CHAPTER 1 CSL Auto Router

TABLE 1-1CHAPTER OUTLINE.

1.1 Definitions

TABLE 1.1 Definitions

NCA	Nearest common ancestor algorithm
RTL	Register Transfer Logic
AR	Auto Router

1.2 CSL Auto Router Overview

This chapter explains why the auto router is needed and how it works. The CSLOm records the scope and the name of each CSL signal. The auto router then attempts to connect all signals/ports which are not connected. A connection has a LHS (left hand side) and a RHS (right hand side) expression inside the CSLOm. If either the LHS or the RHS expression is missing then the connection expression is added to the CSLOm. Each connection expression is addded to the CSLOm connection list.

After all csl files have been parsed, the CSLOm connection list is traversed. The CSL auto router creates the missing intermmediate signals, signal groups, ports and interfaces in the CSLOm.

1.2.0.1 The signal router

- will route the signals from one unit to another unit in the design hierarchy. The signal will be added to each unit interface in the design that is in the shortest path between the two or more units in the start point and endpoint lists.
- infers from the design hierarchy and the connect command which signals are missing from the design hierarchy.
- traverses the design hierarchy to determine to which units the inferred signals will be added
- is done after creation of the connection.
- is used to connect signals in different units. The user does not need to specify the intermediate units between the two units.
- determines which units are in the path between the source and destination unit.
- adds the signal name(s) to those units during the signal routing phase.

If the port interface is declared then the auto router checks that the port interface (*signal_list*) and port list (input|output|inout) are correct and then routes the signals in the port list to the modules which use/assign the signals. Signals which are unused or unassigned are routed to the top most module in the list of modules.

Port interfaces do not need to be declared. The interfaces can be inferred from the signals which are used or assigned in the module but are not declared in the module. If the signals are used or assigned in the module and are not declared in any other module then the signals are an input or an output of the top module respectively.
If the signals are used or assigned in the module and are declared in another block then the signals are an input of the module that they are used in and an output of the module that they are assigned in. If the widths of the signals are different then bits which are not driven or are not used are flagged as undriven or unused.

Normally when writing verilog code a signal needs to be specified in each module interface that it is connected to. Instead CSL allows to only specify the endpoints and the cslc signal router will route the signal through the design hierarchy. By route we mean the signal router will add the signal to the modules which connect the two endpoints.
An endpoint can be any of the following:

- signal
- interface
- signal group
- port name
- unit
- instance

Restriction exists for certain connection paths as they are presented in table.
This allows the user to create ports in different blocks and connect the ports. The autorouter will create the intermediate port mappings to wires and ports to connect the two endpoints.

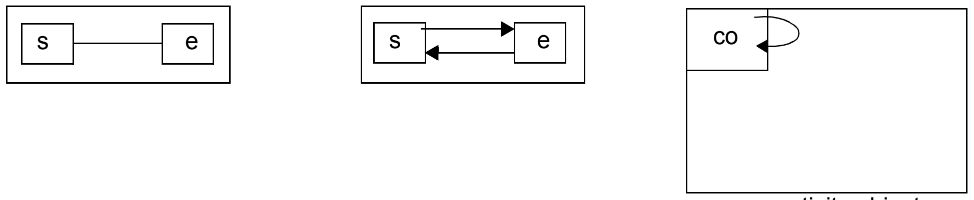
The auto router will connect two or more signals by finding the nearest common ancestor (NCA). The NCA algorithm works by finding the shortest path between the two connection endpoints through the tree hierarchy of the design.

1.3 CSL Auto Router Concepts

1.3.1 Segment routing

Ada t5o connection has a starting point (s) and an end point (e). This is the minimum required to be known in order to make a connection. There are several relationship scenarios when a connection is being made: sibling, parent-child, child-parent, remote, self, ancestor-offspring, offspring-ancestor, no connection (hierarchy is broken).

FIGURE 1.1 Sibling relationship



co = connectivity objects

FIGURE 1.2 Parent to child

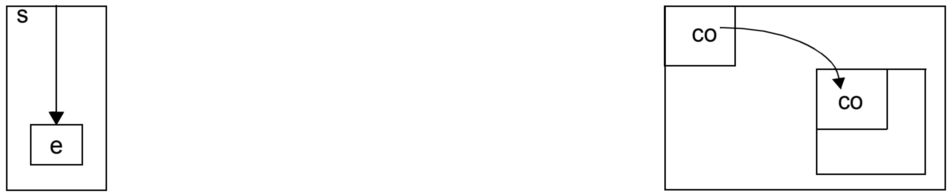


FIGURE 1.3 Child to parent

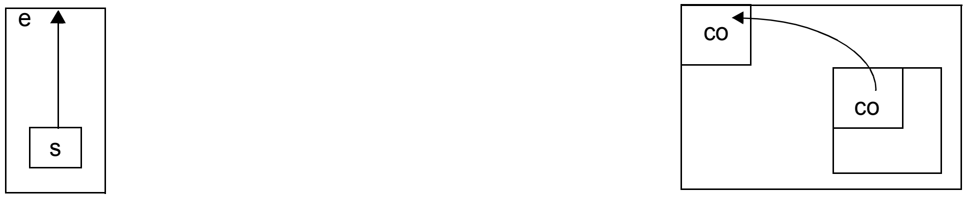


FIGURE 1.4 Ancestor to offspring

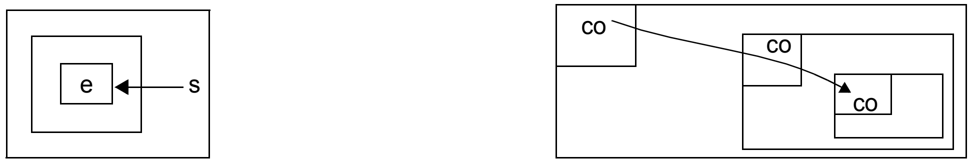
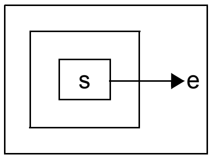
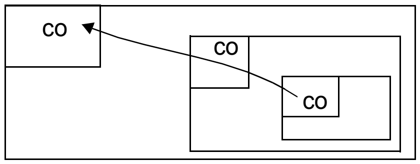


