基于FPGA的
组合逻辑电路设计和实现
EDA实验一

赵晓燕
电工电子实验中心

主要内容

一、FPGA：PLD的最新发展

二、FPGA设计流程

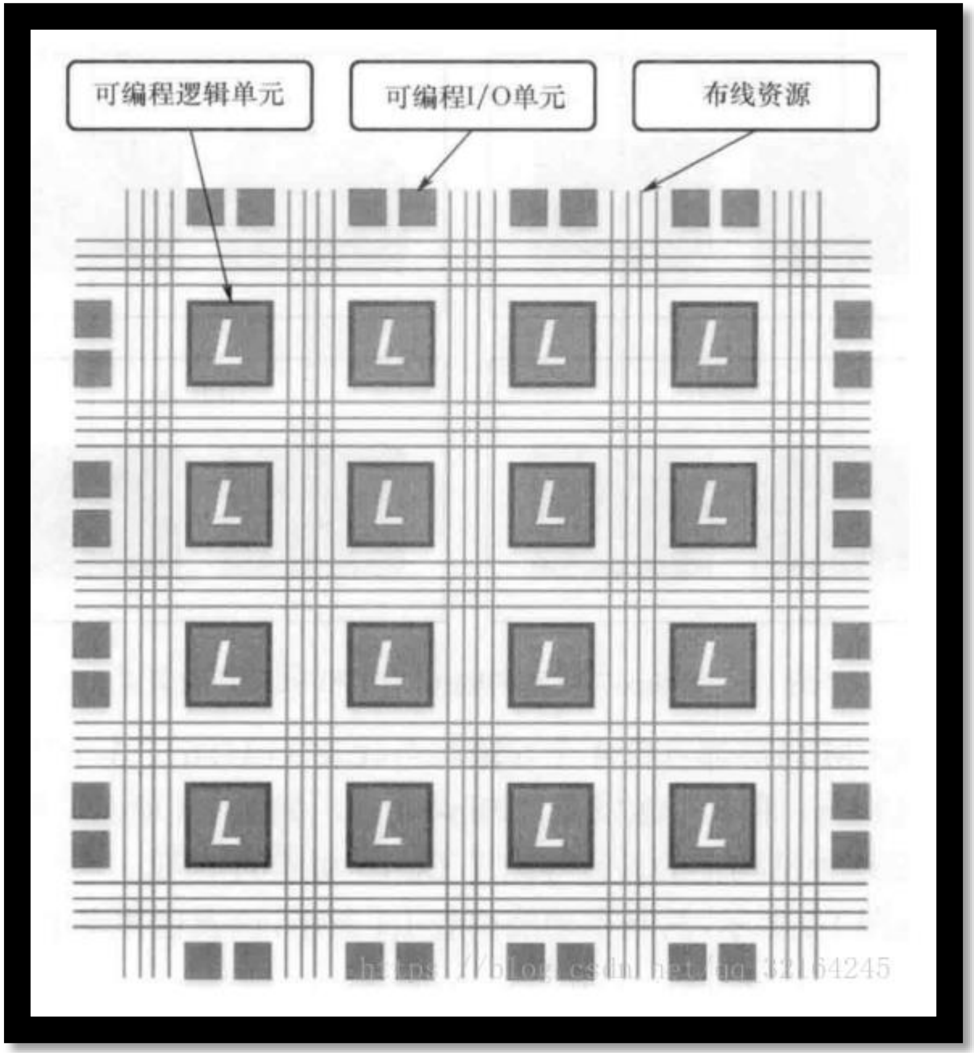
三、HDL硬件描述语言

四、基于FPGA的组合逻辑电路实现

五、EDA实验一内容布置

六、 EDA实验二内容展望

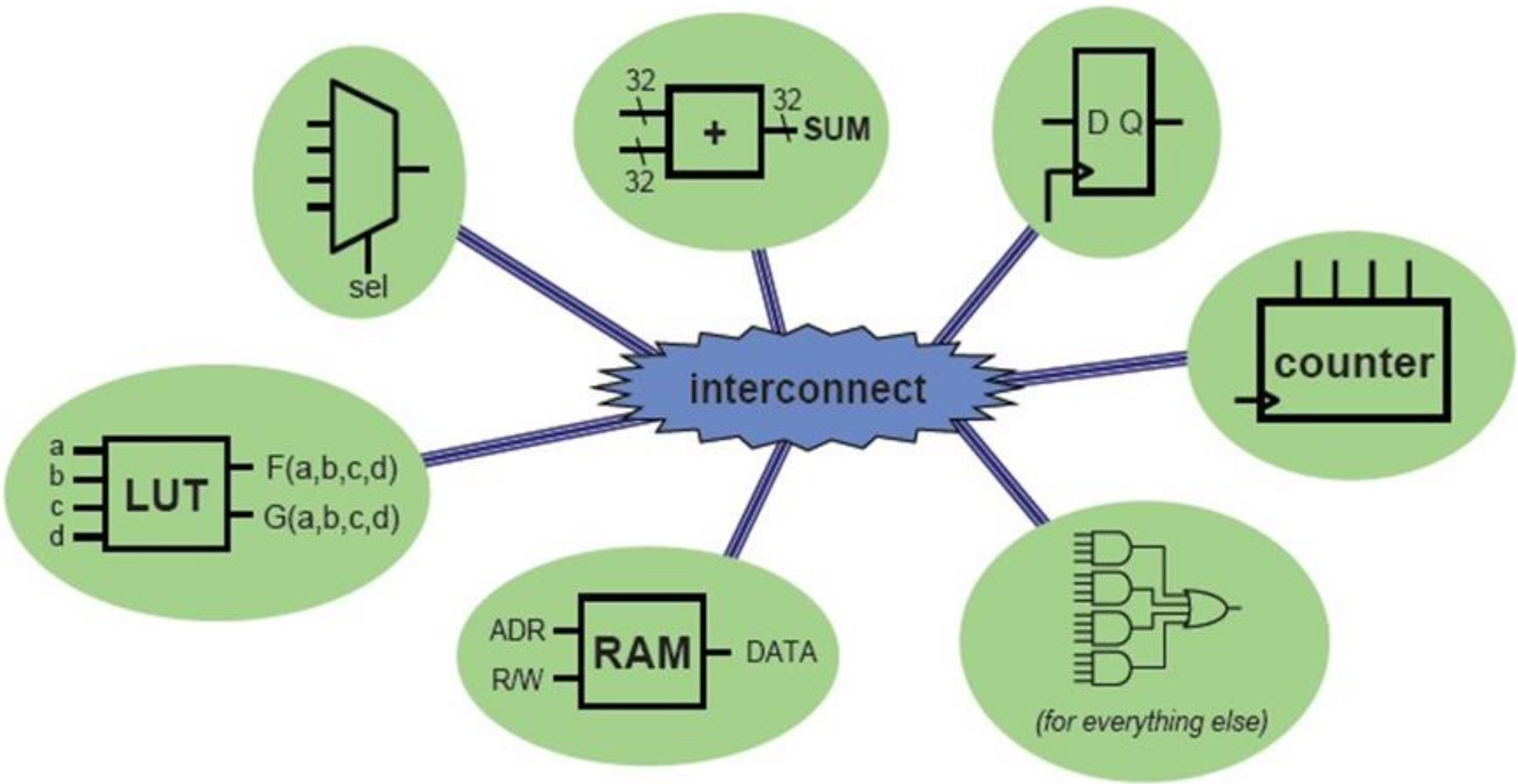
一、FPGA：PLD的最新发展



实验板FPGA芯片：

EP3C16Q240C8N

FPGA：Field Programmable Gate Array



An FPGA is like an electronic breadboard that is wired together by an automated synthesis tool

二、FPGA设计流程

新建工程和设计文件

↓

分析与综合 

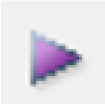
语法检查，并将电路综合成可与FPGA相映射的网表文件，包括基本逻辑单元和单元间的连线。

