# Electric

projects/LC3-tools/**electricBinary.jar**

**(or download directly from StaticFree web site)**

## BASIC ELECTRIC OPERATIONS

**--- Open tutorial.jelib**

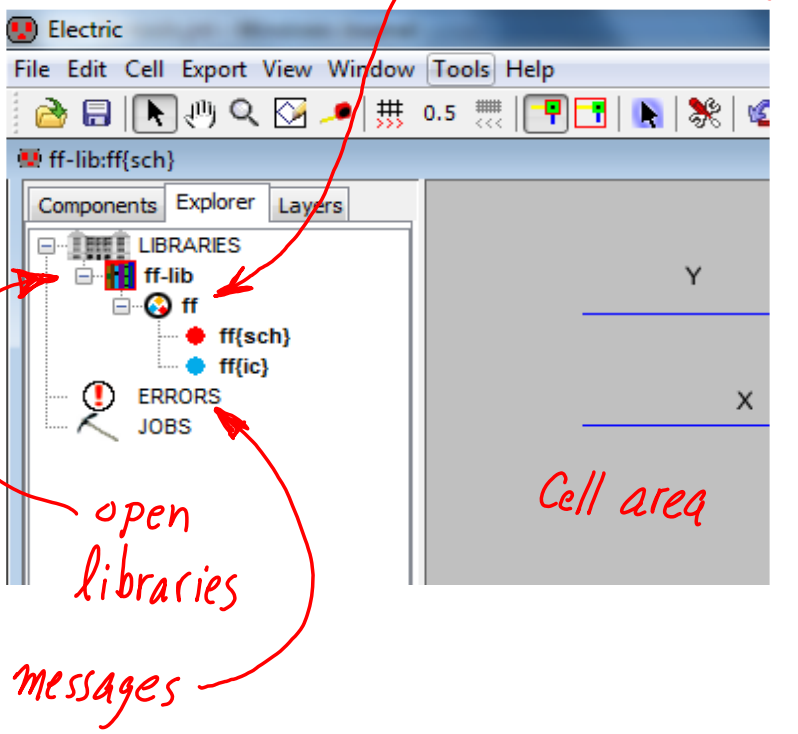
start ElectricBinary.jar (double click)

