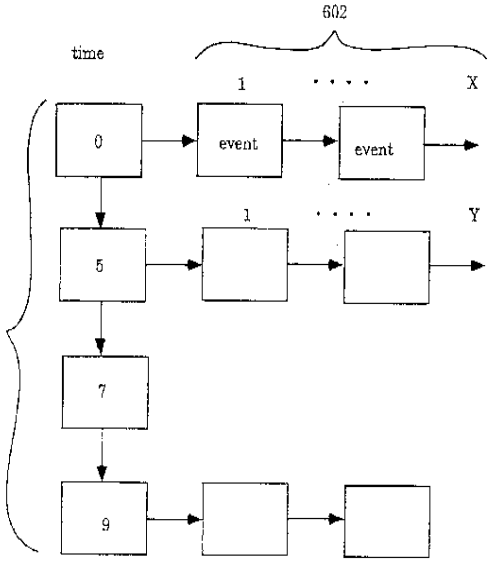


FIG. 6b

HDL和组合逻辑

- Modules：说明输入，输出，双向和内部信号
- 连续赋值：一个门的输出任何时刻都是输入的函数结果（不需要调用）
- 传播延迟：输入影响输出的时间和延迟的概念
- 构成Composition：通过线将模块连接在一起
- 层次化：模块分解成一些功能模块

组合电路的Verilog描述

- 类型：
  - Input/output
  - Wire
  - Reg
- 赋值：
  - Assign
  - 阻塞赋值=
  - 选择赋值
- Always @()
  - 敏感向量表
  - if-else
  - case/casex
- 信号的合并与拆分

Verilog模块定义- module-endmodule

```
Module xor_gate (z, a, b);      // 模块定义
  input a, b;                   // 输入端口定义
  output z;                     // 输出端口定义

  <模块内功能描述>

endmodule                       // 模块结束
```

异或门XOR的Verilog结构级描述

```
Module xor_gate (z, a, b);          // 模块定义
  input a, b;                       // 输入端口定义
  output z;                         // 输出端口定义
  wire abar, bbar, t1, t2           // 连线定义

  not_gate inva(abar, a);           // 实例化一个非门inva
  not_gate invb(bbar, b);           // 实例化一个非门invb
  and_gate and1(t1, a, bbar);       // 实例化一个与门and1
  and_gate and2(t2, b, abar);       // 实例化一个与门and2
  or_gate or1(z, t1, t2);           // 实例化一个或门or1
endmodule                           // 模块结束
```

异或门的Verilog行为级描述- always @()

```
Module xor_gate (z, a, b);          // 模块定义
  input a, b;                       // 输入端口定义
  output z;                         // 输出端口定义
  reg z;                            // 寄存器定义

  always @ (a, b) begin             // @() 敏感向量表
                                    //  a或b值变化时才执行
     z = a ^ b;                     // 位异或计算;
     // always体中的=只能对reg类型赋值
  end                               // always结束
endmodule                           // 模块结束
```

Verilog运算符和优先级

	运算符	含义
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	逻辑左移/逻辑右移
	<<<, >>>	算术左移/算术右移
	<, <=, >, >=	相对比较
	==, !=	相等比较
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	条件

Verilog中的always中的阻塞赋值

```
Module xor_gate (z, a, b);          // 模块定义
  input a, b;                       // 输入端口定义
  output z;                         // 输出端口定义
  reg temp, z;                      // 寄存器定义

  always @ (a, b) begin             // @() 敏感向量表 //@(*)
                                    //  a或b值变化时才执行
     temp = a ^ b;                  // 阻塞赋值;
     z = temp;                      // z = a ^ b;
  end                               // always结束
endmodule                           // 模块结束
```

异或门的Verilog行为级描述- if-else

```
Module xor_gate (z, a, b);          // 模块定义
  input a, b;                       // 输入端口定义
  output z;                         // 输出端口定义
  reg z;                            // 寄存器定义

  always @ (a, b) begin             // @() 敏感向量表
                                    //  a或b值变化时才执行
     if (a) z=~b else z = b;        // 0通过1取反
  end                               // always结束

endmodule                           // 模块结束
```