↓

功能仿真

功能仿真验证电路的逻辑正确性
但不考虑硬件特性及任何时序问题。

↓

全编译 

将电路与FPGA芯片相关联，进行布局布线，产生可下载到FPGA中的数据文件

↓

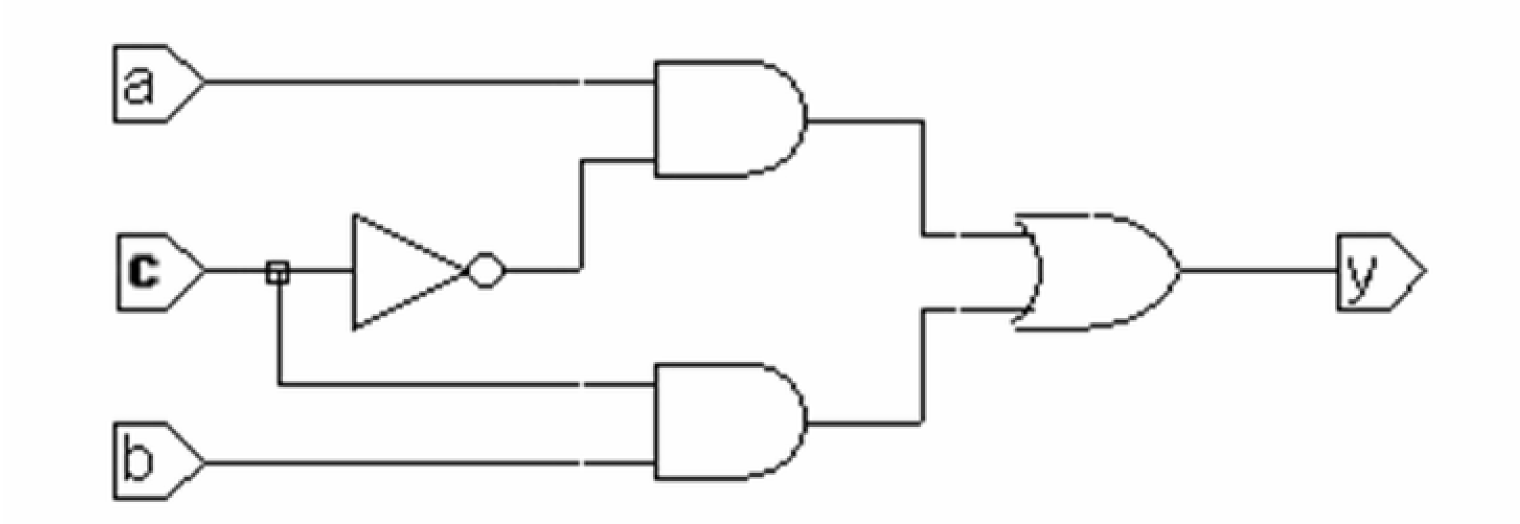
时序仿真

传输延时等时序分析

↓

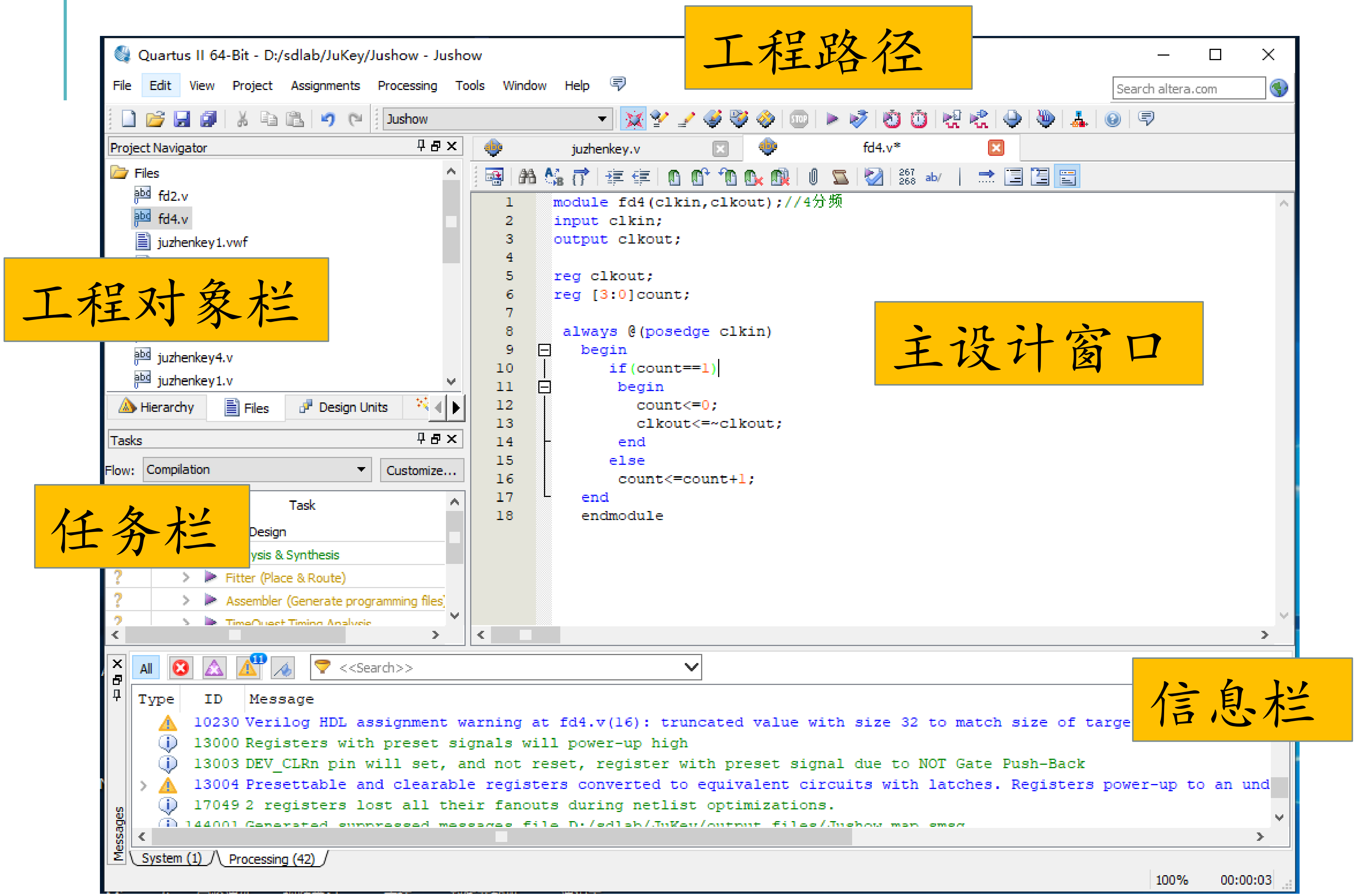
引脚锁定和下载 

操作实例



$$y = ac' + bc$$

Quaturs 13.0软件界面



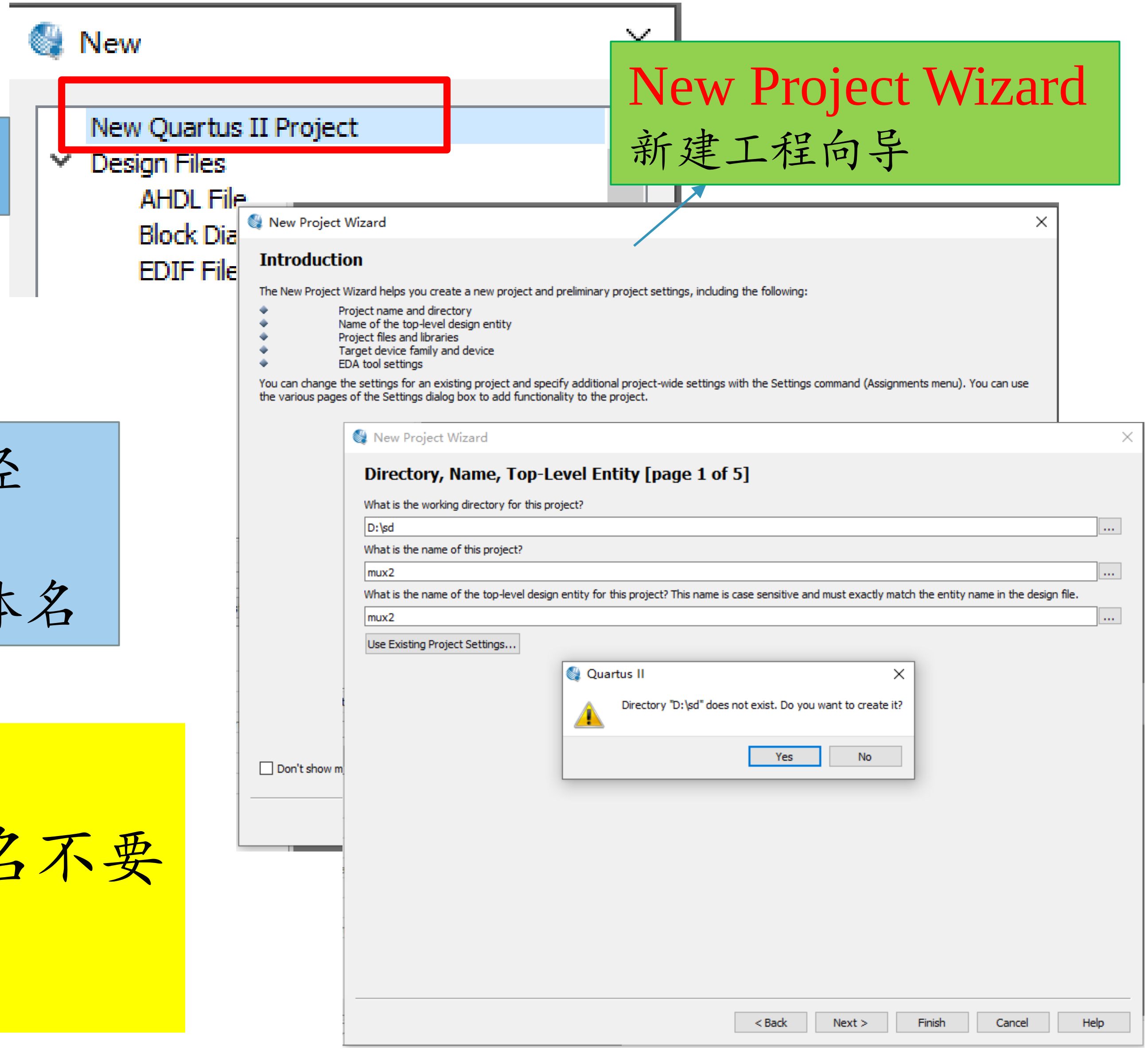
1、新建工程(*.qpf)

打开工程向导

创建工程目录路径
创建工程名
创建工程顶层实体名

注意：
路径名和工程名不要有中文和空格

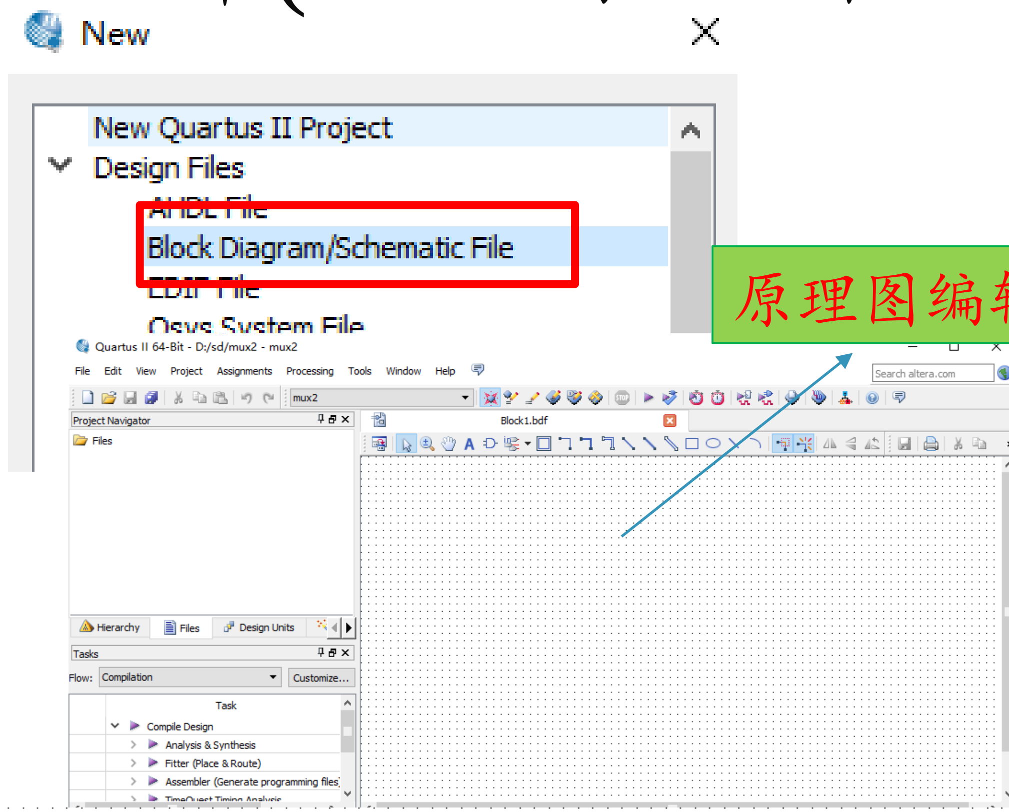
New Project Wizard
新建工程向导



2、输入设计文件(*.bdf原理图文件)

