《数字逻辑》
Digital Logic

Verilog介绍（1）

北京工业大学软件学院
王晓懿

Verilog是什么？

- 硬件描述语言 VS 编程语言
  - 并行结构 VS 顺序执行的语句
- 使用Verilog语言的子集
  - 可综合语句 VS 仿真语句

Verilog数据类型： Bit-vector

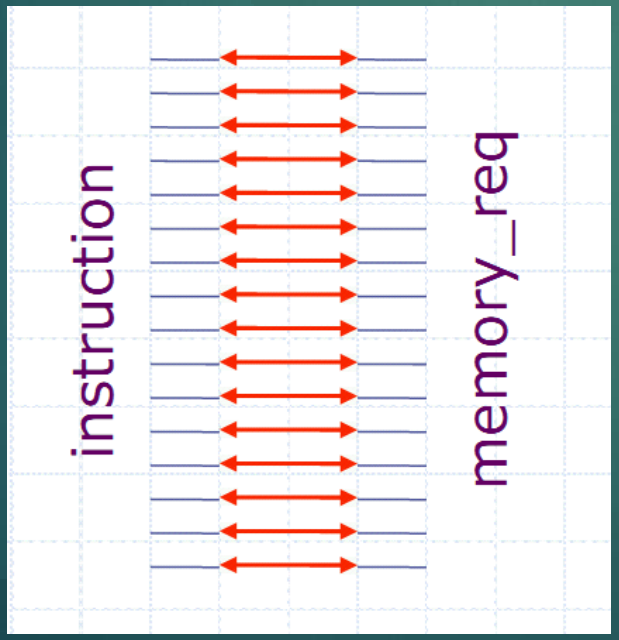
取值	含义
0	逻辑0
1	逻辑1
X	未知值
Z	高阻态

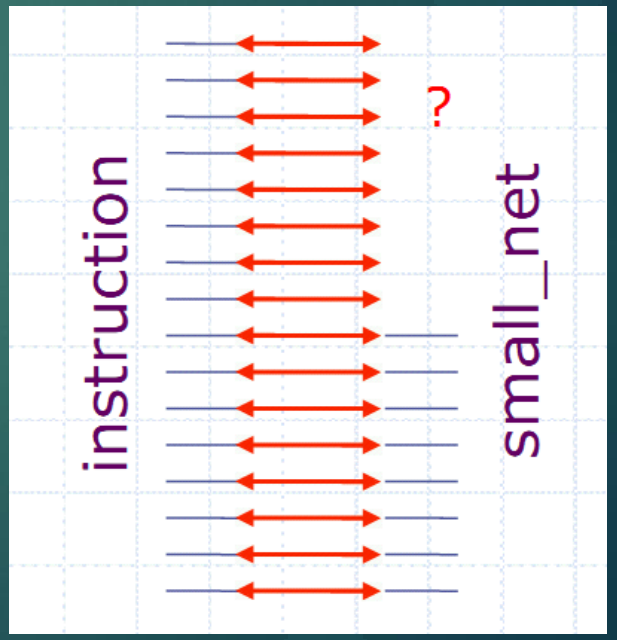
- 未知值表示可能为0,1，Z或正在变化中间状态

使用wire描述线网

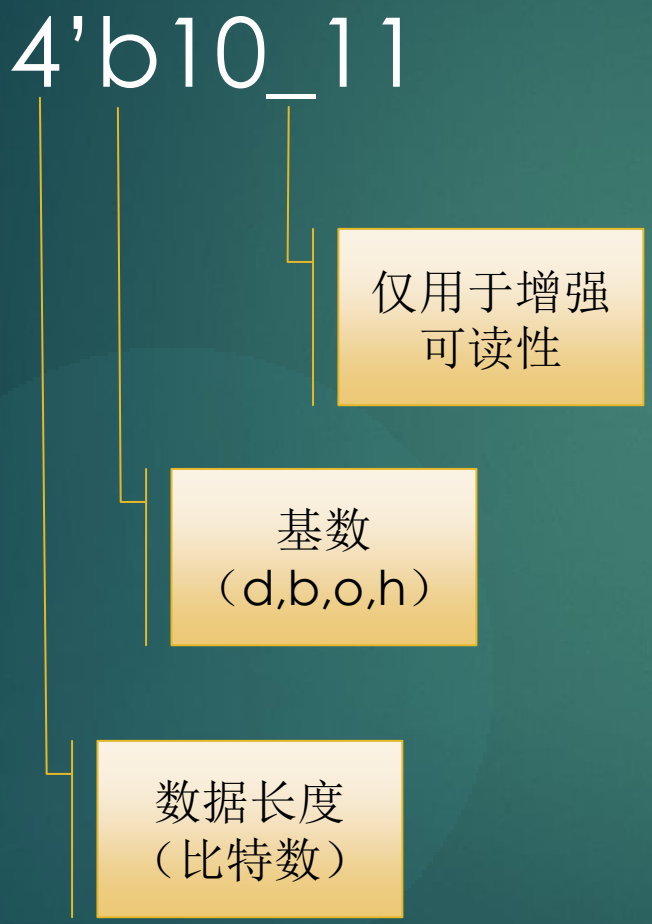
```
wire [15:0] instruction;
wire [15:0] memory_req;
wire [ 7:0] small_net;
```

- 无法检查线网数据类型是否匹配！





常量



- 二进制“字符串”
  - 8’b0000_0000
  - 8’b0xx0_1xx1
- 十六进制“字符串”
  - 32’h0a34_def1
  - 16’haxxx
- 十进制“字符串”
  - 32’d42