Verilog数值的表示

数字	位	基数	值	存储
3'b101	3	2	5	101
'b11	?	2	3	000… 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00… 0101010

异或门的verilog行为级描述- case()

```
Module xor_gate (z, a, b);                  // 模块定义
  input a, b;                               // 输入端口定义
  output z;                                 // 输出端口定义
  reg z;                                    // 寄存器定义

  always @ (a, b) begin                     // @() 敏感向量表
    case(a)                                 // case体
       1’b0: z = b;                         // 0通过
       1’b1: z = ~b;                        // 1取反
       default: z = x;                      // 默认输出
    endcase                                 // case结束
  end                                       // always结束
endmodule                                   // 模块结束
```

异或门的verilog行为级描述- assign

```
Module xor_gate (z, a, b);      // 模块定义
  input a, b;                   // 输入端口定义
  output z;                     // 输出端口定义

assign z = a ^ b;               // 无论输入值是否
                                // 变化都执行
// assign 可以对output, reg, wire类型赋值

endmodule                       // 模块结束
```

Verilog中的- assign阻塞赋值

```
Module xor_gate (z, a, b);      // 模块定义
  input a, b;                   // 输入端口定义
  output z;                     // 输出端口定义
  wire temp;                    // 连线定义

assign temp = a ^ b;            // 阻塞赋值
assign z = temp;                // z = a ^ b;

endmodule                       // 模块结束
```

Verilog中的 assign阻塞赋值的并行性

```
Module xor_gate (z, a, b);      // 模块定义
  input a, b;                   // 输入端口定义
  output z;                     // 输出端口定义
  wire temp;                    // 连线定义

assign z = temp;                // z = a ^ b;
assign temp = a ^ b;            // 阻塞赋值
// 交换顺序仍然正确，描述的硬件是一样的

endmodule                       // 模块结束
```

异或门的verilog行为级描述-选择assign

```
module xor_gate (z, a, b);      // 模块定义
  input a, b;                   // 输入端口定义
  output z;                     // 输出端口定义

 assign z = (a) ? ~b : b;       // 0通过1取反

endmodule                       // 模块结束
```

1-bit全加器进位的verilog行为级描述-casex

```
module add1( cout, a, b, cin);
    input a, b, cin;
    output cout;
    reg cout;

    always @(a, b) begin
        casex({a, b, cin})
            // 3位输入为1的个数：大于等于2，进位1; 小于等于1，进位0
            3'b11x, 3'b1x1, 3'bx11: cout = 1;
            3'b00x, 3'b0x0, 3'bx00: cout = 0;
          default: cout = 1'bx;
        endcase
    end
endmodue
```

2-bit全加器的verilog行为级描述-信号连接

```
module adder2(s, cout, a, b, cin)
   input [1:0] a,b;                    // 2-bit输入信号
   input cin;                          // 1-bit输入信号
   out [1:0] s;                        // 2-bit输出信号
   out cout;                           // 1-bit输出信号

assign {cout, s} = a   + b  + cin;     // 信号连接
//          1bit   2bit   2bit    2bit   1bit
endmodue
```

2-bit全加器的Verilog行为级描述-信号连接

```
module adder2(s, cout, a, b, cin)
   input [1:0] a,b;                    // 2-bit输入信号
   input cin;                          // 1-bit输入信号
   out [1:0] s;                        // 2-bit输出信号
   out cout;                           // 1-bit输出信号
   reg [1:0] s;                        // 2-bit寄存器
   reg cout;                           // 1-bit寄存器

   always @ (a, b, cin) begin
     {cout, s} = a + b + cin;          // 信号连接
   end
endmodue
```

2-bit 全加器的Verilog行为级描述- 信号连接

```
module adder2(in5, out2)
   input [4:0] in5;                          // 5-bit输入信号
   out [2:0] out3;                           // 2-bit输出信号
   wire [1:0] a, b, s;                       // 2-bit连线
   wire cin, cout;                           // 1-bit连线

assign {a, b, cin} = in5;                    //信号连接
//  a = in5[4:3]; b = in5[2:1]; cin = in5[0];
assign {cout, s} = a + b + cin;              //信号连接
assign out3 = {cout, s};                     //信号连接

endmodue
```