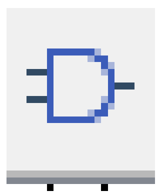
打开原理图编辑器

绘制原理图

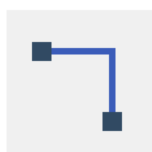


原理图编辑器

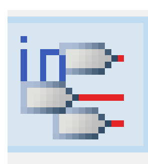
放置器件



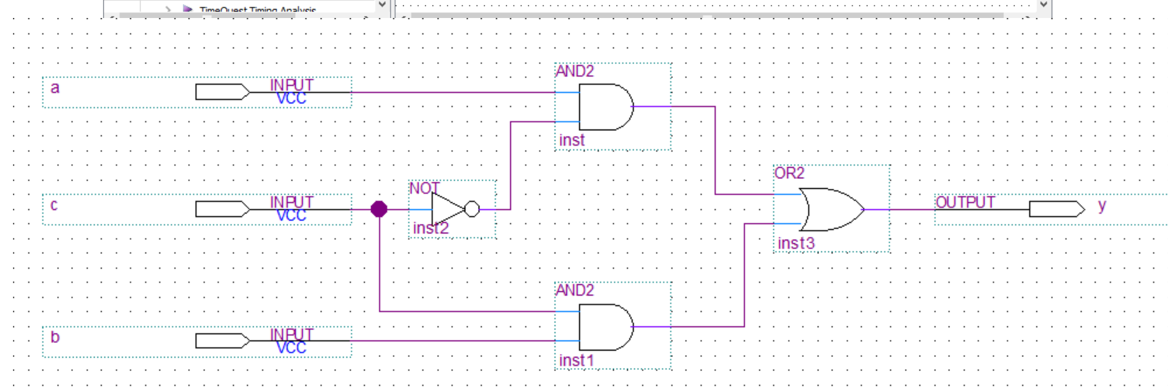
放置连线



放置输入、输出口



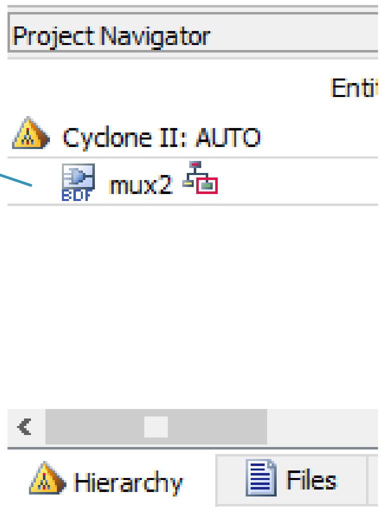
定义输入、输出口



3、综合与分析

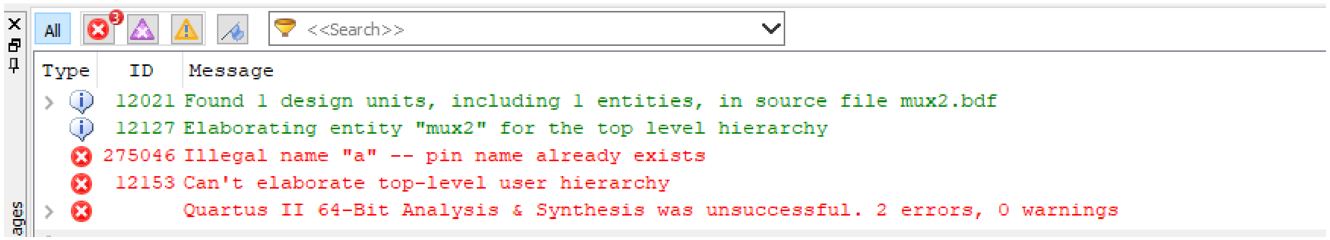
将被仿真文件置顶

综合与分析



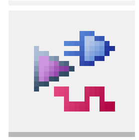
■软件自动完成综合

■注意信息栏修改错误



错误，可检查语法问题、不能检查算法问题（仿真波形，RTL Viewer）

警告，综合分析没问题，全编译会有问题

4、功能仿真  (*.vwf波形文件)