FIGURE 1.5 Offspring to ancestor





1.3.2 Autorouted connection types

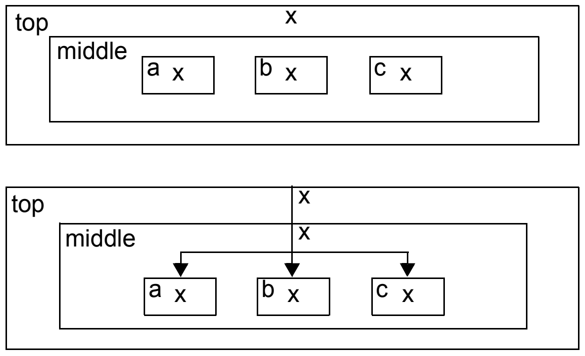
There are the following type of connections:

1. ***connect-by-name***
   - connect by connectivity objects:
     -one-to-one connection: connection of objects with the same name and connection compatibility;
     Connectivity objects can be anywhere in the design as long as they are different hierarchy levels.
     -one-to-many connection: connection of one connectivity object with more than 1 connectivity compatible object;
   - connect by scope:
     The connectivity objects from one scope are connected to the connectivity objects in another scope by name if they are compatible;
2. ***connect by pattern:*** Patterns can be used in connection commands. These commands are expanded by the CSL preprocessor resulting in a set of multiple connect statements of the types listed above.

EXAMPLE :

Figure 1.6 shows an example of auto router name inference. Units a, b and c each have an input named x. These units are instantiated in a unit called middle, which in turn gets instantiated in another unit, higher in the hierarchy, named top. Unit top also has an input named x. The AR connects port **x** from unit top, to all ports **x** in the units a, b and c. The intermediate port from unit middle is automatically inferred by the AR.

FIGURE 1.6 Autorouter connects the points


top
x
middle
a x
b x
c x
BEFORE: only endpoints are shown
top
x
middle
x
a x
b x
c x
AFTER: the connections are shown

CSL CODE

```
csl_unit a{
  csl_port x(input);
  a() {}
};
csl_unit b{
  csl_port x(input);
  b() {}
};
csl_unit c{
  csl_port x(input);
  c(){}
};
csl_unit top{
  csl_port x(input);
  a a0;
  b b0;
  c c0;
  top(){
    x.connect(x);
  }
};
```

VERILOG CODE

```
module top(x);
  input x;
  a a0(.x(x));
  b b0(.x(x));
  c c0(.x(x));
```

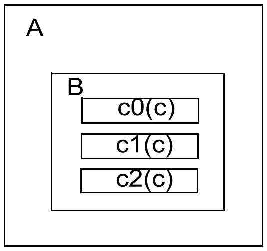
```
endmodule
module a(x);
  input x;
endmodule
module b(x);
 input x;
endmodule
module c(x);
  input x;
endmodule
```

The cslc AR automatically routes signals across design hierarchies. The cslc AR generates the intermediate port lists in the intermediate modules and port mapping in the instantiations.

1.3.2.1 Hierarchy and interconnect //Where should I put this?

The unit hierarchy is the tree of units that comprise a design. Units can be instantiated in other units in order to build design hierarchies. We call the unit where we instantiate other units as the parent unit. The instantiated units we call the children units.
In CSL, the connections in instantiation statements are made by name.
The design interconnect is defined by the unit portlists which are connected to parent unit ports when the unit is instantiated. The formal signal is the port name in the port list. When we instantiate a unit we can specify connection between instance ports and other connectivity objects by specifying a port mapping. In a named port mapping the actual signal is the signal in the parent unit which is connected to the actual signal in the instance which is instantiated.

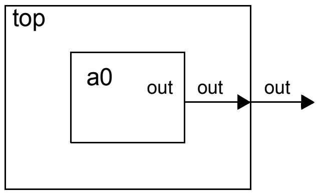
FIGURE 1.7



Units can contain hierarchies of other units and connectivity objects.
An uninstantiated unit is at the top level of the design hierarchy. In a CSL design, there can be only one top unit.

1.The port directions are the same and one of the connection elements is the nearest common ancestor for the other.

EXAMPLE :
// example description

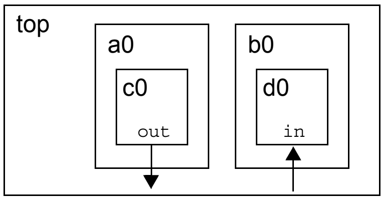


CSL CODE
```
    csl_unit a{
      csl_port out(output);
      a() {}
    };
    csl_unit top{
      csl_port out(output);
      a a0;
      top(){
        a0.out.connect(out);
      }
    };
```

VERILOG CODE
```
    module top(out);
      output out;
      a a0(.out(out));
    endmodule
    module a(out);
      output out;
    endmodule
```

2.The port directions are opposite and the nearest common ancestor is none of the connection elements.

EXAMPLE :
// example description



CSL CODE

```
csl_unit c{
  csl_port out(output);
  c() {}
};
csl_unit d{
  csl_port in(input);
  d() {}
};
csl_unit a{
  c c0;
  a() {}
};
csl_unit b{
  d d0;
  b() {}
};
csl_unit top{
  a a0;
  b b0;
  top(){
    a0.c0.out.connect(b0.d0.in);
  }
};
```

VERILOG CODE

```
module top();
  wire ar_out_in0;
  a a0(.ar_out_in_out0(ar_out_in0));
  b b0(.ar_out_in_in0(ar_out_in0));
endmodule
```

```
module a(ar_out_in_out0);
  output ar_out_in_out0;
  c c0(.out(ar_out_in_out0));
endmodule

module b(ar_out_in_in0);
  input ar_out_in_in0;
  d d0(.in(ar_out_in_in0));
endmodule

module c(out);
  output out;
endmodule

module d(in);
  input in;
endmodule
```

Auto connect bottom up finds nearest common parent (shortest path) between common names, checks and adds ports to the module in the path.