测试平台Testbench

```
module testbench1();
  reg a, b, c;
  wire y;

  // instantiate device under test
  sillyfunction dut(a, b, c, y);

  // apply inputs one at a time
    initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1;               #10;
    b = 1; c = 0;        #10;
    c = 1;               #10;
    a = 1; b = 0; c = 0; #10;
    c = 1;               #10;
    b = 1; c = 0;        #10;
    c = 1;               #10;
  end
endmodule
```

Lab2：二进制和BCD的转换模块

- 4月1日 24:00之前提交
- 要求所有的命名都以学号后两位结尾！！！
  - name_<学号后两位>
- 将工程目录压缩成<学号>.zip文件
- 提交到教学网

8位二进制到BCD码的转换

操作	百位	十位	个位	二进制	
十六进制				A	B
开始				1 0 1 0	1 0 1 1
移位3次			1 0 1	0 1 0 1	1
加3			1 0 0 0	0 1 0 1	1
移位4		1	0 0 0 0	1 0 1 1	
移位5		1 0	0 0 0 1	0 1 1	
移位6		1 0 0	0 0 1 0	1 1	
移位		1 0 0 0	0 1 0 1	1	
加3		1 0 1 1	1 0 0 0	1	
移位	1	0 1 1 1	0 0 0 1		
结束 BCD	1	7	1		

思考：BCD码如何转二进制？其它特殊进制的算法呢？

二进制到BCD码的转换模块

```
module binary2BCD(
  input [13:0] binary,
  output reg [3:0] thousands,
  output reg [3:0] hundreds,
  output reg [3:0] tens,
  output reg [3:0] ones
  );

  reg [29:0] shifter;
  integer i;

  always @(binary)
  begin
    shifter[13:0] = binary;
    shifter[29:14] = 0;
for (i = 0; i< 14; i = i+1) begin
        if (shifter[17:14] >= 5)
            shifter[17:14] = shifter[17:14] + 3;
        if (shifter[21:18] >= 5)
            shifter[21:18] = shifter[21:18] + 3;
        if (shifter[25:22] >= 5)
            shifter[25:22] = shifter[25:22] + 3;
        if (shifter[29:26] >= 5)
            shifter[29:26] = shifter[29:26] + 3;
        shifter = shifter  << 1;
    end
    thousands = shifter[29:26];
    hundreds = shifter[25:22];
    tens = shifter[21:18];
    ones = shifter[17:14];
  end
endmodule
```

组合电路的实现技术

- 标准逻辑门Standard gates
  - 逻辑门封装芯片, 标准单元库
- 规整逻辑Regular logic
  - 多选器multiplexers
  - 译码器decoders
- 两级可编程逻辑Two-level programmable logic
  - PALs, PLAs,
- 存储器实现逻辑
  - ROMs
  - 现场可编程门阵列(Field-Programming Gate Array， FPGA)

存储器和组合逻辑

- 用存储器组合逻辑实现

F0 = A' B' C + A B' C' + A B' C

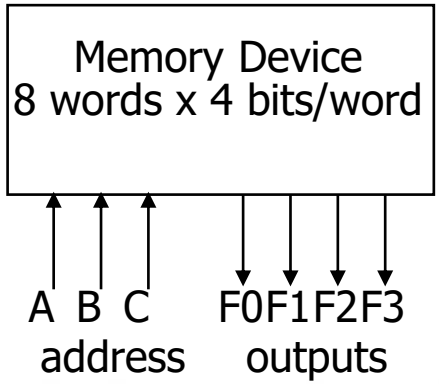
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