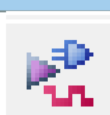
波形编辑器

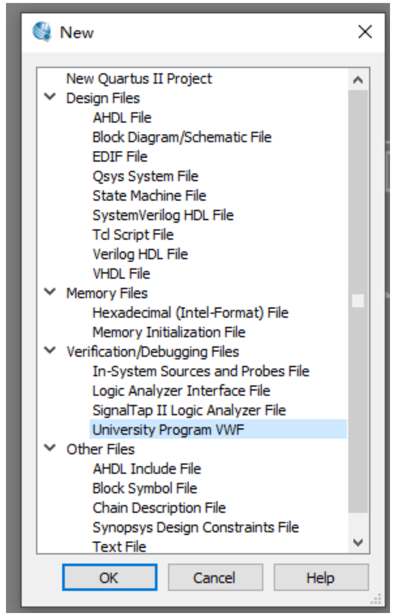
打开波形编辑器

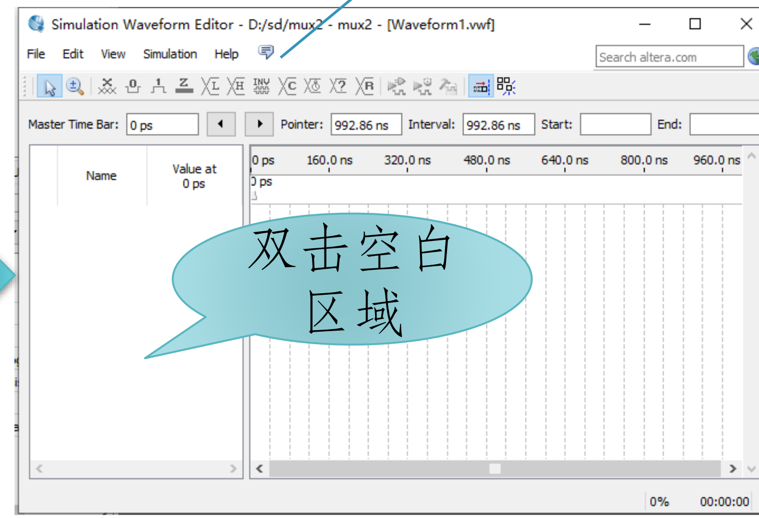
导入信号节点

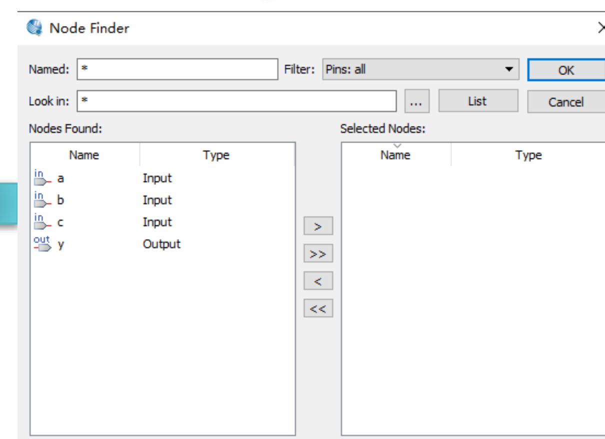
设置仿真信号

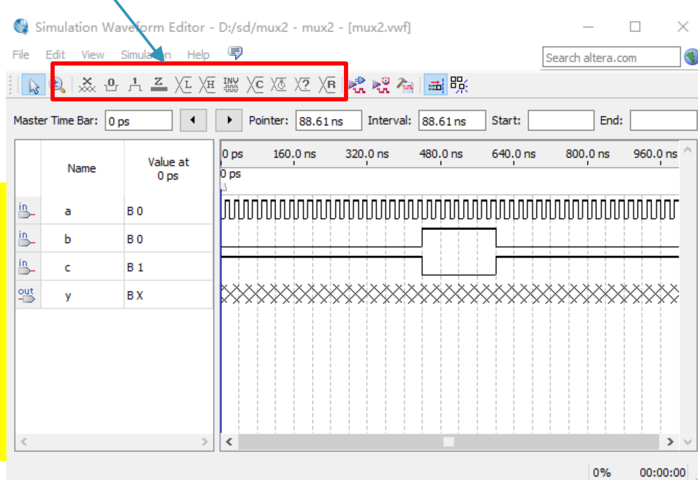
选择仿真器











注意：

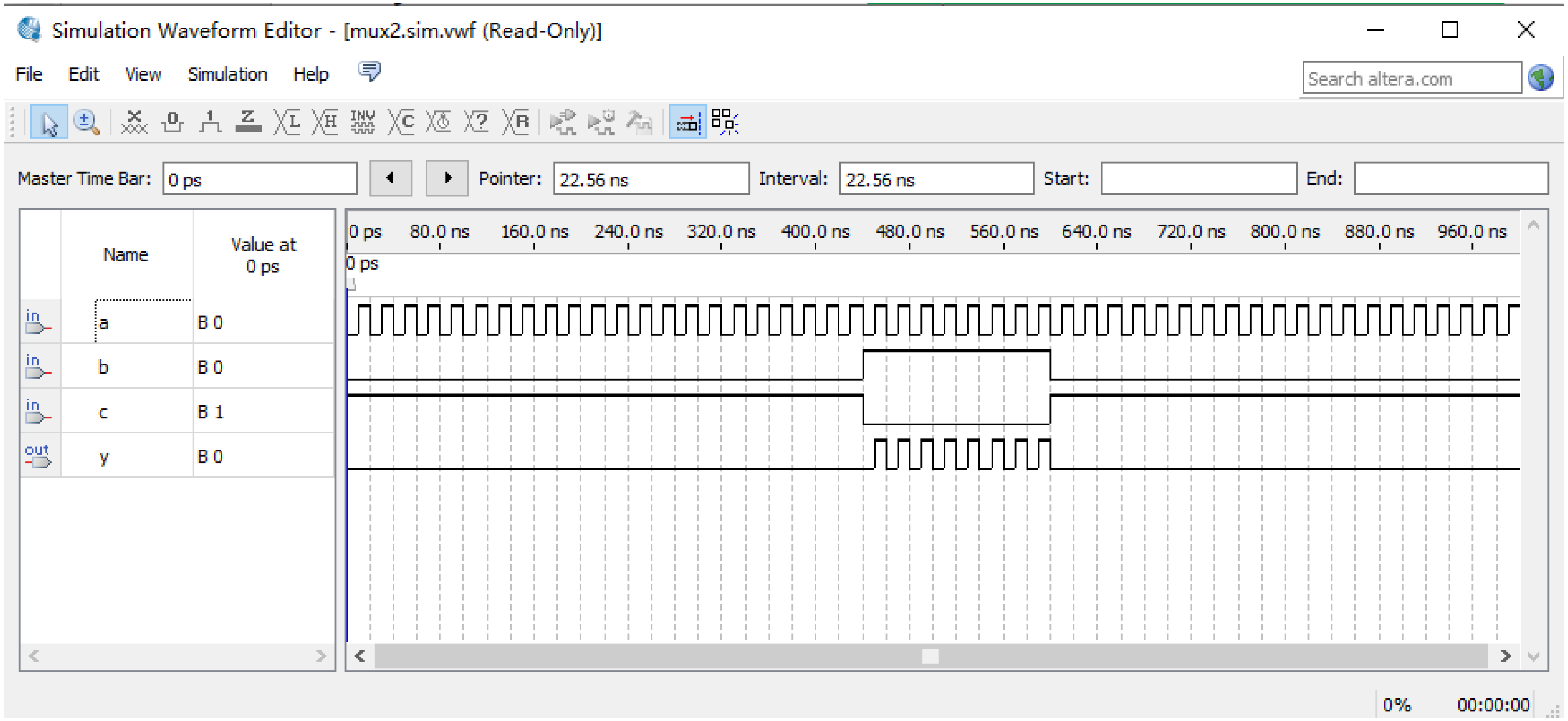
仿真执行的前提：

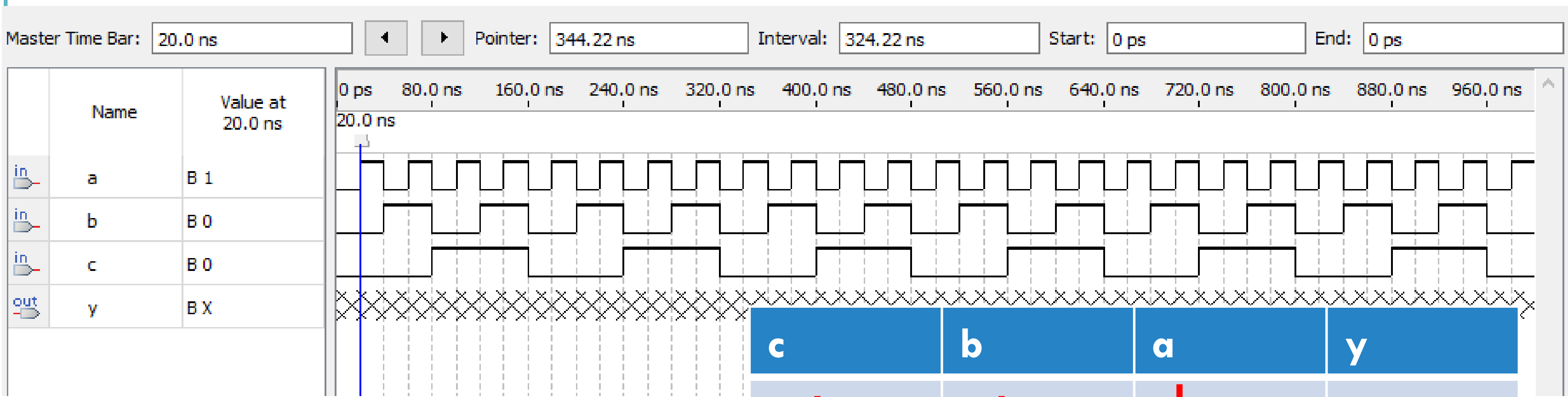
- 被仿真文件置顶
- 综合分析无错误

逻辑功能验证

$$y = ac' + bc$$

仿真波形可检验逻辑设计的正确与否（调试）



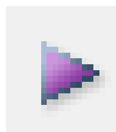


仿真波形设置要遍历输入所有可能情况

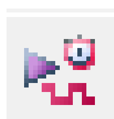
c	b	a	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

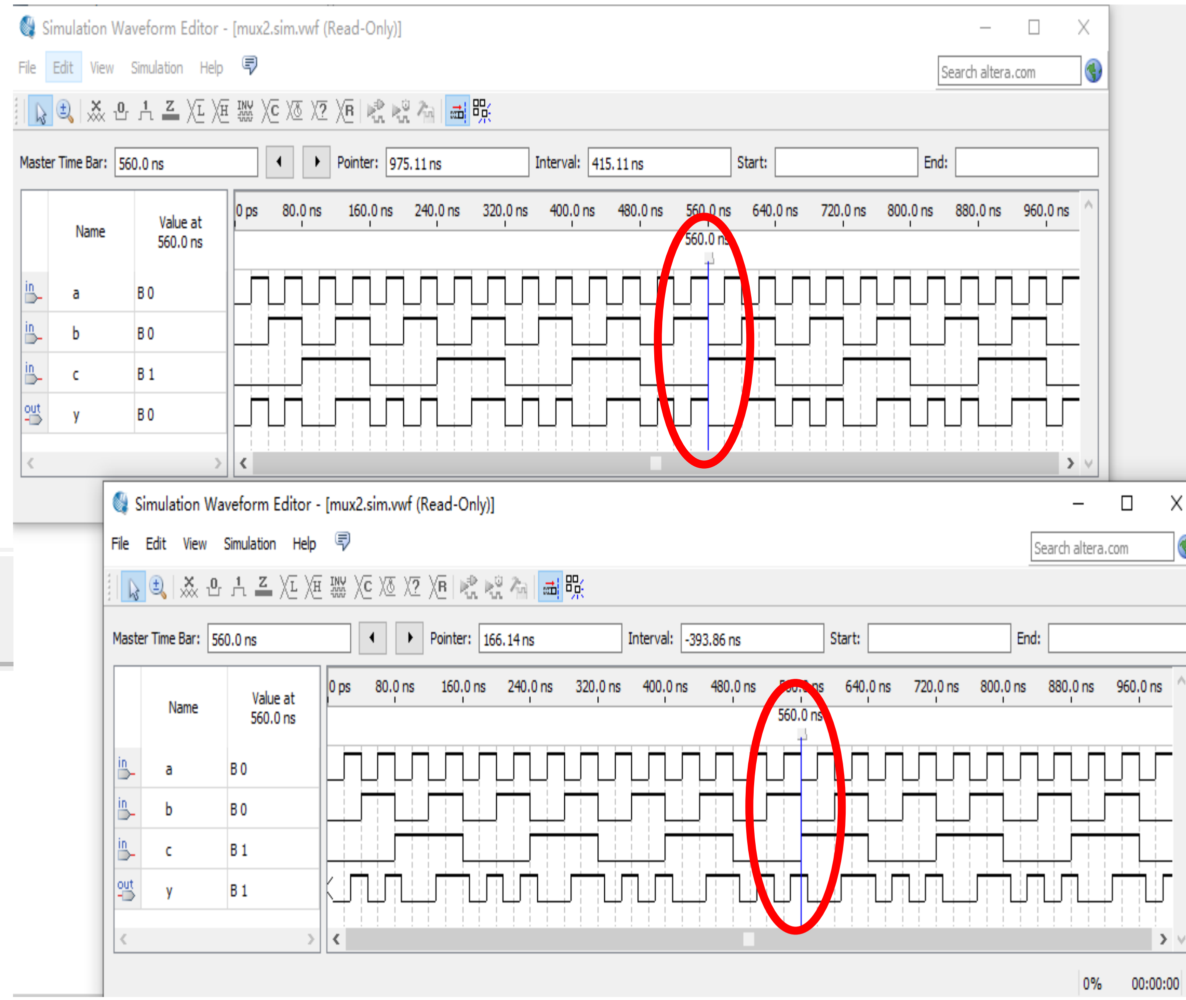
5、时序仿真（可选）

全编译

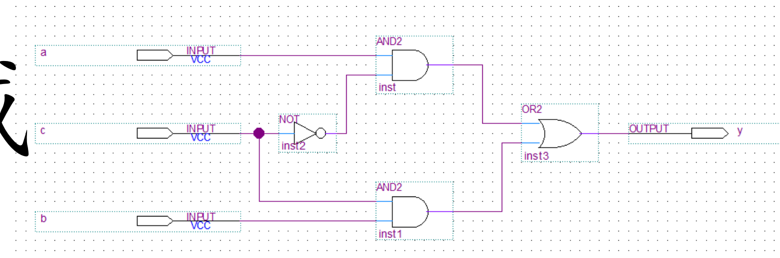


时序仿真



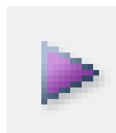


6、引脚锁定和下载

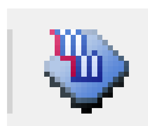


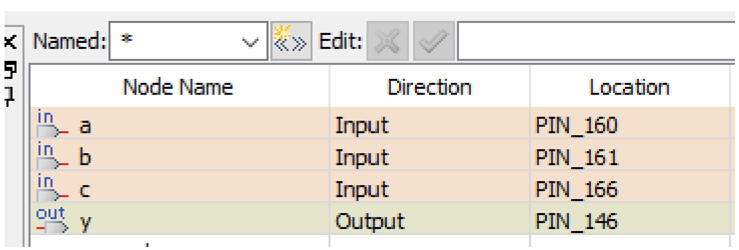
引脚分配

全编译

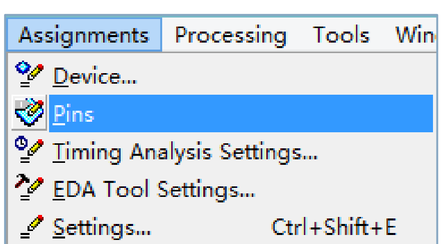


下载




Named: * Edit:

Node Name	Direction	Location
a	Input	PIN_160
b	Input	PIN_161
c	Input	PIN_166
y	Output	PIN_146