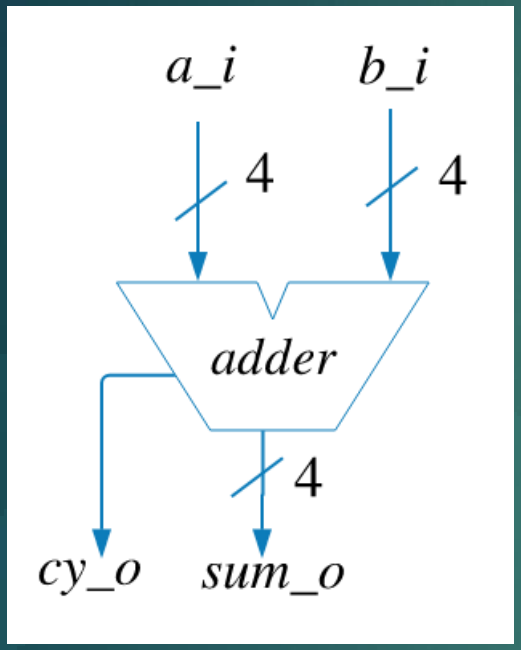
采用的Verilog语言子集

- 用于仿真的语句
- 可综合的语句
- 编码规范
- 注释文档

常用Verilog程序类型

- 描述门结构的Verilog程序（Structural Verilog）

Verilog模块声明

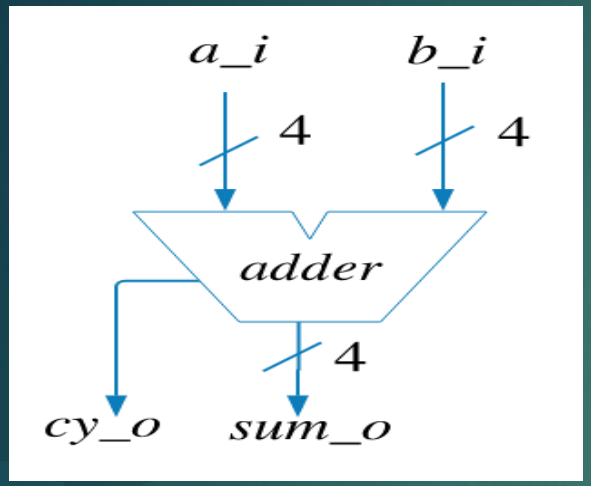


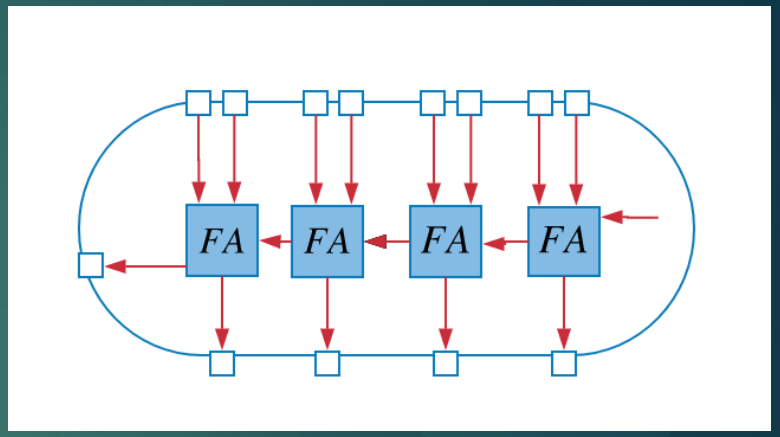

```
module adder( input  [3:0] a_i,
              input  [3:0] b_i,
              output       cy_o,
              output [3:0] sum_o );
    // HDL对加法器的功能描述
endmodule
```

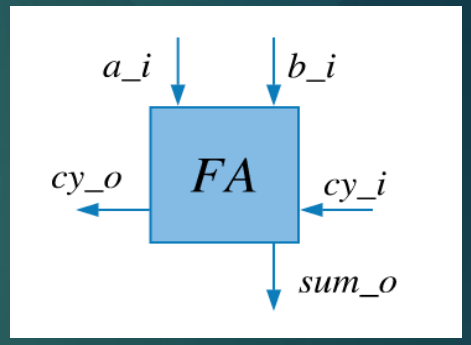
端口需要指定信号位数以及是输入还是输出