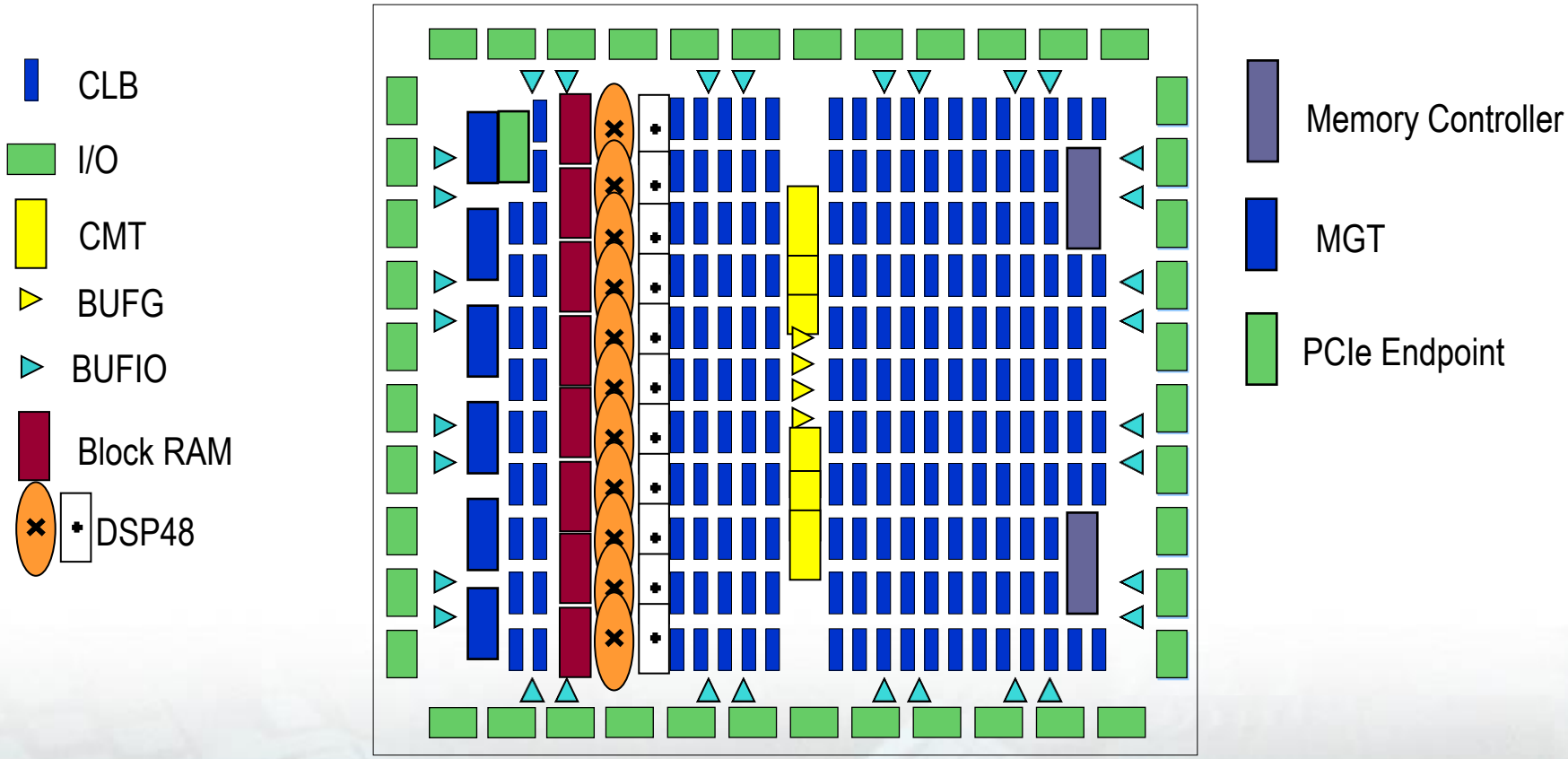
真值表



框图

Xilinx FPGA概述

- FPGA Field Programmable Gate Array
  - 现场可编程门阵列

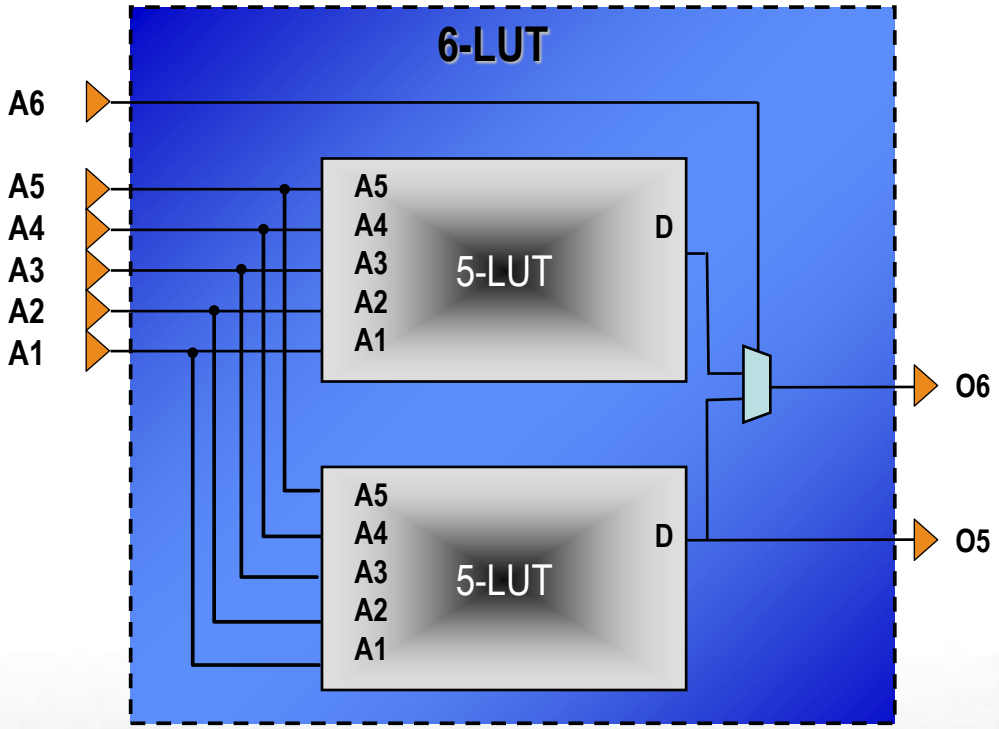


Xilinx FPGA概述