FIGURE 1.8 Auto connection algorithm - nearest common ancestor (NCA)

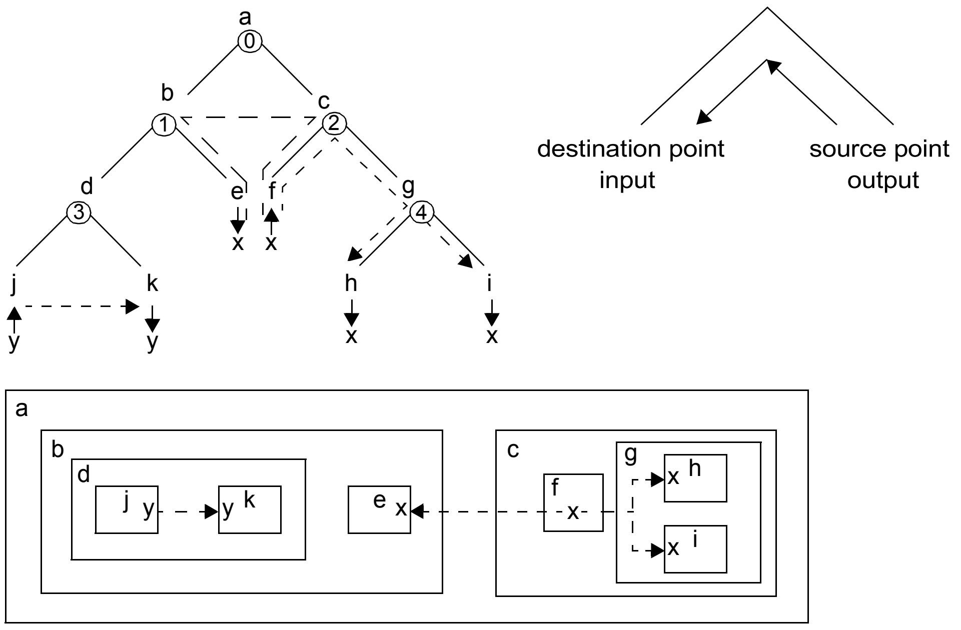


TABLE 1.2 Nearest Common Ancestor Connection between leaf nodes

src	dst_list	nca	nca_path
f.x	h.x	c	(f,c,g,h)
f.x	i.x	c	(f,c,g,i)
f.x	e.x	a	f(c,a,b,e)

In CSLOM (Chip Specification Language Object Model) the auto router traverses through the design connection list and finds all unconnected (undriven and unassigned) signals, then attempts to connect the signals using NCA and additional information which specifies whether the signal is connected to the upper level interface, within the local scope or to the connectivity objects of the lower level units

1.3.2.2 Connection relationship

Units are connected to other units explicitly using the CSL connect command or implicitly by name. When connecting units within design hierarchy, three connection relationships need to be handled in the design hierarchy:

- parent to child connection
- child to parent connection
- sibling to sibling connection

TABLE 1.3 Direction inference

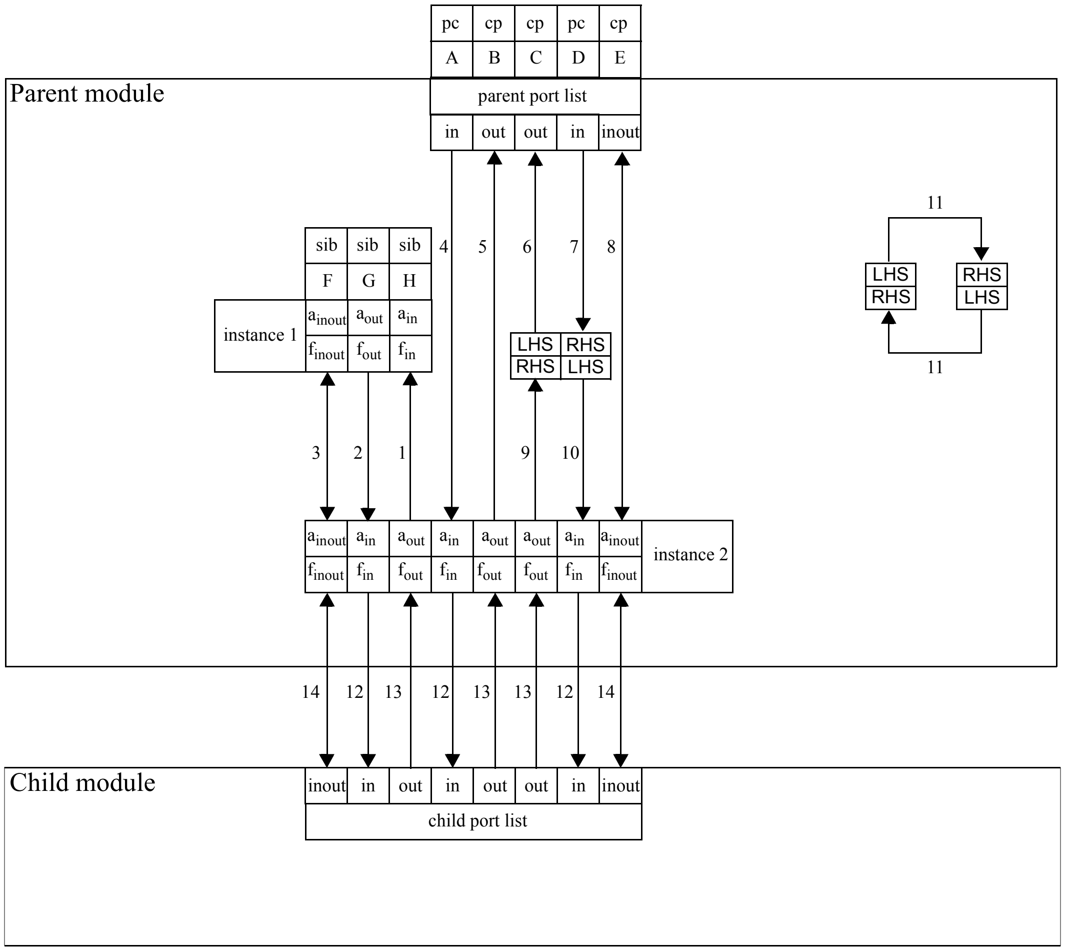
relationship	unit a	unit b	port dir unit a	port dir unit b	wire	Q	Error	Done
sib	in	-		out	x	x		
p c	in	-		in		x		
c p	in	-		in		x		
sib	out	-		in	x	x		
p c	out	-		out		x		
c p	out	-		out		x		
sib	inout	-		inout	x	x		
p c	inout	-		inout		x		
c p	inout	-		inout		x		
sib	in	in					x	x
p c	in	in						x
c p	in	in						x

TABLE 1.3 Direction inference

relationship	unit a	unit b	port dir unit a	port dir unit b	wire	Q	Error	Done
sib	in	out			x			
p c	in	out					x	
c p	in	out					x	
sib	in	inout						
p c	in	inout						x
c p	in	inout						
sib	out	in			x			
p c	out	in					x	
c p	out	in					x	
sib	out	out					x	
p c	out	out						x
c p	out	out						x
sib	out	inout			x			
p c	out	inout					x	
c p	out	inout					x	
sib	inout	in					x	
p c	inout	in						x
c p	inout	in						x
sib	inout	out			x			
p c	inout	out					x	
c p	inout	out					x	
sib	inout	inout					x	
p c	inout	inout						x
c p	inout	inout						x

If there are no other connections create a port and a wire else push an unsolved Q.