注意声明最后的分号

模块中使用其他模块的实例



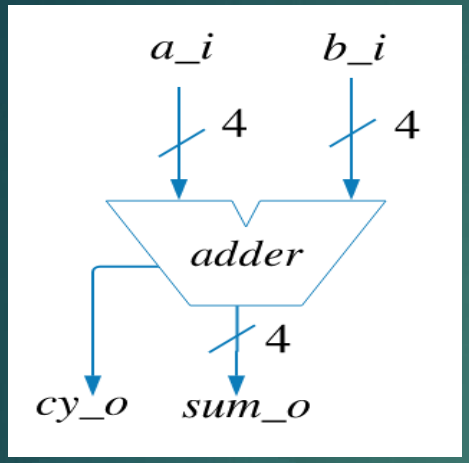


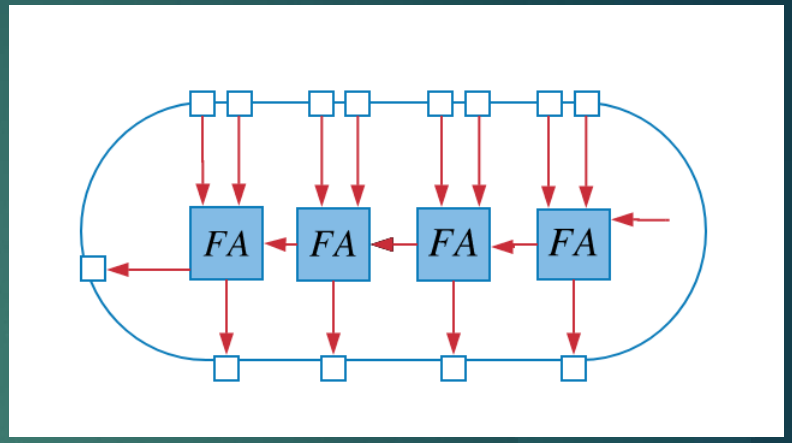



```
module adder( input   [3:0] a_i,
              input   [3:0] b_i,
              output        cy_o,
              output  [3:0] sum_o );
    wire c0, c1, c2;
    FA fa0( ... );
    FA fa1( ... );
    FA fa2( ... );
    FA fa3( ... );
endmodule
```

```
module FA( input a_i, b_i, cy_i,
           output cy_o, sum_o );
    // 1位全加器的HDL描述
endmodule
```