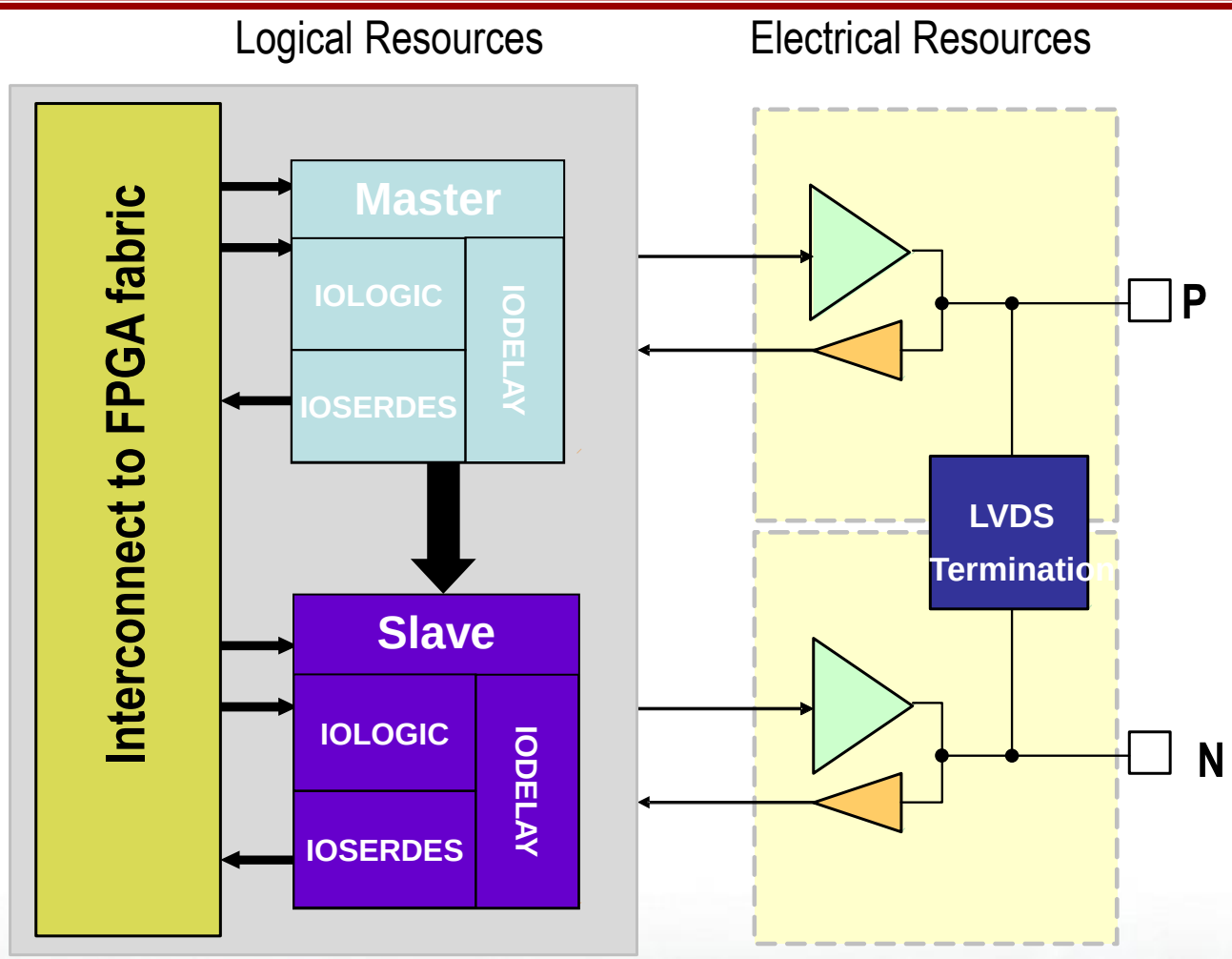
- 所有的Xilinx FPGAs都包含相同的基本资源
  - 逻辑资源Logic Resources
    - 片(Slices)组成CLB(Configurable Logic Block 可配置逻辑块）
      - 包含组合逻辑和寄存器资源
    - 存储器Memory
    - 乘法器Multipliers
  - 互连资源Interconnect Resources
    - 可编程互连资源Programmable interconnect
    - 输入输出块IOBs
      - FPGA与外部的接口
  - 其它资源Other resources
    - 全局时钟缓冲器Global clock buffers
    - 边缘检测逻辑Boundary scan logic