**^File.OpenLibrary**

**--- See Documentation**

**^Explorer tab**
**^^0AAA-ReadMe{doc}**

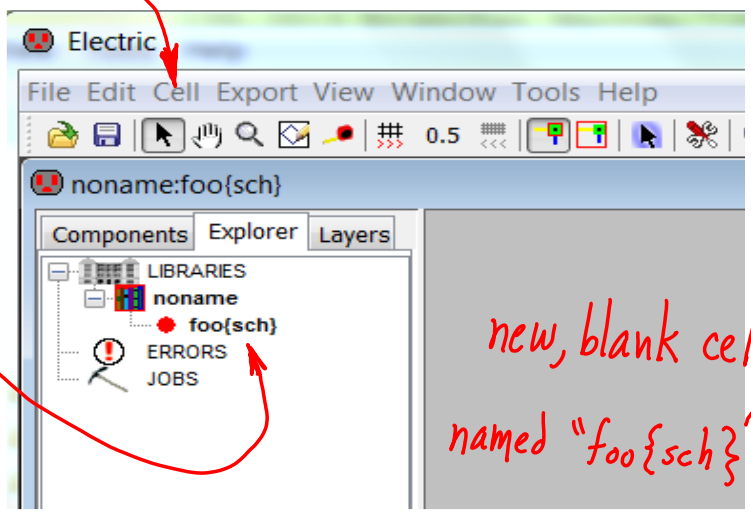
also see text boxes in other schematics




**--- Create a cell**

**^Cell.NewCell**
**Library[ noname ]**
**Name: _____** foo
**Type[ schematic ]**

default, if no libs open




**--- Place objects (gates/wires) in cell**

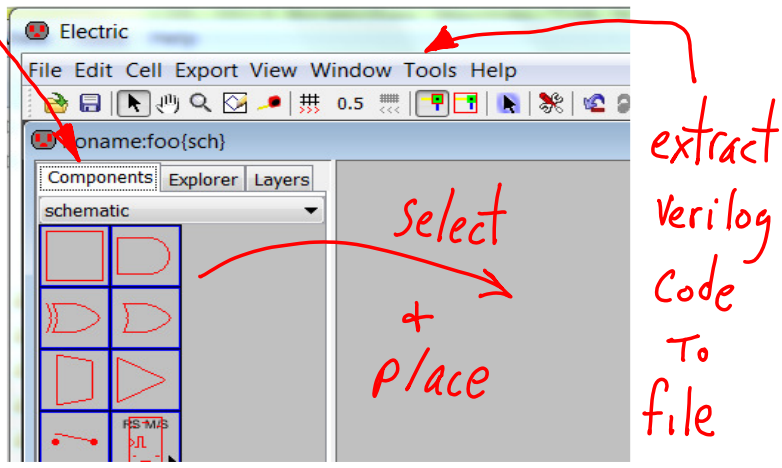
**^Components.Schematic.{OR gate}**

**--- Place verilog code in cell**

**^Components.Schematic.Misc.VerilogCode**
**^Edit.Properties.ObjectProperties**

**--- Extract verilog code**

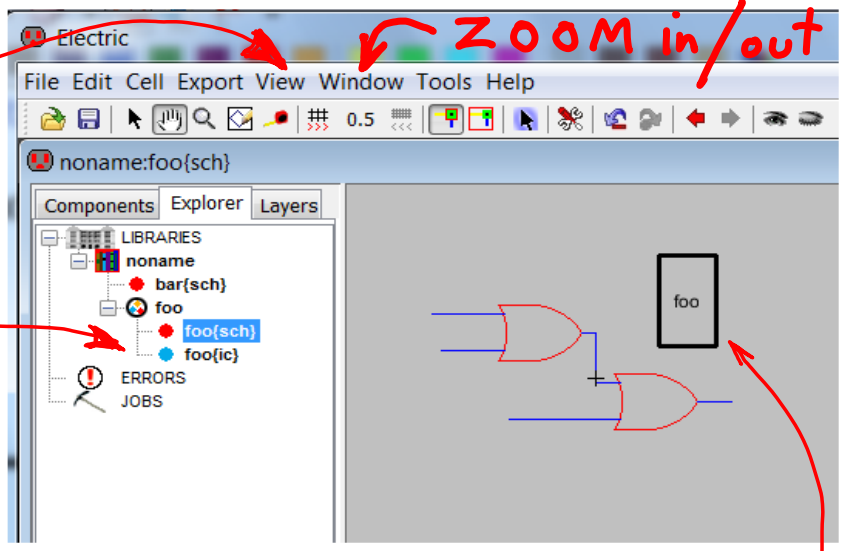