连接使用模块的实例






```
module adder( input  [3:0] a_i,
              input  [3:0] b_i,
              output       cy_o,
              output [3:0] sum_o );
    wire c0, c1, c2;
    FA fa0( a_i[0], b_i[0], 1'b0, c0,    sum_o[0] );
    FA fa1( a_i[1], b_i[1], c0,   c1,    sum_o[1] );
    FA fa2( a_i[2], b_i[2], c1,   c2,    sum_o[2] );
    FA fa3( a_i[3], b_i[3], c2,   cy_o,  sum_o[3] );

endmodule
```

进位链

形参与实参

- 按位置匹配形参与实参

```
module FA( input a_i, b_i, cy_i,
           output cy_o, sum_o );
    // 1位全加器的HDL描述
endmodule
```

```
FA fa0 ( a_i[0], b_i[0], 1'b0, c0, sum_o[0] );
```

- 按名称匹配形参与实参

```
FA fa0 ( .a_i (a_i[0])
        ,.b_i (b_i[0])
        ,.cy_i(1'b0)
        ,.cy_o(c0)
        ,.sum_o(sum_o[0])
       );
```

提示：
虽然在PPT中为了节省空间采用了按位置匹配参数的方法，在实际代码采用按名称匹配参数更不容易出错。

如何实现全加器？使用原语器件

- 内置的原语器件

  and, nand, or, nor, xor, xnor, not, buf

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

- 可以通过真值表自定义原语器件

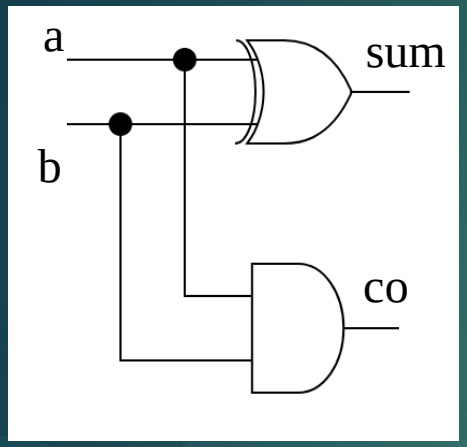
使用自定义原语实现全加器

```
primitive AddCarry(cy_o, cy_i,
                   a_i, b_i);
  output cy_o;
  input cy_i,a_i,b_i;

table
// cy_i a_i  b_i  : cy_o
    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;
endtable
```

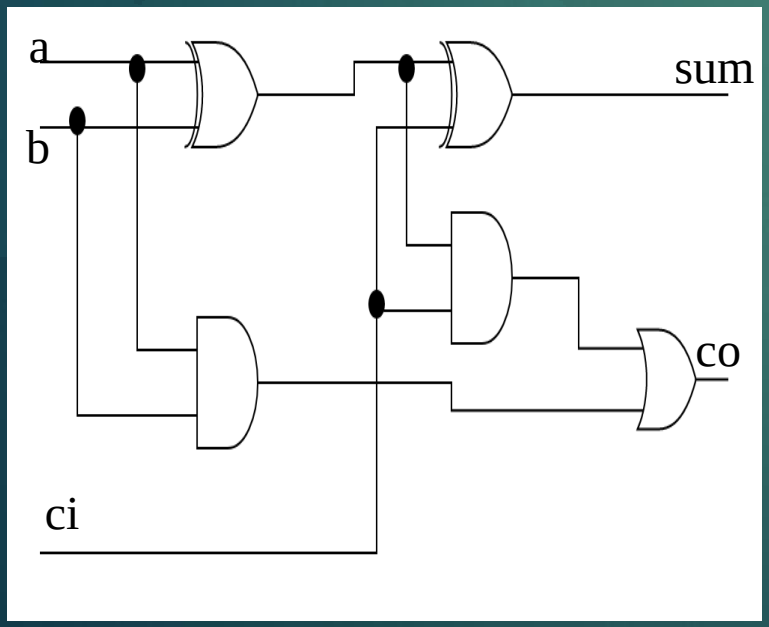
```
primitive AddSum(sum_o, cy_i,
                 a_i, b_i);
  output sum_o;
  input cy_i,a_i,b_i;

table
// cy_i a_i  b_i  : sum_o
    0 0    0    : 0 ;
    0 0    1    : 1;
    0 1    0    : 1;
    0 1    1    : 0;
    1 0    0    : 1;
    1 0    1    : 0;
    1 1    0    : 0;
    1 1    1    : 1;
endtable
```

```
module FA( input a_i, b_i, cy_i,
           output cy_o, sum_o );

    AddSum s(sum_o, cy_i, a_i, b_i);
    AddCarry c(cy_o, cy_i, a_i, b_i);
endmodule
```