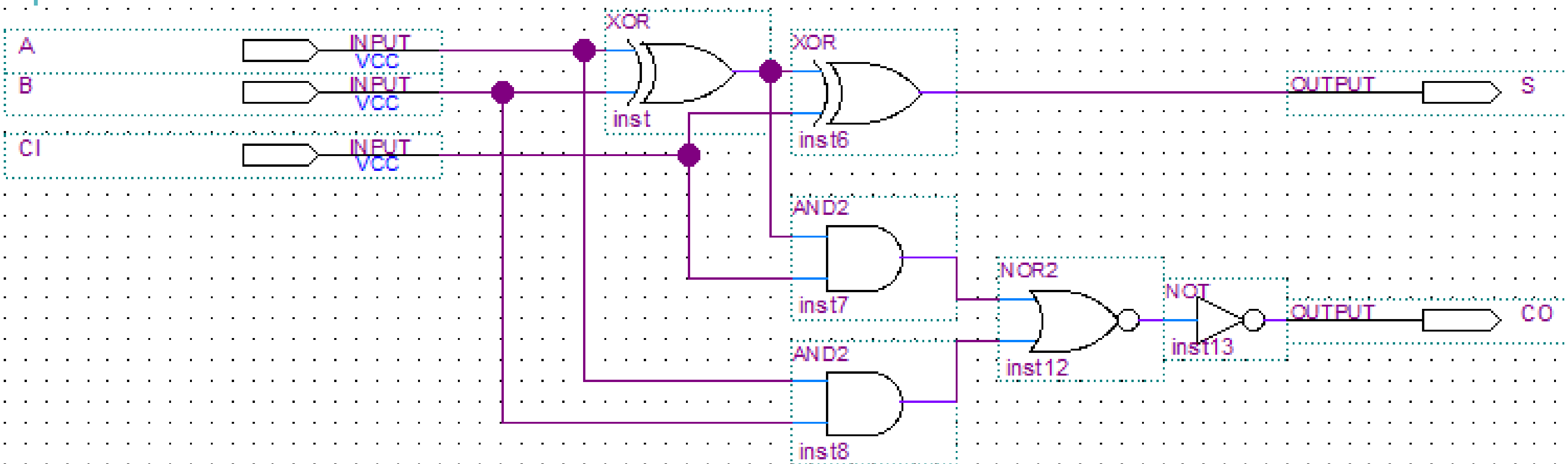
LED3	LED4	LED5	LED6
PIN_145	PIN_146	PIN_167	PIN_168
输入为 1 时，相应的发光二极管被点亮			

SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
PIN_160	PIN_161	PIN_166	PIN_164	PIN_174	PIN_175	PIN_177	PIN_176
开关在上端输出0，开关在下端输出1							

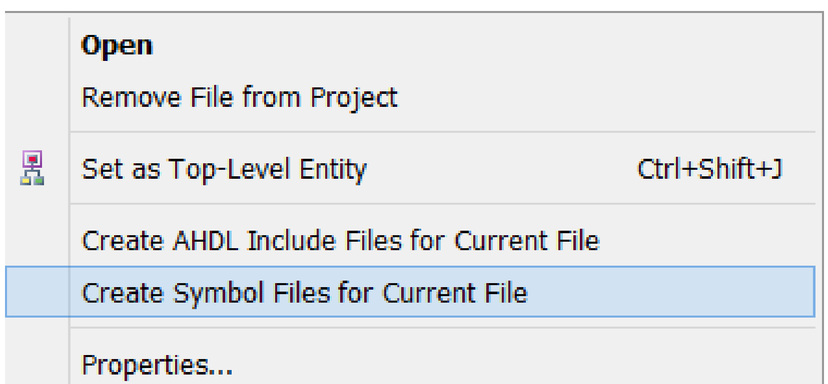
设计文件输入

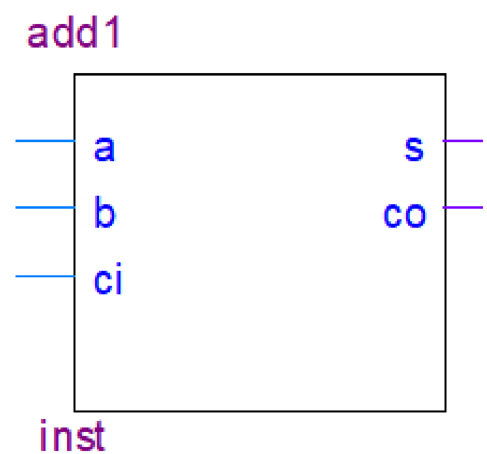
- 原理图输入
- 代码（HDL）输入
- 混合输入

原理图输入（以一位全加器为例）

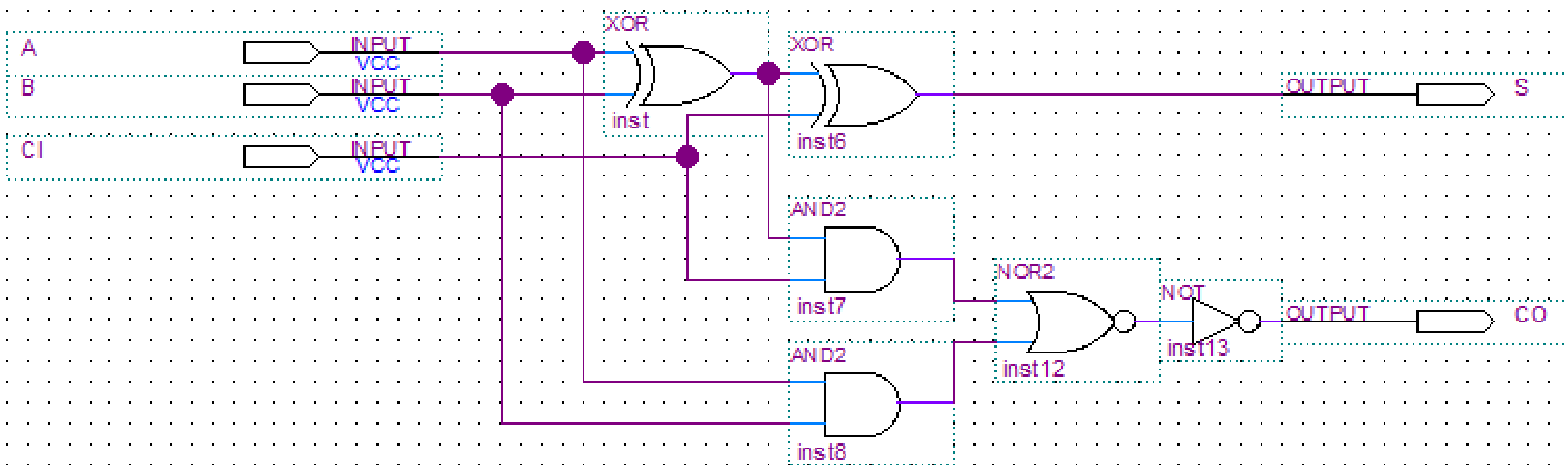


封装

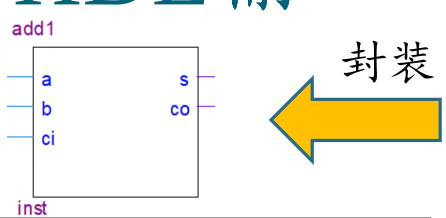


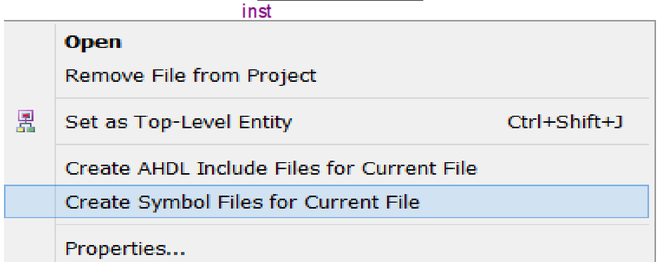


代码（HDL）输入



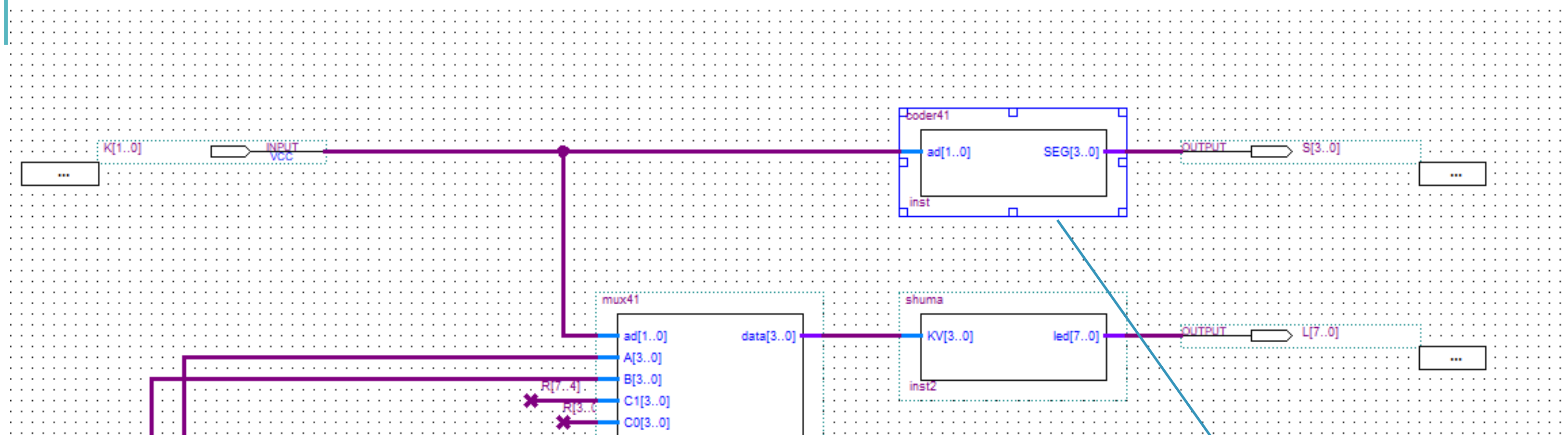
或者：HDL输入






```
module add1(a,b,ci,s,co);
input a,b,ci;
output s,co;

assign {co, s} = a + b + ci;

endmodule
```

混合输入



- 电路信号关系直观明确
- 顶层模块原理图输入
- 底层模块代码输入

```
module coder41(ad,SEG);
input[1:0] ad;
output[3:0]SEG;
reg [3:0]SEG;

always@(*)
   case(ad)
   2'b00: SEG=4'b0111;
   2'b01: SEG=4'b1011;
   2'b10: SEG=4'b1101;
   2'b11: SEG=4'b1110;

   endcase
endmodule
```

三、硬件描述语言HDL

- A high-level programming language offering special constructs to model microelectronic circuits
  - Describe the $\boxed{\text{operation of a circuit}}$ at various level of abstraction
    - Behavior
    - Function
    - Structure
  - Describe the timing of a circuit
  - Express the concurrency of circuit operation
  - Similar to C
- Trend
  - Designers think in functionality
  - CAD tools take care of implementation

Verilog HDL vs. VHDL

- Verilog HDL结构比较灵活，是一种非常容易掌握的硬件描述语言（类C语言）。
- VHDL语言的高层抽象能力要稍优一些。语言规范十分严谨，甚至于繁琐，但是可读性却十分好。
- 大学、研究机构更多使用VHDL，而工业界更多使用Verilog HDL。

Hardware structures can be modeled effectively in either VHDL and Verilog. Verilog is similar to c and a bit easier to learn.