FIGURE 1.9 connections


pc cp cp pc cp
A B C D E
Parent module
parent port list
in out out in inout
sib sib sib
F G H
instance 1
a_{inout} a_{out} a_{in}
f_{inout} f_{out} f_{in}
1 2 3 4 5 6 7 8 9 10 11 11
LHS RHS
RHS LHS
instance 2
a_{inout} a_{in} a_{out} a_{in} a_{out} a_{out} a_{in} a_{inout}
f_{inout} f_{in} f_{out} f_{in} f_{out} f_{out} f_{in} f_{inout}
14 12 13 12 13 13 12 14
Child module
inout in out in out out in inout
child port list

Details.

TABLE 1.4 Notations

i	instance
c	child
p	parent
a	actual

LHS	=	RHS
output		input
actual in		actual out

TABLE 1.4 Notations

f	formal
l	local

TABLE 1.5 Segment router truth table

No	LHS	RHS				cp	cp	pc	cp	sib	sib	sib
		input				b	c	d	e	f	g	h
1	i.a.in	i.a.out	instance.actual.in	instance.actual.out							1	
2	i.a.out	i.a.in	instance.actual.out	instance.actual.in						2		
3	i.a.inout	i.a.inout	instance.actual.inout	instance.actual.inout						3		
4	p.p.in	i.a.in	parent.port.in	instance.actual.in	4							
5	p.p.out	i.a.out	parent.port.out	instance.actual.out		5						
6	p.p.out	l.lhs	parent.port.out	local.lhs			6					
7	p.p.in	l.rhs	parent.port.in	local.rhs				7				
8	p.p.inout	i.a.inout	parent.port.inout	instance.actual.inout					8			
9	l.rhs	i.a.in	local.rhs	instance.actual.in			9					
10	l.lhs	i.a.out	local.lhs	instance.actual.out				10				
11	l.lhs	l.rhs	local.lhs	local.rhs	12							
12	i.f.in	c.p.in	instance.formal.in	child.port.in	12			12				12
13	i.f.out	c.p.out	instance.formal.out	child.port.out		13	13				13	
14	i.f.inout	c.p.inout	instance.formal.inout	child.port.inout					14	14		

TABLE 1.6 Connectivity table between connection objects

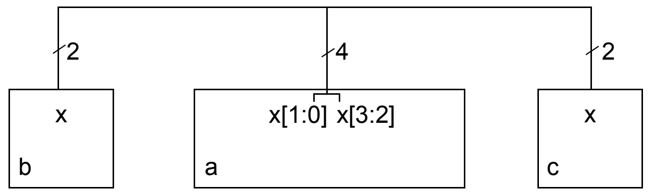
Connection object	signal	port	signal group	interface
signal	X*	X	-	-
port	X	X	-	-
signal group	-	-	X**	X
interface	-	-	X	X

* - only if at least one of the signals can have inferred direction (e.g. connected to a port)
** - only if at least one of the signal groups in the connection can have inferred direction (e.g. connected to an interface)

TABLE 1.7

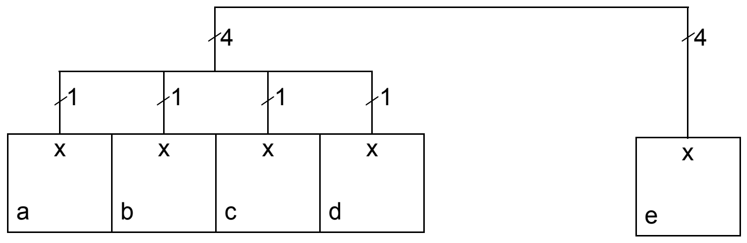
			RHS														
			Parent module			Child module			Actual instance			Formal instance			Local		
			IN	O	IO	IN	O	IO	IN	O	IO	IN	O	IO	IN	O	IO
LHS	Parent module	IN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		O	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
		IO	-	-	-	-	-	-	-	-	4	-	-	-	-	-	13
	Child module	IN	-	-	-	-	-	-	-	-	-	11	-	-	-	-	-
		O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		IO	-	-	-	-	-	-	-	-	-	-	-	17	-	-	-
	Actual instance	IN	3	-	-	-	-	-	1	2	-	-	-	-	-	8	-
		O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		IO	-	-	-	-	-	-	-	-	16	-	-	-	-	-	15
	Formal instance	IN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		IO	-	-	-	-	-	12	-	-	-	-	-	-	-	-	-
	Local	IN	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		O	-	-	-	-	-	-	-	7	-	-	-	-	9/10	-	-
		IO	-	-	13	-	-	-	-	-	15	-	-	-	-	-	14