6-Input LUT with Dual Output

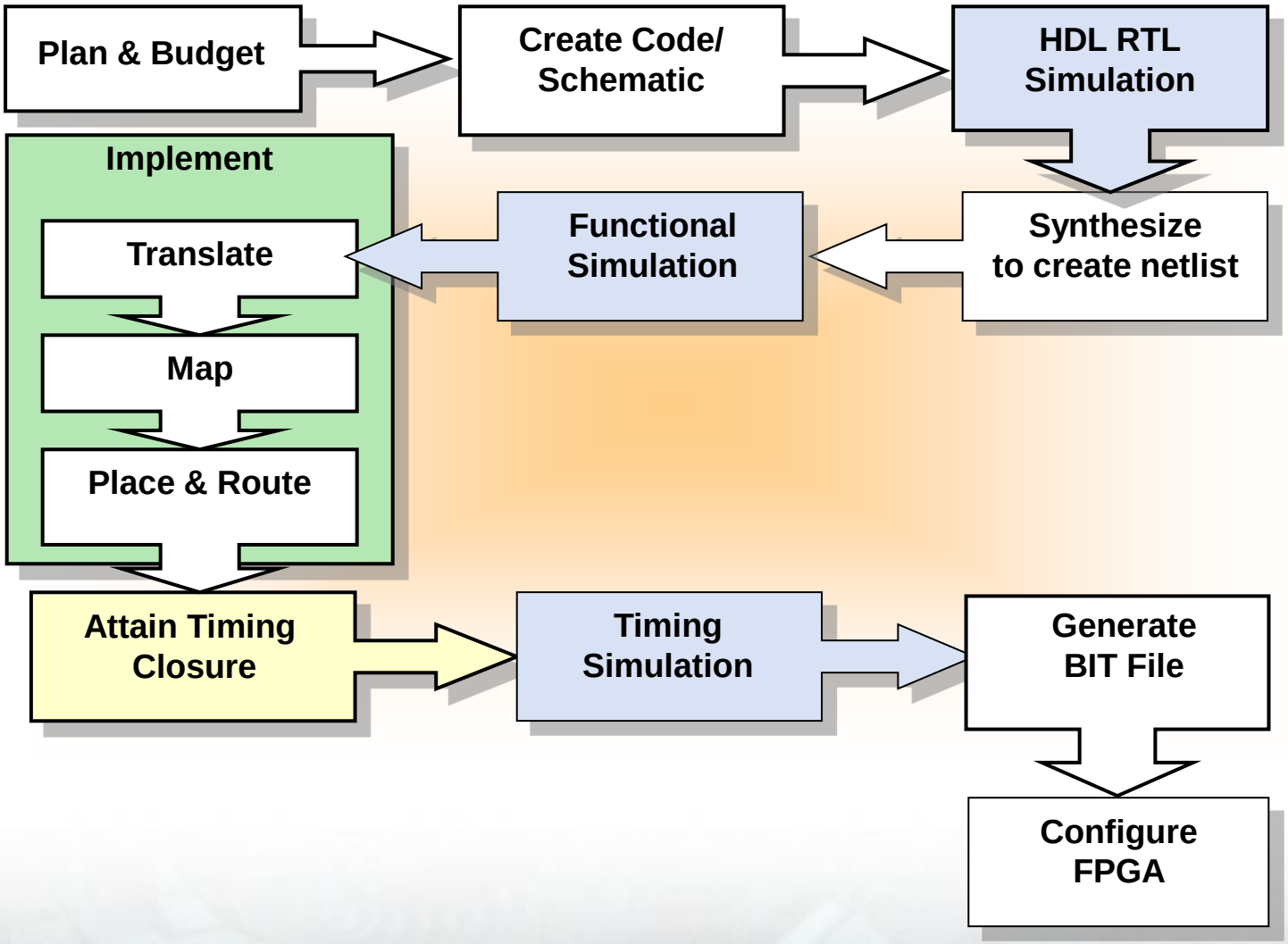
- 6-input LUT 包含两个共用输入的5-input LUTs
  - 影响一个6-input LUT最优速度
  - 一个或者两个输出
  - 任意6变量的函数
  - 两个独立的5变量函数



输入输出块(IOB)的结构图



Xilinx设计流程



电子设计自动化软件

- CAD, Computer-aid Design
- EDA, Electronic Design Automatic
- ESL, Electronic System Level

- 系统设计(C/C++，SystemC）
- 设计输入(Verilog, VHDL, Schematic)
- 设计验证(Verification)
- 设计综合(ASIC, FPGA, Analog)
- 后端(Back-end)设计

1 设计输入 Design Entry

- 原理图输入
- 激励和输出逻辑方程
- 状态表
- 状态图或者ASM图
- 硬件描述语言
  - Verilog
  - VHDL

2 综合 Synthesis

- 状态优化
- 选择状态分配方案
- 选择触发器类型
- 对组合逻辑进行优化

3 设计验证Verification

- 功能分析与验证
  - 模拟 Simulation
  - 形式化验证 Formal Verification
- 时序分析
  - Timing Analysis
  - Timing Methodology

4后端(Back-end)设计

- 时钟树生成
- Place & Route
- 参数提取，后仿真
- 物理验证（信号完整性分析等）
- 形成加工工艺文件
  - FPGA
  - ASIC, GDSII
  - …