VHDL(Very High Speed Integrated Circuit Hardware Description Language)

Verilog HDL与VHDL描述加法器电路

library ieee;	
Use ieee.std_logic_1164.all;	
Use ieee.std_logic_arith.all;	
entity vadd is	module kadd (a,b,c,s);
port (a,b: in std_logic_vector(7 downto 0);	input[7:0] a ,b;
c: in std_logic_vector(0 to 0);	input c;
s : out std_logic_vector(8 downto 0));	output[8:0] s;
End vadd;	
architecture rtl of vadd is	
begin	
s <= unsigned(a)+unsigned(b)+unsigned(c);	assign s = a+b+c;
End rtl;	endmodule

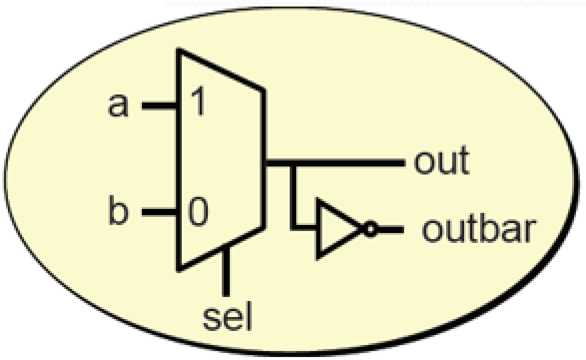
Verilog与C语言

C	Verilog
sub-function	module、 function 、 task
if、 else if、 else	if、 else if、 else
case	case
{,}	begin、 end
for	for
while	while
break	disable
define	define
int	int
printf	monitor、 display 、 strobe

C	Verilog	功　能
>=	>=	大于等于
<=	<=	小于等于
==	==	等于
!=	!=	不等于
～	～	位反相
&	&	按位逻辑与
|	|	按位逻辑或
^	^	
～^	～^	
>>	>>	
<<	<<	
？ ：	？ ：	

C	Verilog	功　能
*	*	乘
/	/	除
+	+	加
-	-	减
%	%	取模
!	!	反逻辑
&&	&&	逻辑与
||	||	逻辑或
>	>	大于
<	<	小于

代码描述方式

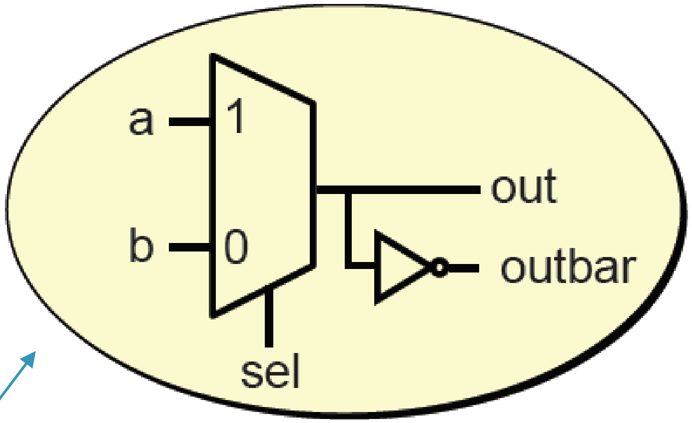


- 结构描述（画电路）
- 数据流描述（写逻辑函数表达式）
- 行为描述

Dataflow Description

- Verilog designs consist of interconnected modules.
- A module can be an element or collection of lower level design blocks.
- A simple module with combinational logic might look like this:

定义外部接口



$$Out = sel \bullet a + sel' \bullet b$$

2-to-1 multiplexer with inverted output

```
module mux_2_to_1(a, b, out,
                  outbar, sel);
```

Declare and name a module; list its ports. Don't forget that semicolon.

```
// This is 2:1 multiplexor
```

Comment starts with //
Verilog skips from // to end of the line

```
  input a, b, sel;
  output out, outbar;
```

Specify each port as input, output, or inout

```
  assign out = sel ? a : b;
  assign outbar = ~out;
```

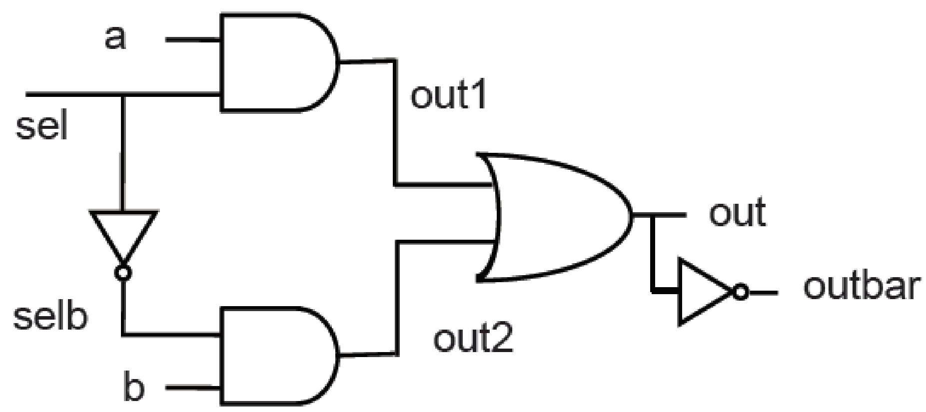
Express the module's behavior. Each statement executes in parallel; order does not matter.

```
endmodule
```

Conclude the module code.

Structure Description

```
module muxgate (a, b, out,
outbar, sel);
input a, b, sel;
output out, outbar;
wire out1, out2, selb;
and a1 (out1, a, sel);
not i1 (selb, sel);
and a2 (out2, b , selb);
or o1 (out, out1, out2);
assign outbar = ~out;
endmodule
```



- Verilog supports basic logic gates as primitives
  - `and, nand, or, nor, xor, xnor, not, buf`
  - can be extended to multiple inputs: e.g., `nand` nand3in (out, in1, in2,in3);
  - `bufif1` and `bufif0` are tri-state buffers
- Net represents connections between hardware elements. Nets are declared with the keyword `wire`.

Behavior Description

- Procedural assignment allows an alternative, often higher-level, behavioral description of combinational logic
- Two structured procedure statements: `initial` and `always`
- Supports richer, C-like control structures such as `if`, `for`, `while`, `case`

```
module mux_2_to_1(a, b, out,
                  outbar, sel);
  input a, b, sel;
  output out, outbar;
```

Exactly the same as before.

```
  reg out, outbar;
```

Anything assigned in an `always` block must *also* be declared as type `reg`

```
  always @ (a or b or sel)
```

Conceptually, the `always` block runs whenever a signal in the sensitivity list changes value

```
  begin
    if (sel) out = a;
    else out = b;

    outbar = ~out;
```

Statements within the `always` block are executed sequentially. Order matters!

```
  end
```

Surround multiple statements in a single `always` block with `begin/end`.

```
endmodule
```

The `case` Statement

- `case` and `if` may be used interchangeably to implement conditional execution within `always` blocks
- `case` is easier to read than a long string of `if...else` statements

```
module mux_2_to_1(a, b, out,
                  outbar, sel);
  input a, b, sel;
  output out, outbar;
  reg out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

  assign outbar = ~out;

endmodule
```

```
module mux_2_to_1(a, b, out,
                  outbar, sel);
  input a, b, sel;
  output out, outbar;
  reg out;

  always @ (a or b or sel)
  begin
    case (sel)
      1'b1: out = a;
      1'b0: out = b;
    endcase
  end

  assign outbar = ~out;

endmodule
```

Note: Number specification notation: <size>'<base><number>
(4'b1010 if a 4-bit binary value, 16'h6cda is a 16 bit hex number, and 8'd40 is an 8-bit decimal value)

Module Structure

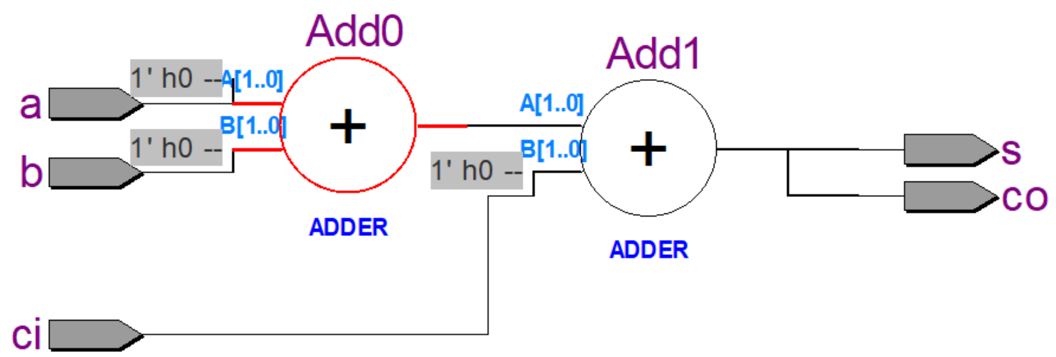
```
module M (P1, P2, P3, P4);
  input P1, P2;
  output [7:0] P3;
  inout P4;

  reg [7:0] R1, M1[1:1024];
  wire W1, W2, W3, W4;
  parameter C1 = "This is a string";

  initial
  begin : BlockName
    // Statements
  end

  always
  begin
    // Statements
  end

  // Continuous assignments...
  assign W1 = Expression;

  // Module instances...
  COMP U1 (W3, W4);
  COMP U2 (.P1(W3), .P2(W4));

endmodule
```