使用门电路实现全加器




```
module HA( output cy_o, output sum_o,
           input a_i, input b_i);
    xor(sum_o, a_i, b_i);
    and(cy_o,  a_i, b_i);
endmodule
```




```
module FA( output cy_o, output sum_o,
     input a_i, input b_i, input cy_i);
  wire s0, c1, c2;
  // 第一个半加器
  HA ha0 (c1, s0, a_i, b_i);
  // 第二个半加器
  HA ha1 (c2, sum_o, s0, cy_i);
  or g0 (cy_o, c1, c2);
endmodule
```

常用Verilog程序类型

- 描述数据流的Verilog程序（Dataflow Verilog）

模块的接口与实现

- 接口：
  - 模块声明中的输入与输出Port
  - 模块文档中声明的功能
- 原则：
  - 无论模块内部如何实现，不能影响外部对模块的使用
  - 以4选1数据选择器为例

赋值语句（assignments）

```
module mux4( input  a_i, b_i, c_i, d_i,
             input [1:0] sel_i,
             output z_o );

wire t0, t1;

assign z_o = ~((t0 | sel_i[0]) & (t1 | ~sel_i[0]));
assign t1  = ~((sel_i[1] & d_i) | (~sel_i[1] & b_i));
assgin t0  = ~((sel_i[1] & c_i) | (~sel_i[1] & a_i));

endmodule
```

Verilog语言定义的操作符

赋值语句的先后顺序不影响结果！因为它们是并行执行的。

每当输入信号变化时，赋值语句就会被执行（正如你对组合逻辑所期望的）。注意不要写出死循环。

使用？：语句实现4输入输入选择器

```
module mux4( input  a_i, b_i, c_i, d_i,
             input [1:0] sel_i,
             output z_o );

assign z_o = ( sel_i == 0 ) ? a_i :
             ( sel_i == 1 ) ? b_i :
             ( sel_i == 2 ) ? c_i :
             ( sel_i == 3 ) ? d_i : 1'bx;

endmodule
```

当sel_i未定义时，可以将未知值进行传播，使得输出也变为红色显示的未定义值

Verilog中常用运算符

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		

常用Verilog程序类型

- 描述行为的Verilog程序（RTL Verilog）

使用“always”语句

```
module mux4( input  a_i, b_i, c_i, d_i,
             input [1:0] sel_i,
             output reg z_o );
    reg t0, t1;

    always_comb  // 等价于 always @(*)
    begin
      t0  =   (sel_i[1] & c_i) | (~sel_i[1] & a_i);
      t1  = ~((sel_i[1] & d_i) | (~sel_i[1] & b_i));
      t0  = ~t0;
      z_o = ~((t0 | sel_i[0])  & (t1 | ~sel_i[0]) );
    end

endmodule
```

注意！
在always语句的begin/end代码块中，可以认为语句是**顺序执行**的。
同一变量的最后一个赋值语句为输出时变量的返回值。
输入变量变化后经过延迟输出变量随之变化。

在“always”语句中使用更多的控制语句

```
module mux4( input  a_i, b_i, c_i, d_i,
             input [1:0] sel_i,
             output reg z_o );

    always @(*)
    begin
      if (sel_i == 2'd0 )
        z_o = a_i;
      else if (sel_i == 2'd1 )
        z_o = b_i;
      else if (sel_i == 2'd2 )
        z_o = c_i;
      else if (sel_i == 2'd3 )
        z_o = d_i;
      else
        z_o = 1'bx;
    end
```

```
always @(*)
    begin
      case ( sel_i )
        2'd0 : z_o = a_i;
        2'd1 : z_o = b_i;
        2'd2 : z_o = c_i;
        2'd3 : z_o = d_i;
        default : z_o = 1'bx;
      endcase
end
```