**^Tools.SimulationVerilogDeck)**



# Cell instances

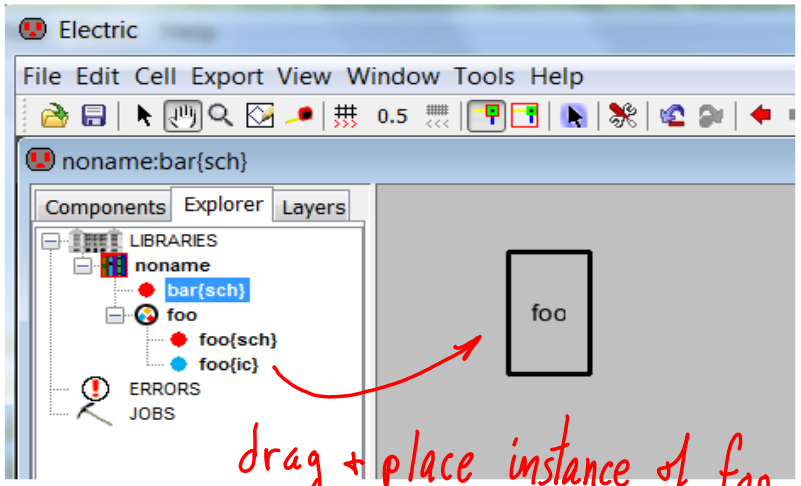
**Create an ICON view**

**^View.MakeIconView**




**Use the icon,**
**place an instance in another cell**

**Hierarchical design**



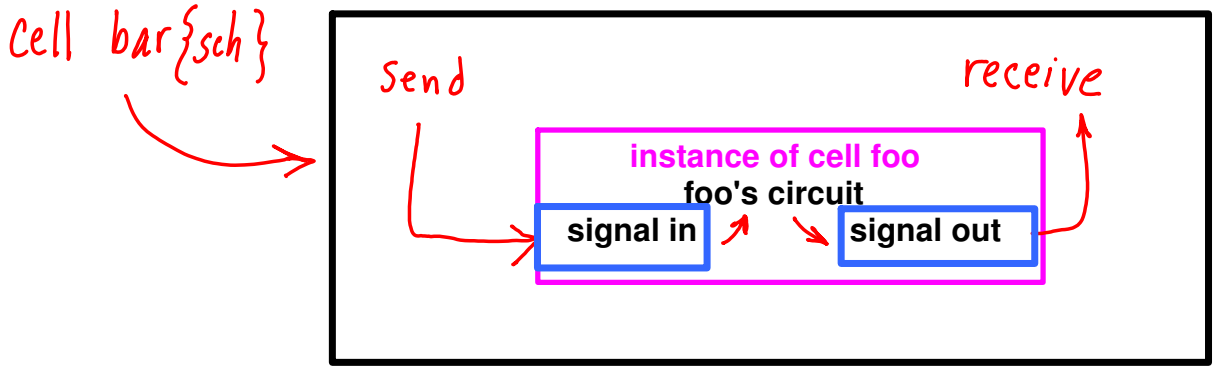

– schematic cell vs. icon cell

**Schematic Cell** — **foo{sch} has circuit design**

**Icon Cell** — **foo{ic} has graphical design**

**Hierarchy:**
**place icon foo{ic} into** cell **bar{sch} ===> creates an instance of foo in circuit**

# We need hierarchical connections



**Make a connection point in foo{sch}**

**Place a pin**

**^Components.{wire pin, blue dot}**

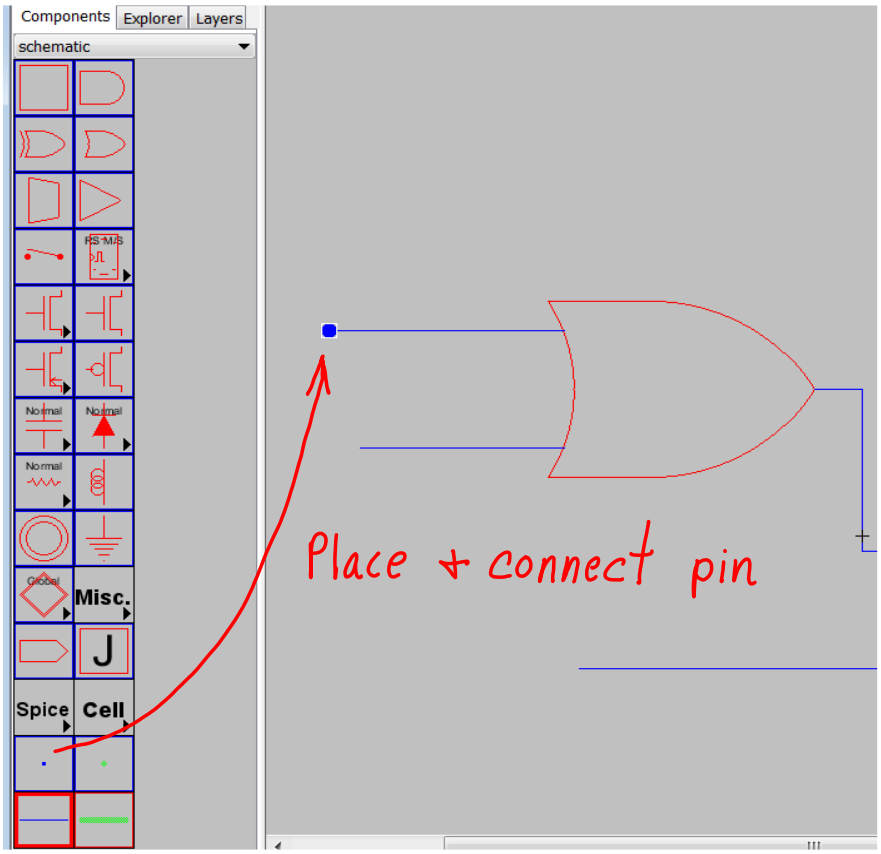
**^{cell area or end of wire}**

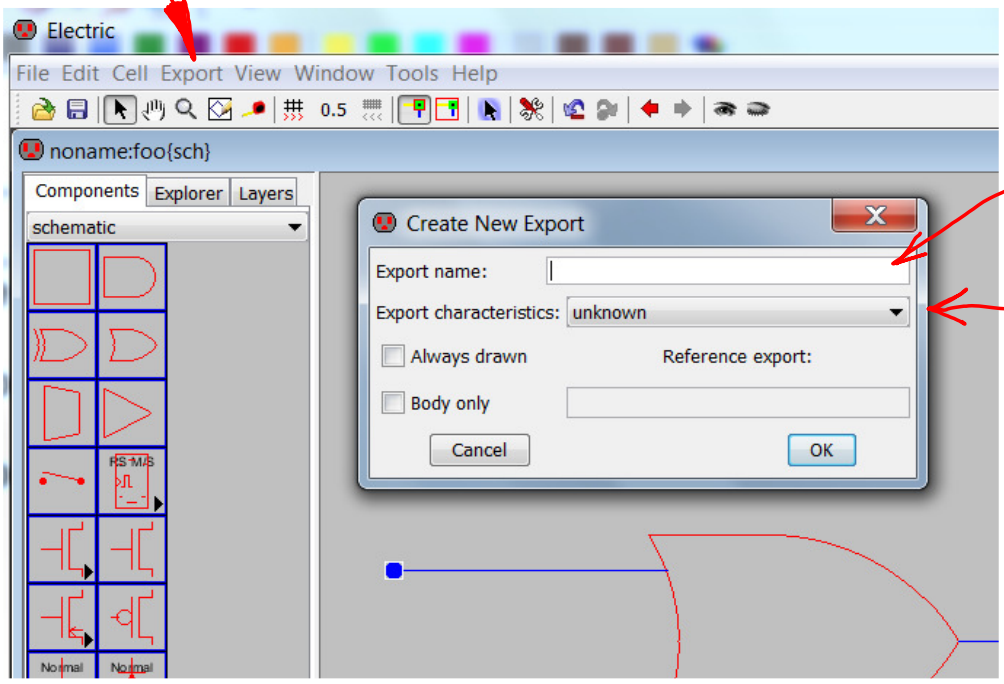
**Select pin**

**^{pin you just placed}**

**Create the "Export"**

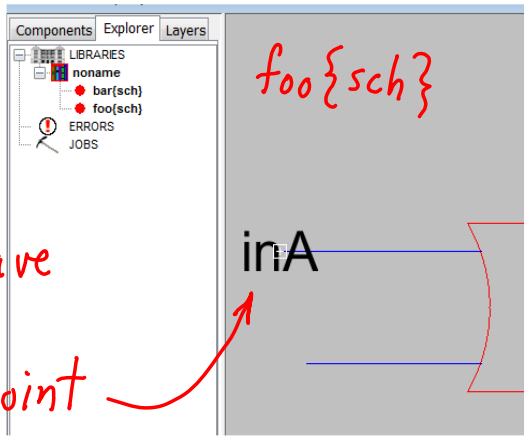
**^Export.CreateExport**



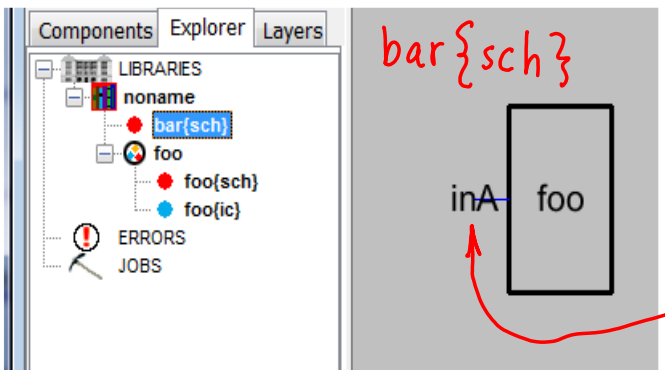




name the export e.g. "inA"

choose its type "input" or "output"