FIGURE 1.10 Part select in connect function



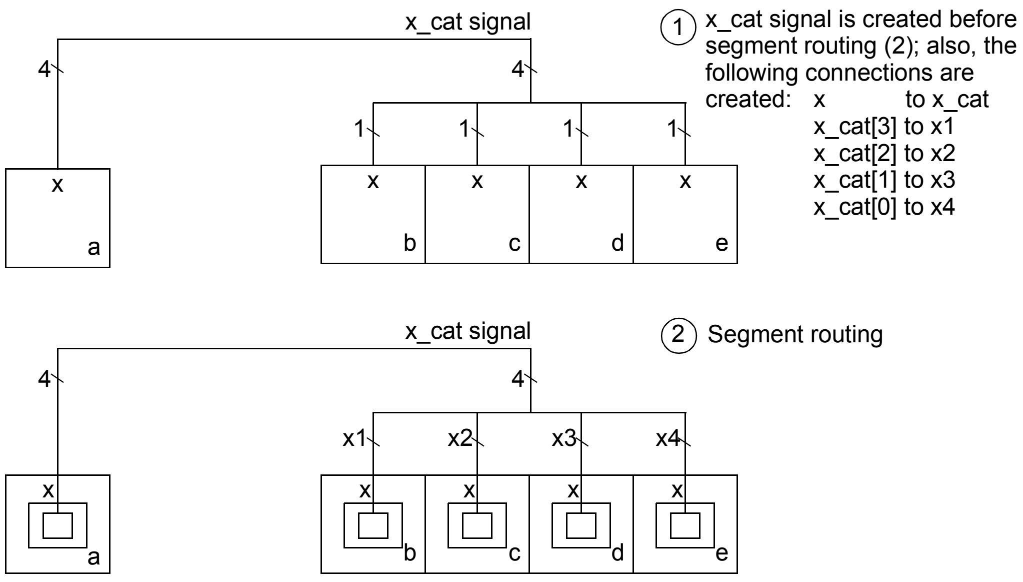
```
a.x(1,0).connect(b.x);
a.x(3,2).connect(c.x);
```

FIGURE 1.11 Concatenation in connect function (LHS)



```
list l(a.x,b.x,c.x,d.x);
l.connect(e.x);
```

FIGURE 1.12 Concatenation in connect function (RHS)



1.3.3 Autorouter inferred objects

As it connects two or more endpoints the autorouter infers different objects depending on the relation between the connected endpoints in the tree.

1.3.4 Checker

A checker verifies which signals are connected and if these connections are defined correctly (width, type and direction check).
The interconnect processing steps are:

- build the CSLOm
- run the AR
- check CSLOm connections

1.3.5 Connectivity object

In CSL the connections are specified between connectivity objects. The connectivity objects are: ports, interfaces, signal, signal group, concatenation of connectivity objects expressions that evaluate to sized connectivity object. In order to simplify connection specification, special constructs are allowed which are converted by the CSL compiler into simple connections.
Part selects are allowed to be used to specify connects between group of bits of a connectivity object with other connectivity objects.
A multibit connectivity object can be connected to a heterogeneous set of connectivity objects if they respect the connection rules.

TABLE 1.8 Formal to actual connection types

Actual arg	Example types
expression	w w0 (.x(a&b))
constant	w w0 (.x(1’b0))
concatenation	w w0 (.x({a,b,c,d}))
rename	w w0 (.x(y))
part select	w w0 (.x(p[7:4]))

1.4 CSL Auto Router Rules

!!!TO BE REVIEWED!!!

1. When connecting a port of the parent unit to a port in the child unit no wire is created.

2.When connecting two ports belonging to sibling units the AR only creates wires.

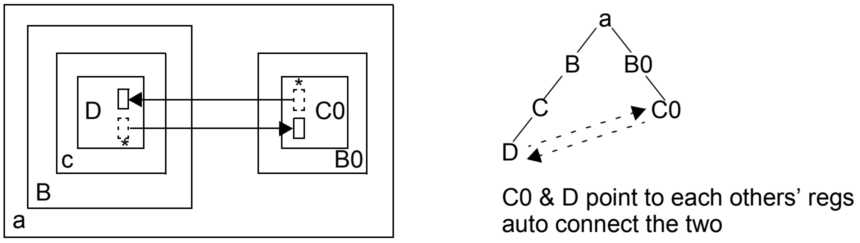
3.When connecting ports on different branches of the tree, the autorouter will infer ports for each level until it reaches the scope of the nearest common ancestor where it will infer wires.

4.Each unit or instance should store a vector of numbers (according to the path from top to that of unit instance and considering the numbers given by the unique number build factory)

5.Add the pair (number, reference to the instance) to the global map named *mapNumberToInstances*.

<ADD>

FIGURE 1.13



</ADD>

Connections can be divided in the following categories
-Local connections: the connection do not cross hierarchy levels
-Cross hierarchy connections: the connections cross at least one hierarchy boundary
Local connections can be of the following types:
non hierarchical: sibling to sibling;
hierarchical: parent-child; child-parent;

Local connections are specified using formal to actual mapping. Trying to make a connection using the connect statement for a local connection will issue an error message.

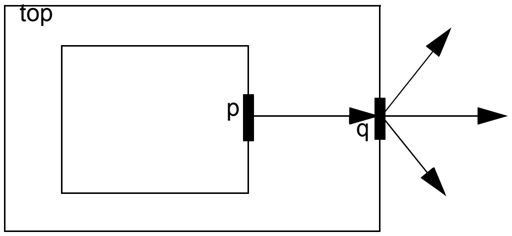
The connect statement is used to make connections that create signal paths that cross at least one hierarchy boundary
A connection can be complete or partial
A complete connection is a connection between two connectivity objects that have direction attribute.
A partial connection is a connection between two connectivity objects where only one has the direction attribute

Routing ports

Signal paths that connect connectivity objects in different hierarchies must cross hierarchy bound-

ary through ports or ports in interfaces.
In the process of creating a connection between two connectivity objects in different hierarchies the auto router will do the following:
-search to route the connection through suitable ports
A port is suitable to route the connection if:
a) it has the same name as the start connectivity object;
b) the direction of the port does not conflict with the direction of the connection;
c) there are no other explicit connections routed through that port, that induce name conflicts;
d) the port width fits the width of the connection path.
The search can have the following results:
-there is no suitable port to route the signal path through it. In this case the auto router will create a suitable port through which will be routed the signal path.
-there is a suitable port. The signal path will be routed through that port.
Note that a port can be crossed by multiple signal paths, in this case the port is only entered by one signal and drives at its output multiple signals.

FIGURE 1.14



Port routing multiple signal paths.