情况列举不全的case语句？

```
module mux3( input  a_i, b_i, c_i,
             input [1:0] sel_i,
             output reg z_o );

always @(*)
  begin
    case ( sel_i )
      2'd0 : z_o = a_i;
      2'd1 : z_o = b_i;
      2'd2 : z_o = c_i;
    endcase
  end

endmodule
```

如果sel_i = 3, 数据选择器输出保持为前一次输出的值！

还是组合逻辑吗？

情况列举不全的case语句？

```
module mux3( input  a_i, b_i, c_i,
             input [1:0] sel_i,
             output reg z_o );

always @(*)
  begin
    case ( sel_i )
      2'd0 : z_o = a_i;
      2'd1 : z_o = b_i;
      2'd2 : z_o = c_i;
      default : z_o = 1'bx;
    endcase
  end

endmodule
```

通过default语句避免未赋值的case

参数化的模块

默认值

```
module mux4 #( parameter WIDTH = 1 )
             ( input  [WIDTH-1:0] a_i, b_i, c_i, d_i,
               input  [1:0] sel_i,
               output [WIDTH-1:0] z_o );

  wire [WIDTH-1:0] t0, t1;

  assign t0  = (sel_i[1] ? c_i : a_i);
  assign t1  = (sel_i[1] ? d_i : b_i);
  assign z_o = (sel_i[0] ? t0  : t1);
endmodule
```

实例化

```
mux4#(32) alu_mux
( .a_i (op1),
  .b_i (op2),
  .c_i (op3),
  .d_i (op4),
  .sel_i(alu_mux_sel),
  .z_o(alu_mux_out) );
```

参数化的模块有利于模块复用

采用何种Verilog描述？

- 无论用何种方式描述，对外提供接口要一致
- 可以借助工具综合、转化描述
- 需要理解掌握底层描述方式

加法器数据流描述

```
// Dataflow description of four-bit adder

// Verilog 2001, 2005 module port syntax

module binary_adder (
  output [3: 0]          Sum,
  output                 C_out,
  input [3: 0]           A, B,
  input                  C_in
);

  assign {C_out, Sum} = A + B + C_in;
endmodule
```

- 编写Verilog测试程序（Testbench）

Testbench I

```
‘timescale 1 ns / 1 ns
module testbed();
  reg [3:0] y; reg [3:0] x;
  wire c_out; wire [3:0]sum;

  adder A0(c_out, sum, x, y);
initial
  begin
    $dumpfile("dump.vcd");
    $dumpvars;
    $display ("display info here");
    $monitor ("%g\t %b %b %b %b", $time,
              x,y,sum, c_out);
  end
```