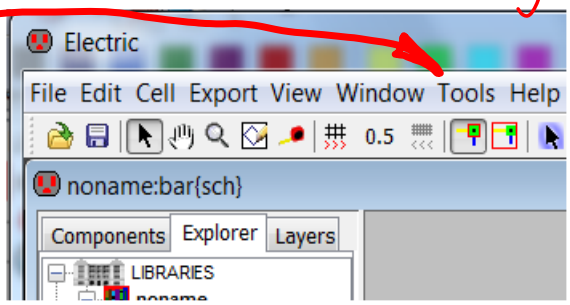

now you have an external connection point




**NB--Need to View.MakeIconView again to show port.**
**Have to delete old icon first.**

# Electric + Verilog

^ Tools. Simulation (Verilog). Make Verilog Deck



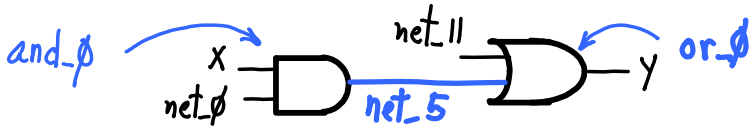

| Electric | | Verilog |
|---|---|---|
| schematic cell | — | module def |
| schematic exports | — | module parameters |
| icon instance | — | module instance |
| wires | — | instances of wire object |
| connect icon's exports | — | instance's actual wire args |

```
/* Verilog for cell 'ff{sch}' from library 'ff-lib' */
/* Created on Fri Jan 18, 2013 11:51:35 */
/* Last revised on Fri Jan 18, 2013 12:12:05 */
/* Written on Fri Jan 18, 2013 12:18:34 by Electric VLSI Design System, version 9.03 */

module ff();
  /* user-specified Verilog code */
  //*************************
  //**   Y = m_1 + m_2
  //*************************
  /**/ reg srcX;
  /**/ reg srcY;
  /**/ assign X = srcX;
  /**/ assign Y = srcY;
  /**/ initial begin
  /**/    srcX = 0;
  /**/    #1
  /**/    srcX = 1;
  /**/    #1
  /**/    $display("X = %b", X);
  /**/    #1
  /**/    $finish;
  /**/ end

  wire X, Y, and_0_yc, and_0_yt, and_2_yc, and_2_yt, buf_0_c, buf_1_c, net_0;
  wire net_11, net_5, net_6, or_0_yc, or_0_yt, pin_16_wire;

  and and_0(net_5, net_0, X);
  and and_2(net_11, net_6, Y);
  not buf_0(net_0, Y);
  not buf_1(net_6, X);
  or or_0(Y, net_11, net_5);
endmodule   /* ff */
```

Electric { Trims redundant parts. also, produces unused wires, sometimes?

Electric makes up instance names, if none assigned.

net_5 is a instance of wire
and_0 is a instance of and
or_0 is a instance of or




**and_0**'s output connects to **or_0**'s input

because **net_5** is in the proper place in both argument lists.

# Connections in verilog

1) by location in arg list

```
and and_0( Y, A, B);
```

```
and and_0( .in1(B), .out(Y), .in0(A) );
```

2) by formal .arg-name ( wire name )
= export name

order doesn't matter

**Verilog 2-level syntatic structure**

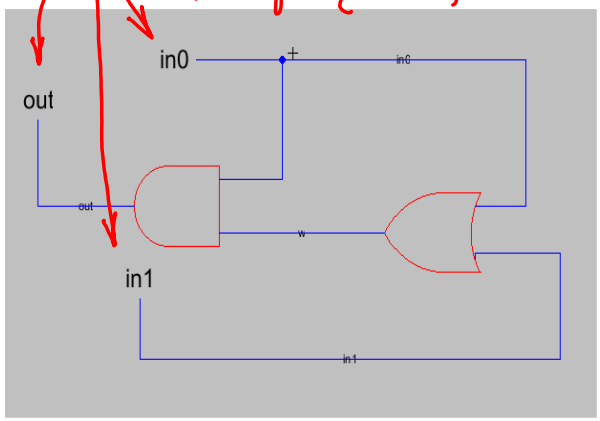
sub-level module definitions

top-level module (a "testbench")

Same syntatic structure as C

```
module and ( out, in0, in1 )

        output out;
         input in0;
         input in1;

         reg out;
         wire in0;
         wire in1;
```

ELectric exports in foo{sch}




⇒ Verilog formal parameters

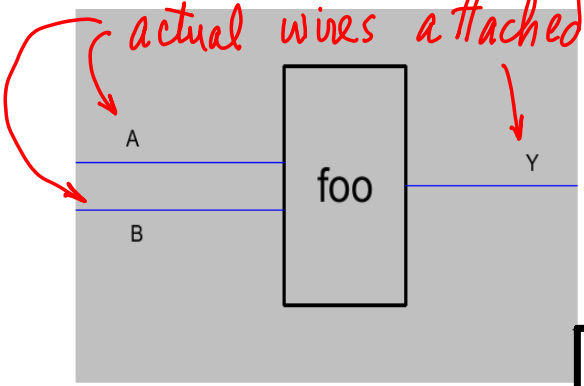
— default type is wire

— outputs can be reg

```
module foo(in0, in1, out);
  input in0;
  input in1;
  output out;

  wire w;

  and and_1(out, in0, w);

  or or_0(w, in0, in1);

endmodule   /* foo */
```