FIGURE 1.15 Port q is not satisfying condition a)

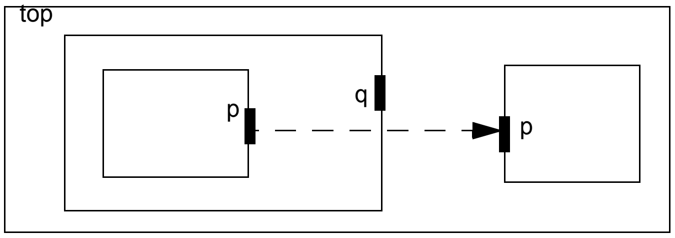


FIGURE 1.16 Port p not satisfying condition b)

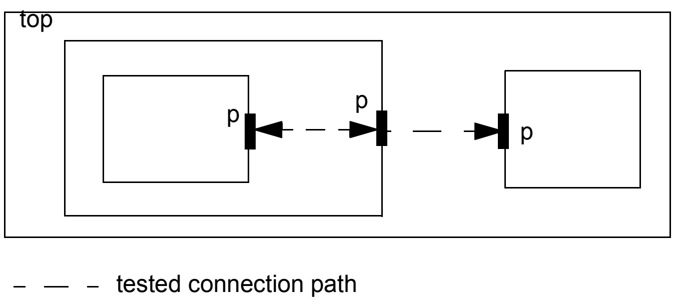


FIGURE 1.17 Port p not satisfying condition c)

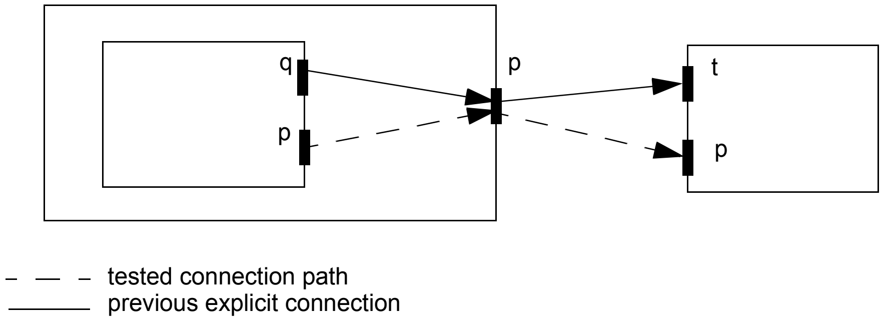
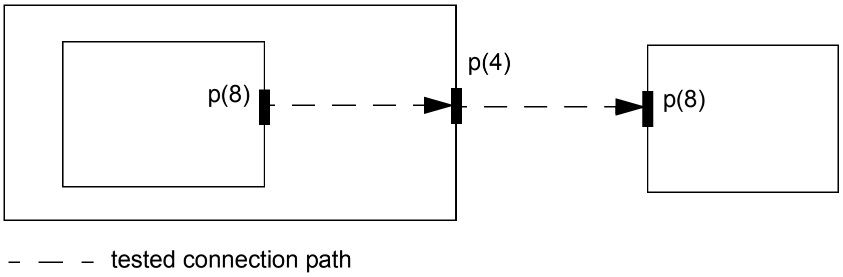


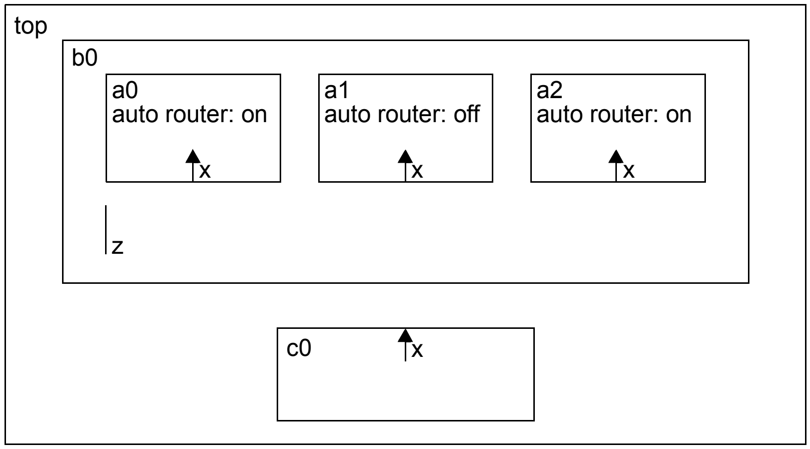
FIGURE 1.18 Port p not satisfying condition d)



1.5 CSL Auto Router Examples

The first example focuses on using the auto router on/off filter on unit instances to control port inference. Figure 1.19 shows how the auto router can search ports and connect them using name matching; the two instances which allow this will have the ports connected, while the unit which doesn't allow for this will not have its ports connected by the auto router. Usually this is needed when a certain port is required to be connected differently than other ports having the same name.

FIGURE 1.19 Initial layout before the auto router operation



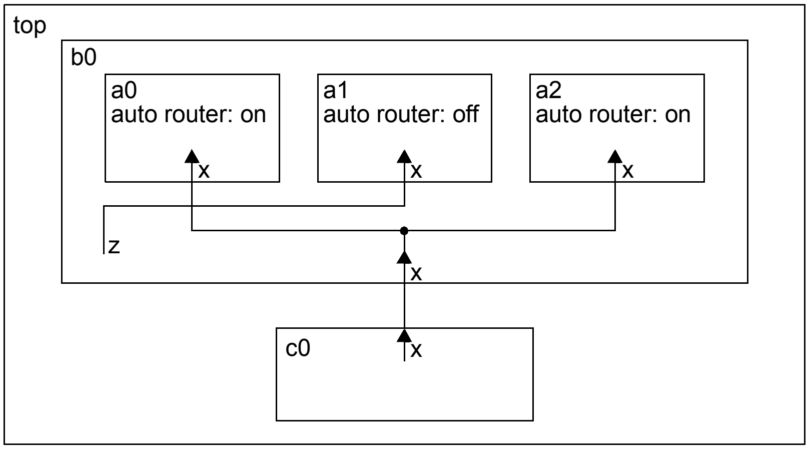
CSL CODE

```
csl_unit a{
  csl_port x(input);
a(){}
};
csl_unit c{
  csl_port x(input);
c(){}
};
csl_unit b{
  csl_signal z;
  a a0;
  a a1;
  a a2;
b(){
a1.set_auto_router(off);
}
};
csl_unit top{
  csl_port x(output);
  b b0;
  c c0;
top(){
x.connect(x);
b0.z.connect(a1.x);
}
};
```

VERILOG CODE

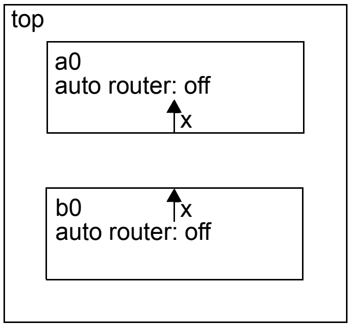
```
module top(x);
  output x;
  b b0();
  c c0();
endmodule
module c(x);
  input x;
endmodule
module b();
  wire z;
  a a0();
  a a1();
  a a2();
endmodule
module a(x);
 input x;
endmodule
```

FIGURE 1.20 Final layout after the auto router operation



The next two examples show how the auto router filter affects the connection between units/ instances with the set_auto_router flag set to off.