代码结构

- 端口定义
- 变量定义
- 并行执行语句
  - 过程语句
  - 连续赋值语句
  - 模块实例化语句
- 顺序执行语句

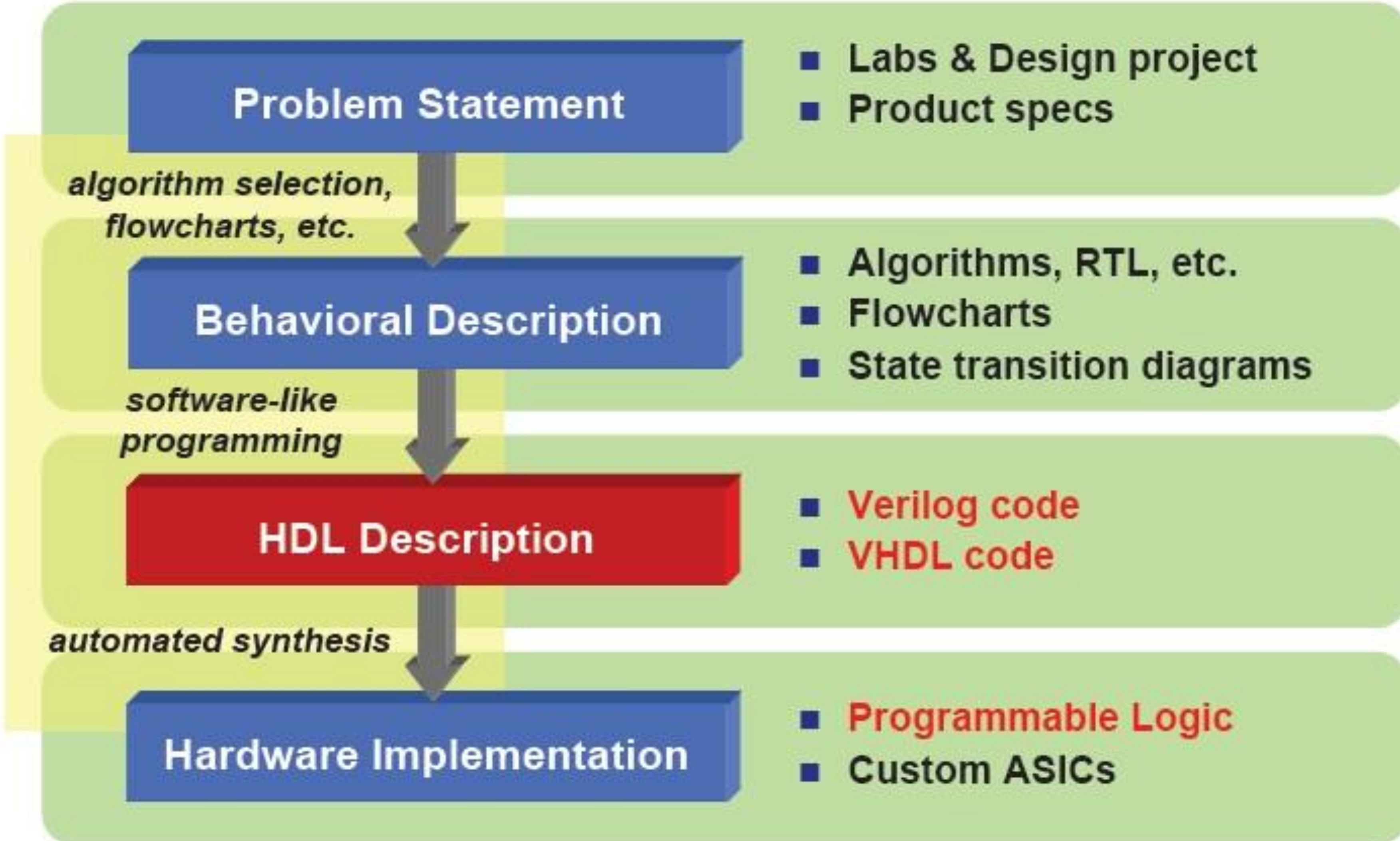
调试

- 功能或时序仿真(Modelsim或quartus波形文件或测试文件)
- 利用板子外设、信号发生器、示波器
- RTL Viewer



Tools->Netlist Viewers->RTL Viewer

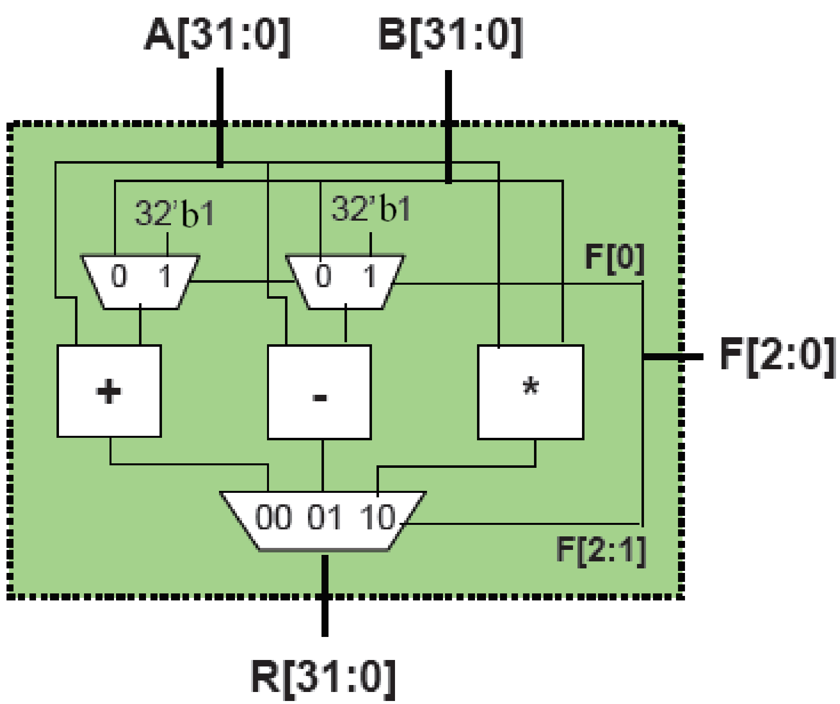
四、基于FPGA的组合逻辑电路实现



自顶向下的设计

- Modularity is essential to the success of large designs
- A Verilog `module` may contain submodules that are "wired together"
- High-level primitives enable direct synthesis of behavioral descriptions (functions such as additions, subtractions, shifts (<< and >>), etc.

Example: A 32-bit ALU



Function Table

F2 F1 F0	Function
0 0 0	A + B
0 0 1	A + 1
0 1 0	A - B
0 1 1	A - 1
1 0 X	A * B

自底向上的实现

2-to-1 MUX

```
module mux32two(i0,i1,sel,out);
input [31:0] i0,i1;
input sel;
output [31:0] out;

assign out = sel ? i1 : i0;

endmodule
```

3-to-1 MUX

```
module mux32three(i0,i1,i2,sel,out);
input [31:0] i0,i1,i2;
input [1:0] sel;
output [31:0] out;
reg [31:0] out;

always @ (i0 or i1 or i2 or sel)
begin
  case (sel)
    2'b00: out = i0;
    2'b01: out = i1;
    2'b10: out = i2;
    default: out = 32'bx;
  endcase
end
endmodule
```

32-bit Adder

```
module add32(i0,i1,sum);
input [31:0] i0,i1;
output [31:0] sum;

assign sum = i0 + i1;

endmodule
```

32-bit Subtracter

```
module sub32(i0,i1,diff);
input [31:0] i0,i1;
output [31:0] diff;

assign diff = i0 - i1;

endmodule
```

16-bit Multiplier

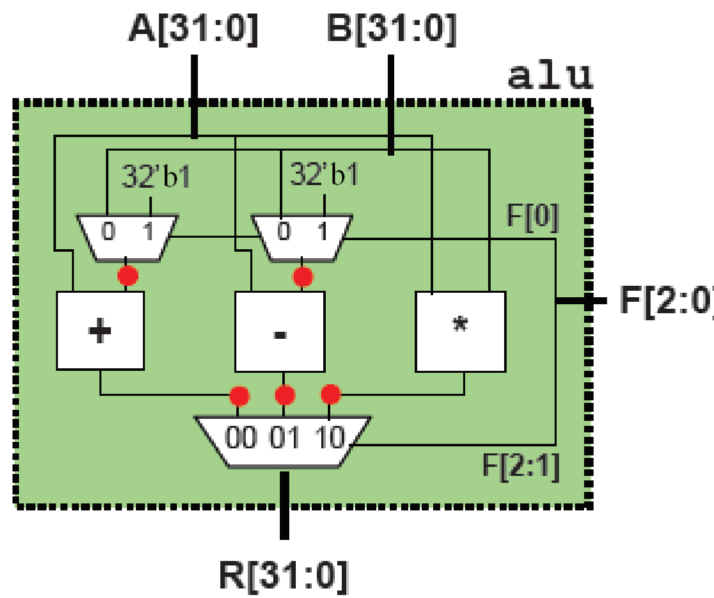
```
module mul16(i0,i1,prod);
input [15:0] i0,i1;
output [31:0] prod;

// this is a magnitude multiplier
// signed arithmetic later
assign prod = i0 * i1;

endmodule
```

自底向上的实现

- **Given submodules:**

```
module mux32two(i0,i1,sel,out);
module mux32three(i0,i1,i2,sel,out);
module add32(i0,i1,sum);
module sub32(i0,i1,diff);
module mul16(i0,i1,prod);
```