formal parameters

in bar{sch}
actual wires attached to exports

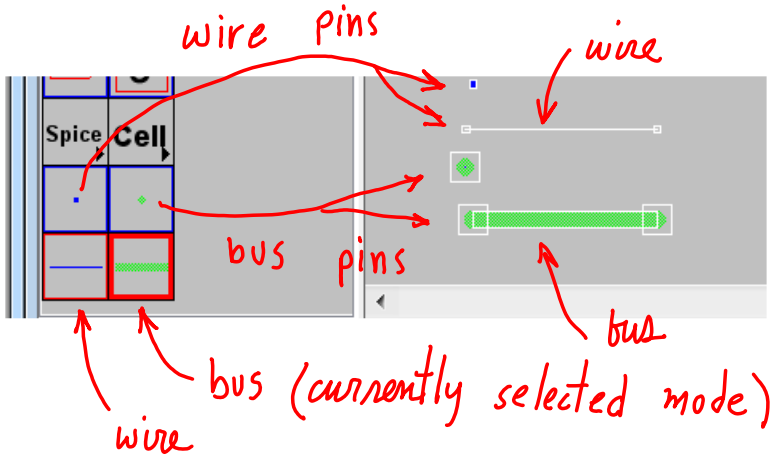



```
module bar();

  wire A, B, Y;

  noname__foo foo_0(.in0(A), .in1(B), .out(Y));

endmodule   /* bar */
```

actual args

**Electric's wires and buses**

wires go from pin to pin.
a bus is a collection of wires.
buses go between bus-pins.

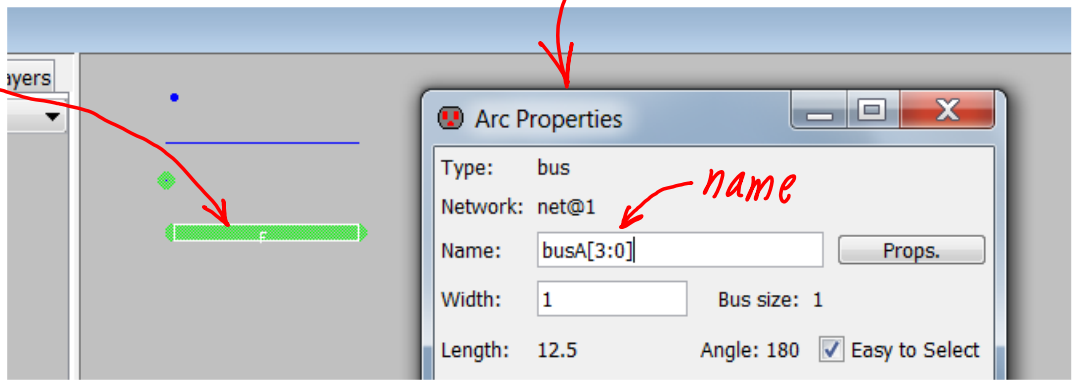



Draw a wire:
1. place a pin
    ^Components.{ wire pin, blue dot }  //-- select pin object
    ^{cell area}  //-- drop pin instance

2. place wire from pin
    ^{pin instance in cell area}  //-- select pin instance
    ^Components.{wire, blue line}  //-- select wire object
    {cell area}^  //-- extend wire from pin

Draw a bus: use bus-pins + bus.
Name the bus, indicate how many wires: ^Edit.Properties.Object Properties

Select bus, not bus-pin




**Naming a bus:**

identifier[ range ]

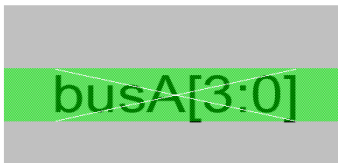
identifier = "busA"
range = "3:0"
means
bus consists of four wires named, "busA[3]", "busA[2]", "busA[1]", and "busA[0]".