FIGURE 1.21 Initial two unconnected sibling units



When the connect function is called on connecting the two units in Figure 1.21 the connection is being made because no ports need to be inferred.

CSL CODE

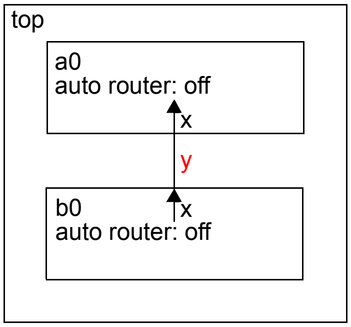
```
csl_unit top,a,b;
a.add_port(input,x);
b.add_port(output,x);
scope top {
  add_instance(a,a0);
  add_instance(b,b0);
  /* this works even if the auto router is set to off because
  the two units are siblings and no ports need to be inferred */
  a0.x.connect(b0.x);
}
```

VERILOG CODE

```
module top();
  wire y;
  a a0(.x(y));
  b b0(.x(y));
endmodule
module a(x);
  input x;
endmodule
module b(x);
  output x;
endmodule
```

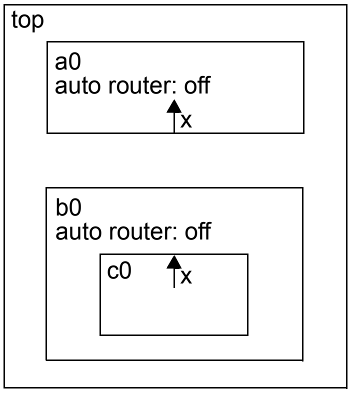
The connection is shown in Figure 1.22

FIGURE 1.22 Two connected sibling units



In Figure 1.23 the connection between the two ports (called 'x') is not being made because the auto router is turned off and since the two instances used in the connection are not at the same level in the hierarchy (are not siblings) the auto router cannot infer a port in the instance b0's interface.

FIGURE 1.23 Ports to be connected do not belong to sibling instances



It doesn't matter what the setting for the auto router is for unit instance c0. As long as the auto router is set to off for unit instance b0, the port required to be inferred in b0's interface cannot be created by the auto router so the connection between port x in c0 and port x in a0 cannot be established this way.

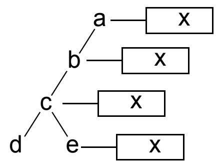
CSL CODE

```
csl_unit top,a,b,c;
a.add_port(input,x);
c.add_port(output,x);
b.add_instance(c,c0);
scope top {
  add_instance(a,a0);
  add_instance(b,b0);
  /* this cannot be established because the auto
  router is turned off in parent unit instance b0 */
  a0.x.connect(b0.c0.x);
}
```

EXAMPLE :

Autorouter connect a.x to e.x

FIGURE 1.24



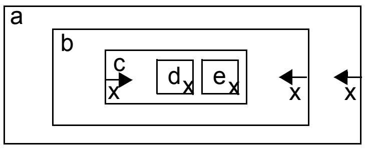


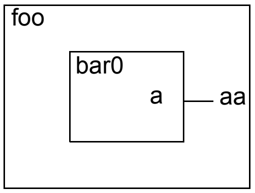
TABLE 1.9

b	add output port x
c	wire x and connect to b b0(.x(x)) and d d0(.x(x))
d	add input port x
e	add input port x

EXAMPLE :

DESCRIPTION :

In the example shown below the unit foo instantiates the unit bar. The signal aa in the unit foo is the actual signal and the signal a in the bar unit is the formal signal.



```
module foo ();
  wire aa;
  bar bar0 (.a(aa));
endmodule

module bar (a);
  output a;
endmodule
```

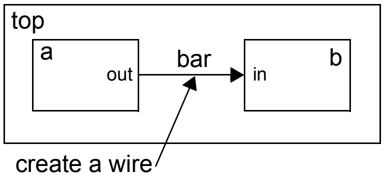
Signals can be auto routed between modules.

EXAMPLE :

DESCRIPTION :

Two sibling units are instantiated in a parent unit top. One has an output port while the other has an input port. When the connect method is used to link the two units, the autorouter will generate a wire in the parent unit.

FIGURE 1.25 Port to port autorouter connection



CSL CODE:

```
csl_unit top,a,b;
top.add_instance(a,a0);
top.add_instance(b,b0);
a.add_port(output,out);
b.add_port(input,in);
//LHS formal name  RHS actual name
top.a0.out.connect(top.b0.in);
// or another approach
top.add_signal(bar);
top.a0.out.connect(top.bar);
top.bar.connect(top.bo.in);
```

VERILOG CODE:

```
module top;
  wire bar;
  a a0(.out(bar));
  b b0(.in(bar));
endmodule
module a(out);
  output out;
endmodule
module b(in);
  input in;
endmodule
```

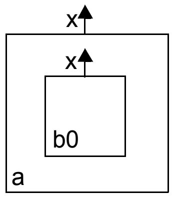
EXAMPLE :

Of two units (a and b) each have an output port named x. One of the units (b) is instantiated inside the other. The autorouter will detect the ports with the same name and connect them, without inferring any new objects.