时间单位/精度

无输入输出

实例化被测模块

输出波形文件

输出调试信息

Testbench II

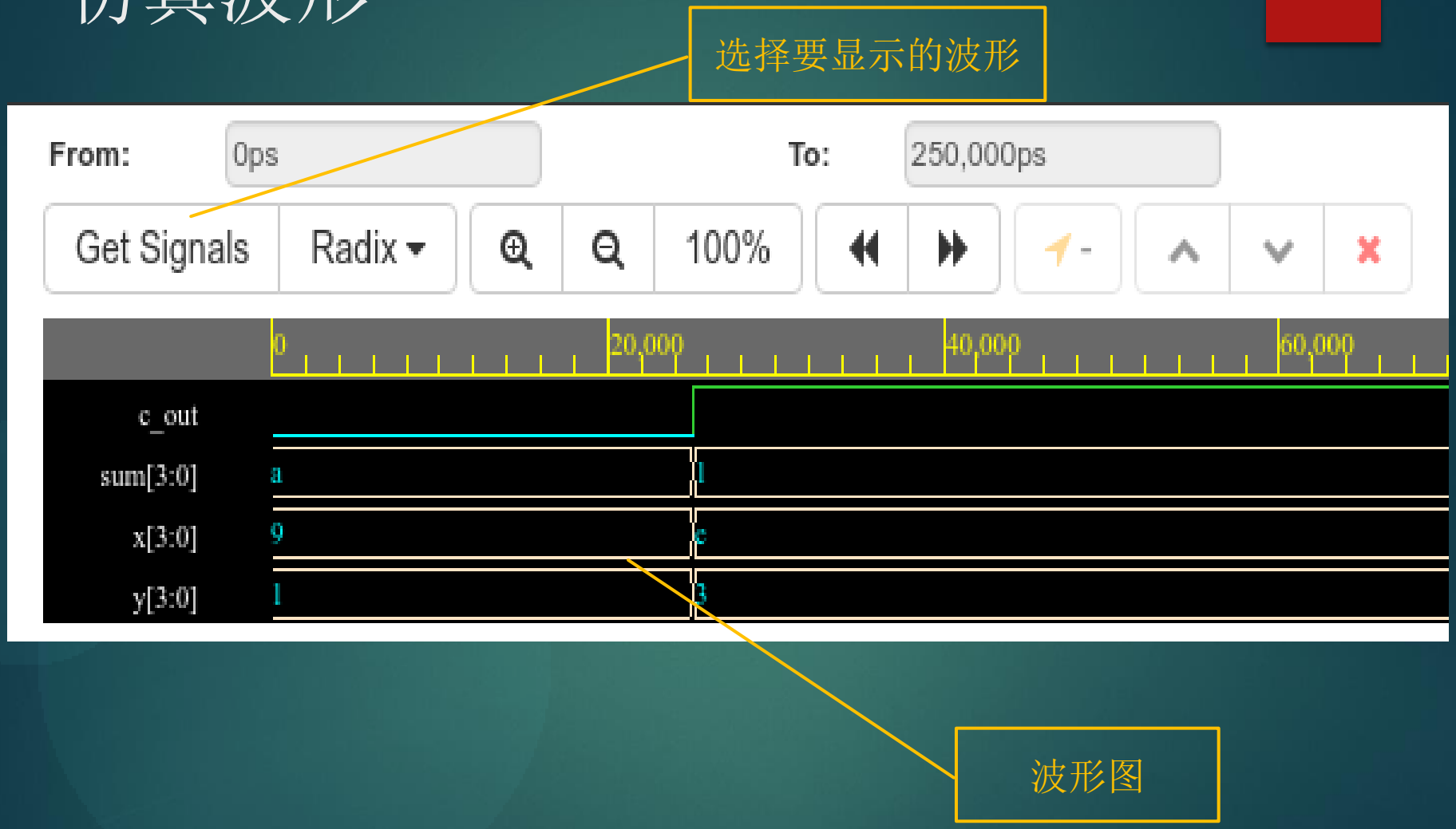
```
initial
  begin    //SIGNAL x
         x = 4'b1001;
       #25000
         x = 4'b1110;
       #25000 ;
  end
initial
  begin    //SIGNAL y
         y = 4'b0001;
       #25000
         y = 4'b0011;
       #25000 ;
  end

initial #250000 $finish;
endmodule
```

使用initial块生成输入信号

延时

设置仿真结束时间

仿真波形


选择要显示的波形
From:
0ps
To:
250,000ps
Get Signals
Radix
100%
0
20,000
40,000
60,000
c_out
sum[3:0]
x[3:0]
y[3:0]
a
1
9
c
1
3
波形图

使用仿真器验证逻辑设计

- EDA Playground
  - https://www.edaplayground.com/
  - 使用教育邮箱注册

译码器的Verilog描述

二-四译码器

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
  output      [0: 3]          D;
  input                       A, B;
  input                       enable;
  wire                        A_not,B_not, enable_not;

  not
    G1  (A_not, A),
    G2  (B_not, B),
    G3  (enable_not, enable);
  nand
    G4  (D[0], A_not, B_not, enable_not),
    G5  (D[1], A_not, B, enable_not),
    G6  (D[2], A, B_not, enable_not),
    G7  (D[3], A, B, enable_not);
endmodule
```

```
// Dataflow description of two-to-four-line decoder

// See Fig. 4.19. Note: The figure uses symbol E, but the
// Verilog model uses enable to clearly indicate functionality.

module decoder_2x4_df (                    // Verilog 2001, 2005 syntax
  output    [0: 3]          D,
  input                     A, B,
                            enable
);
  assign    D[0] = !((!A) && (!B) && (!enable)),
            D[1] = !(*!A) && B && (!enable)),
            D[2] = !(A && B && (!enable)
            D[3] = !(A && B && (!enable))
endmodule
```

减法器的Verilog描述