Range is [ most-significant-bit : least-significant-bit ]

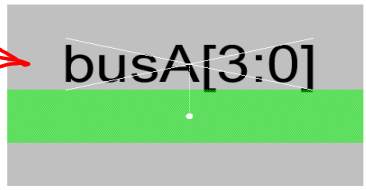
**Selecting Text**

A wire or bus name is displayed as a text object instance.
To move the text, you must select it.
BUT, what if it is under the bus? Select some other text first.
or ^Edit.Text.FindText

bus name text selected

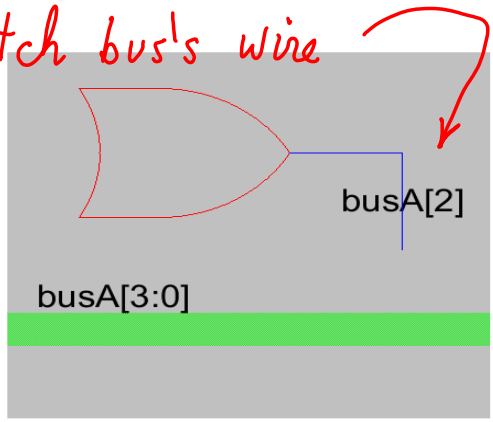



text moved & can see it!

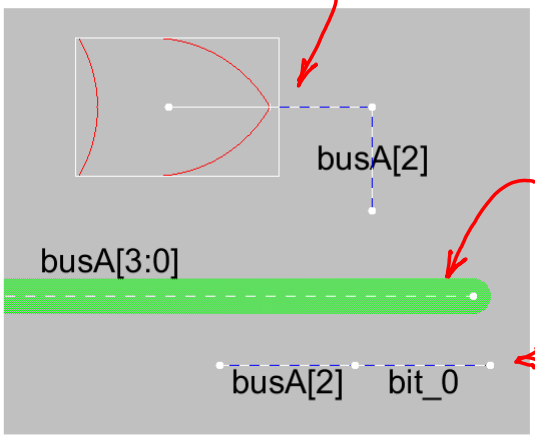



**Connect wire to bus**

name wire to match bus's wire

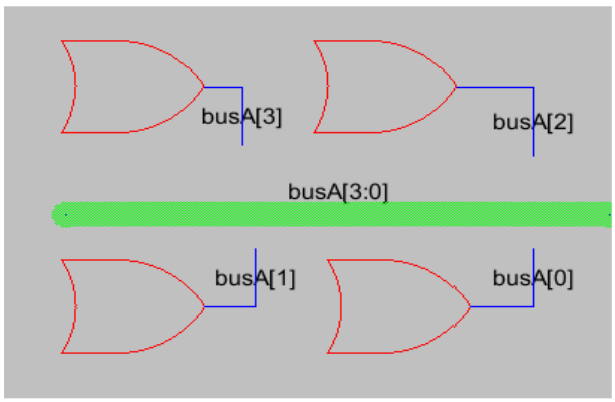



select a pin



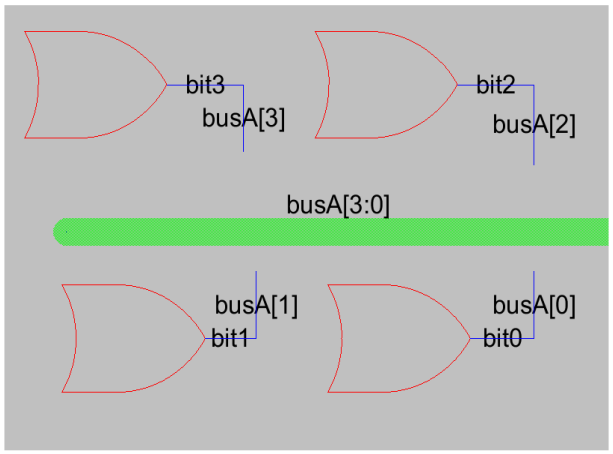

Dashed white shows they are connected.

this is two wires, connected, with different names. Both are connected to busA[2].




here's the verilog connections

```
wire [3:0] busA;

or or_0(busA[2]);
or or_1(busA[3]);
or or_2(busA[1]);
or or_3(busA[0]);
```