CSL CODE:

```
csl_unit a,b;
a.add_port(output,x);
b.add_port(output,x);
a.add_instance(b,b0);
//a.b0.x.connect(b0.x);
```

FIGURE 1.26 Port to port autorouter connection



VERILOG CODE:

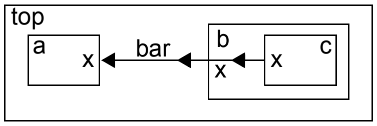
```
module a(x);
  output x;
  b b0(.x(x));
endmodule
module b(x);
  output x;
endmodule
```

EXAMPLE :

DESCRIPTION :

Two units (a and c) have an input port and an output port respectively (both are named x). Unit c is instantiated in unit b, and then units a and b are instantiated in unit top. The autorouter will traverse the hierarchy and infer ports and wire to make the connection between the two x ports. For example: starting from unit c, the output port x is infered for unit b and then linked to the one from the instance of c. As it reaches the scope of unit top, the autorouter will be at the nearest common ancestor for signal objects (ports) x in units a and c. In top instances a and b are siblings so, the autorouter will infer a wire and connect ports x and the infered port x from units a and b respectively, thus completing the connection operation.

FIGURE 1.27



CSL CODE:

```
csl_unit top,a,b,c;
top.add_instance(a,a0);
a.add_port(input,x);
c.add_port(output,x);
b.add_instance(c,c0);
//top.a0.x.connect(top.b0.x)
```

VERILOG CODE:

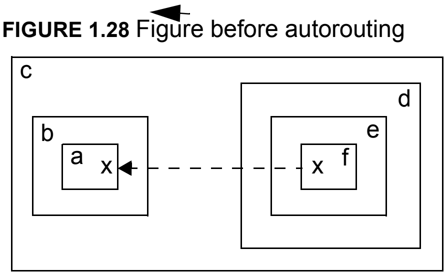
```
module top();
  wire x;
  a a(.x(x));
  b b(.x(x));
endmodule
module a(x);
  input x;
endmodule
module c(x);
  output x;
endmodule
module b(x);
  output x;
  c c(.x(x));
endmodule
```

EXAMPLE :

DESCRIPTION :

//

FIGURE 1.28 Figure before autorouting



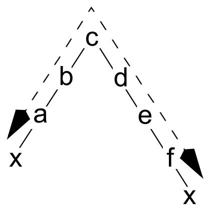
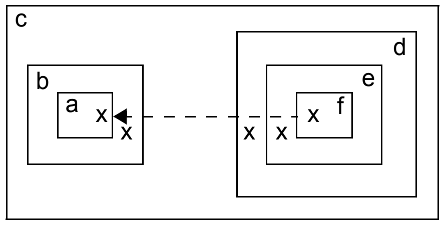


FIGURE 1.29 Figure after autorouting



CSL CODE:

```
csl_unit a,b,c,d,e,f;
c.add_instance(b,b0);
c.add_instance(d,d0);
b.add_instance(a,a0);
d.add_instance(e,e0);
e.add_instance(f,f0);
a.add_port(input,x);
f.add_port(output,x);
//c.b0.a0.x.connect(c.d0.e0.f0.x)
```

VERILOG CODE:

```
module c();
  wire x;
  b b0(.x(x));
  d d0(.x(x));
endmodule
module a(x);
  input x;
endmodule
module b(x);
```

```
    input x;
    a a0(.x(x));
  endmodule
  module f(x);
    output x;
  endmodule
  module e(x);
    output x;
    f f0(.x(x));
  endmodule
  module d(x);
    output x;
    e e0(.x(x));
  endmodule
```

Read csl specification and auto route the signals between units
Read RTL Verilog port lists and port declarations and route the signals between modules

<MOVED FROM CSL INTERCONNECT>
The CSL compiler (cslc) uses the design hierarchy and the signal end points to automatically connect signal and points and mappings with the same name.If all the bits in a csl endpoint vector signal are not driven, then an error is flagged and reported to a log file.

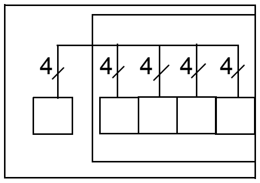
1.5.0.0.1 Design Hierarchy

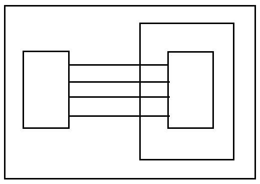
The CSL interconnect specification is used to define the design hierarchy for a chip design. A hierarchical file is a file which is part of the design hierarchy.The hierarchy file specifies the hierarchy for the chip. The design hierarchy files can be nested. The design hierarchy file lists the instance names and template/module names. A leaf level file is a file at the lowest level of the design hierarchy. The leaf level files are user generated.

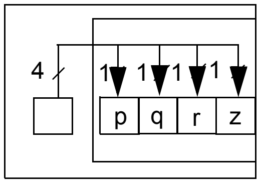
If the signal is not declared and if it is on the LHS of the assignment then it is an output.
If the signal is not declared and if it is on the RHS of the assignment then it is an input.
If the signal is not declared and if it is on the LHS of assignment and the RHS of the assign statement is of the form: *sel* ? *expr* : 1'bZ; then it is an inout trireg.
If the LHS var is in an assign statement and the variable is not used in the RHS of an statement in the module width inference.

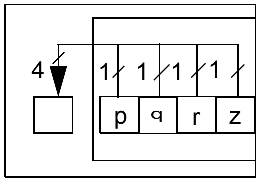
If a signal is used in one module and the signal is driven in another module then the cslc auto router will create the port names in the modules which connect the module which drives the signal and the module that uses the signal.
</MOVED>

0. single point   single point

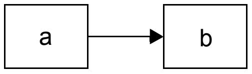


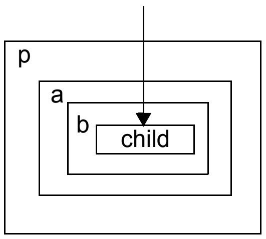






1. single point   multi point or multi to single





a.connect(l(b,c,d))
is equivalent to

```
  a.connect(b)
  a.connect(c)
  a.connect(d)
```

2. multi to multi

   bus to bus connection

```
  l(a,b,c).connect(list(p,q,r))

  a.connect(p);
  b.connect(q);
  c.connect(r);
```

3. split concat

```
   x[3:0].connect({p.x,q.x,r.x,z.x)
```

4. join (concat|), split

```
   ({p.x,q.x,r.x,z.x}).connect(x[3:0])
```

5.csl_unit a,b;

```
 a.get_ifc().connect(b);
```

Copy ifc from a to b.

6.parent.get_ifc().connect(child);