- **Declaration of the ALU Module:**




```
module alu(a, b, f, r);
  input [31:0] a, b;
  input [2:0] f;
  output [31:0] r;

  wire [31:0] addmux_out, submux_out;
  wire [31:0] add_out, sub_out, mul_out;

  mux32two   adder_mux(b, 32'b1, f[0], addmux_out);
  mux32two   sub_mux(b, 32'b1, f[0], submux_out);
  add32      our_adder(a, addmux_out, add_out);
  sub32      our_subtracter(a, submux_out, sub_out);
  mul16      our_multiplier(a[15:0], b[15:0], mul_out);
  mux32three output_mux(add_out, sub_out, mul_out, f[2:1], r);

endmodule
```

intermediate output nodes

module names

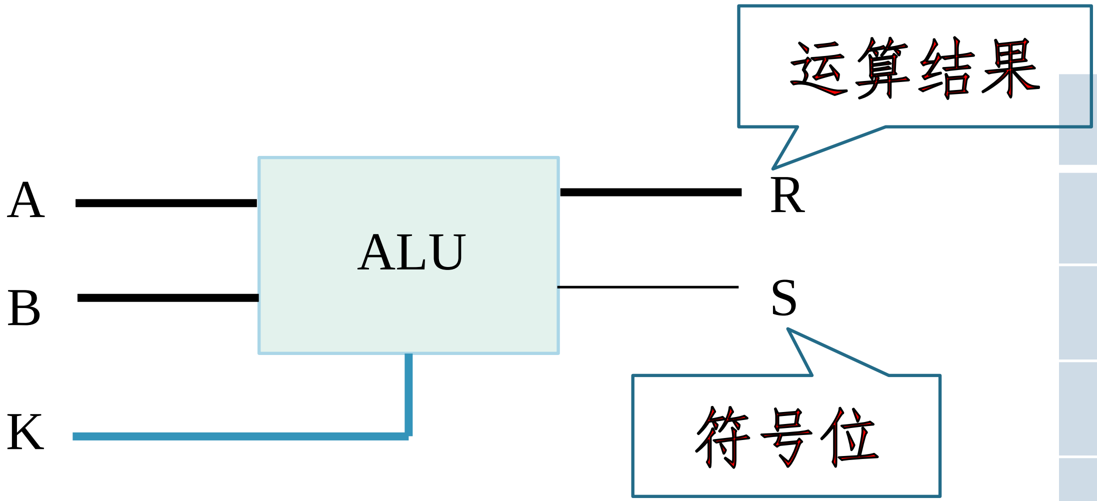
(unique) instance names

corresponding wires/regs in module alu

五、EDA实验一内容布置

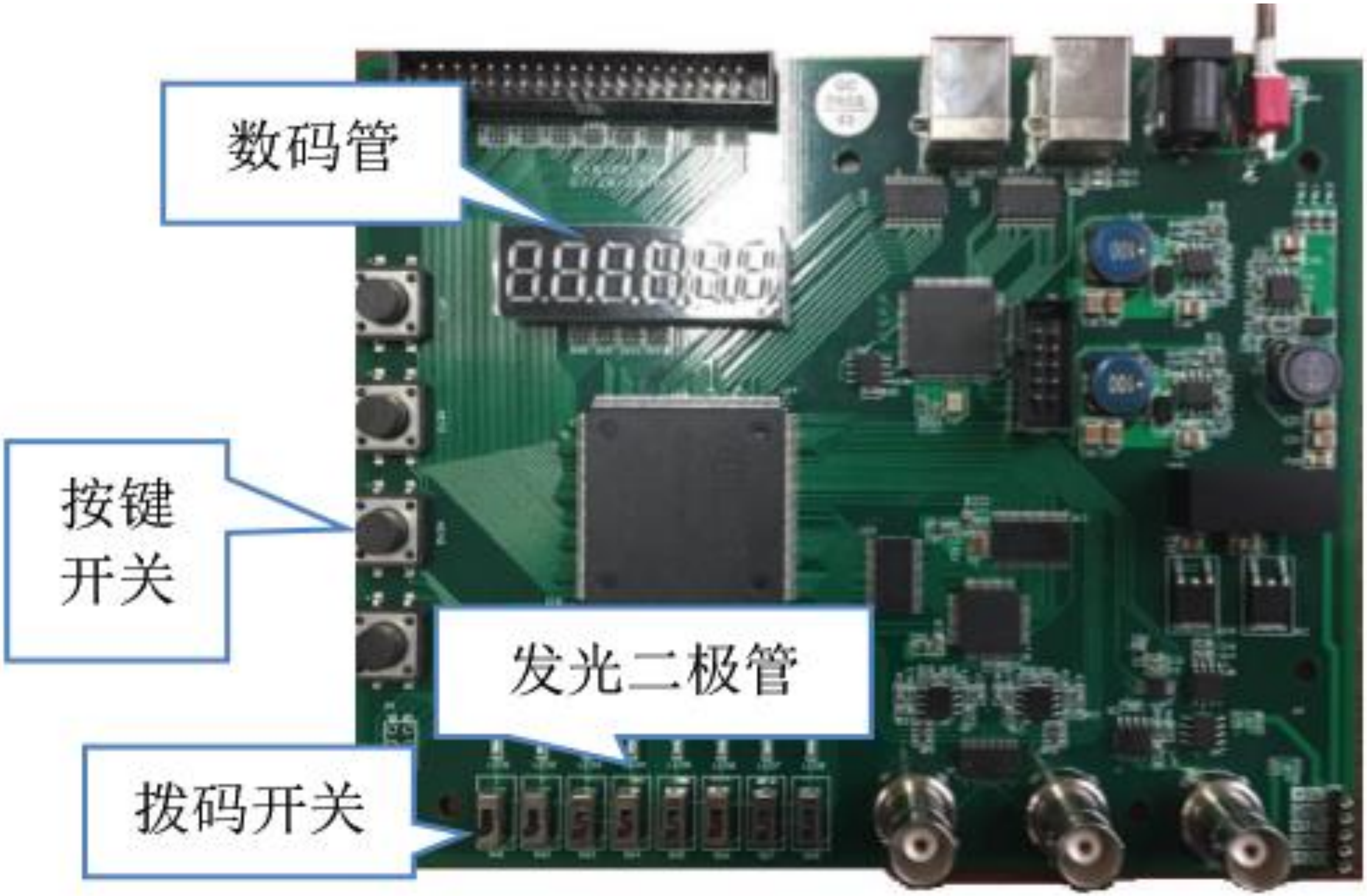
基本内容

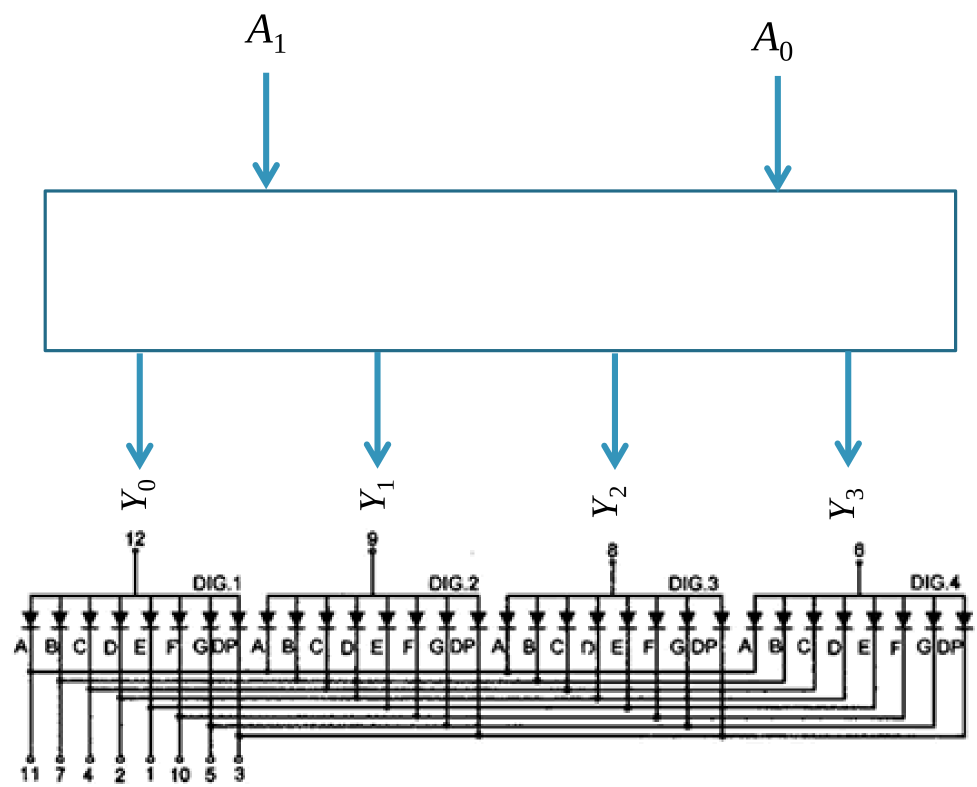
基于FPGA实现一个简易计算器：

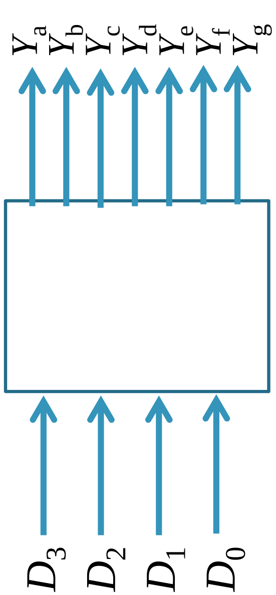


K	R
00	0
01	A+B
10	A-B
11	A*B

其中A和B的取值范围为0～15；用实验板上的拨码开关和按键开关模拟输入；在数码管上以十六进制形式显示运算数和运算结果，负号用发光二极管显示。

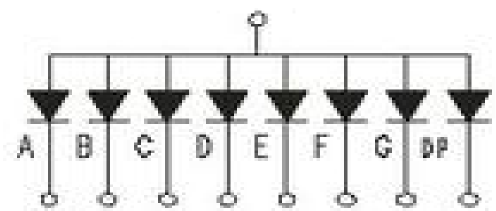

数码管
按键
开关
发光二极管
拨码开关

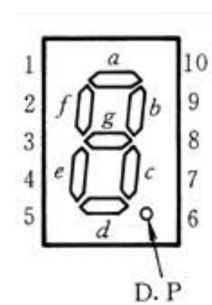




EDA实验一中用两个按键开关控制数码管的显示

EDA实验二中将学习循环扫描显示





六、EDA实验二内容展望

Vending machine

利用实验板上的拨码开关和按键开关模拟投币、购物和退币输入，用发光二极管模拟各种提示信息，用数码管显示余额，实现一个自动售货机内部控制电路。要求满足如下规格:

1）可接受5角、1元和5元的投币，每次购买允许投入多种不同币值的钱币；用3只数码管显示当前投币金额，如055表示已投币5.5元；

2）可售出价格分别为1.5元和2.5元的商品，假设用户每次购买时只选择单件、一种商品；允许用户多次购买商品，每次购买后，可以进行补充投币；

3）选择购买商品后，如果投币金额不足，则提醒；否则，售出相应的商品，并提醒用户取走商品；

4）若用户选择退币，则退回余下的钱，并提醒用户取钱。