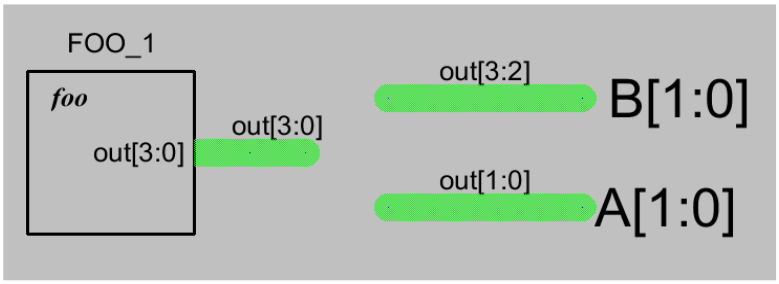

Warning: Electric may rename/trim wires/buses

```
or or_0(bit2);
or or_1(bit3);
or or_2(bit1);
or or_3(bit0);
```

bar { sch }

## Splitting and Joining buses

This cell splits the bus from FOO_1.
Upper 2 bits go to export B[1:0].
Lower 2 bits go to export A[1:0].




The verilog produced drops the bus out[3:0], and uses A[1:0] and B[1:0] instead.

Connection to FOO_1 concatenates A and B buses into a single bus.

```
FOO_1(.out( { B[1], B[0], A[1], A[0] } ) );
```

formal param is bus

bus wires

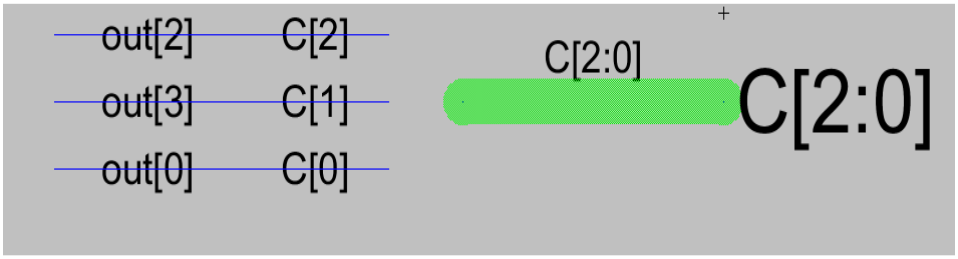
collect to make a 4-wire bus

Collecting wires into a bus:

The wires connected through export C are

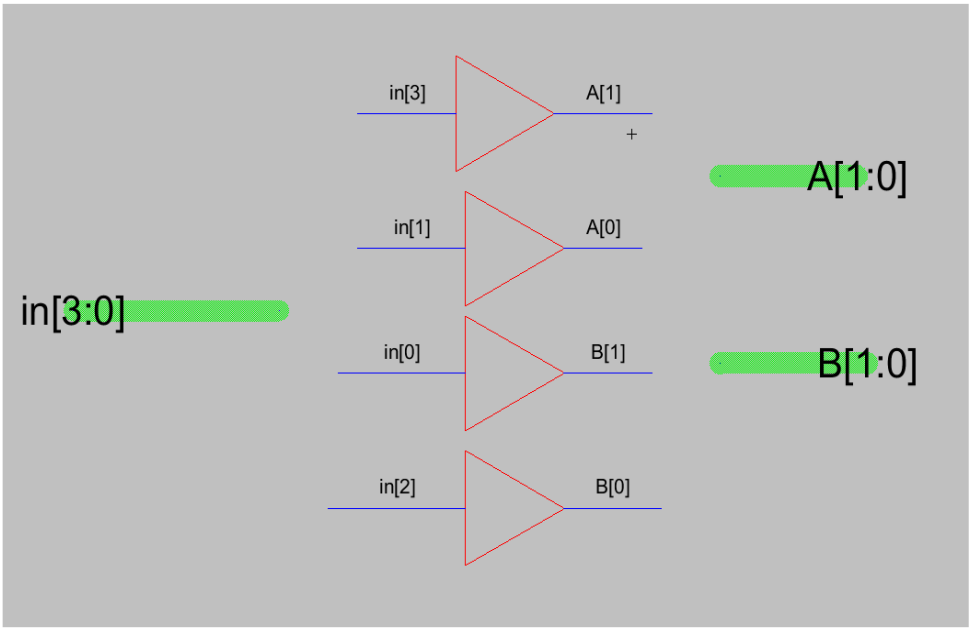
{ out[2], out[3], out[0] }

out[2] is C bus's high bit
out[3] is C bus's bit-1
out[0] is C bus's low bit




BTW, this failed to produce correct verilog: Electric ignored export B. How come?

You need to add buffers so that every signal has a connection to some device, as shown below. Otherwise, Electric trims too much. A buffer is two NOT gates, NOT( NOT() ). They are good for signal restoration or driving long wires.




non-inverting buffer