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 which units to add the inferred signals to.
- 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 route 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

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-descendant,descendant-ancestor,no connection (hierarchy is broken).

FIGURE 1.1 Sibling relationship

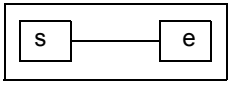


FIGURE 1.2 Parent to child

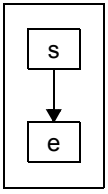


FIGURE 1.3 Child to parent

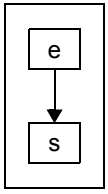


FIGURE 1.4 Ancestor to descendant

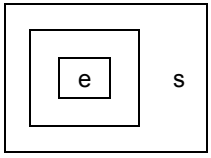
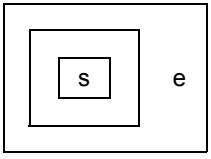


FIGURE 1.5 Descendant to ancestor



1.3.2 Autorouted connection types

The cslc auto router automatically connects signals in the following cases:

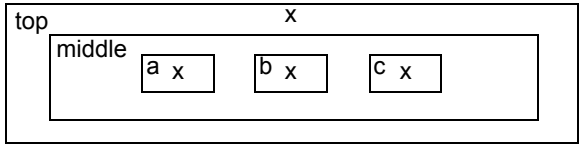
1. connect by name - inference when the names of the connection endpoints are the same.

2. csl connection statement - explicit connections specified by the **connect** method.

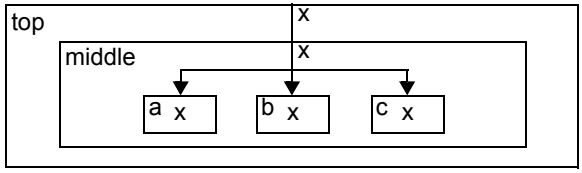
EXAMPLE :

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 infered by the AR.

FIGURE 1.6 Autorouter connects the points



BEFORE: only endpoints are shown



AFTER: the connections are shown

CSL CODE

```
csl_unit top, a, b, c, middle;
top.add_port(input,x);
middle.add_port(input,x);
a.add_port(input,x);
b.add_port(input,x);
c.add_port(input,x);
top.add_instance(middle,middle0);
middle.add_instance(a,a0);
middle.add_instance(b,b0);
middle.add_instance(c,c0);
```

VERILOG CODE

```
module top(x);
  input x;
  middle middle0(.x(x));
endmodule
```

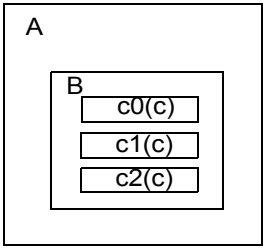
```
module middle(x);
  input x;
  a a0(.x(x));
  b b0(.x(x));
  c c0(.x(x));
endmodule
module a(x);
  input x;
endmodule
module b(x);
  input x;
endmodule
module c(x);
  input x;
endmodule
```

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 instantiate units to build design hierarchies. We call the instantiated units the children and the unit the parent. The design interconnect are the unit portlists which are connected to parent units when the unit is instantiated. The formal signal is the port name in the port list. 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



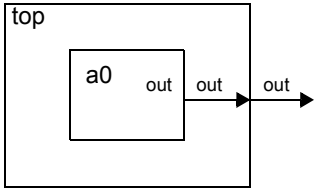
1.3.3 NCA Algorithm

Autoconnect feature works in a buttom-up fashion. It finds the nearest common ancestor between common names and establishes a connection based on one of these two cases:

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 top, a;
top.add_instance(a,a0);
a.add_port(output,out);
top.add_port(output,out);
//top.a0.out.connect(top.out);
```

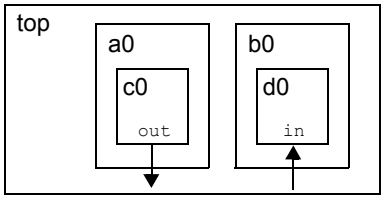
VERILOG CODE

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

module a;
 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 top, a, b, c, d;
top.add_instance(a,a0);
top.add_instance(b,b0);
a.add_instance(c,c0);
b.add_instance(d,d0);
c.add_port(output,out);
d.add_port(input,in);
//top.a0.c0.out.connect(top.b0.d0.in);
```

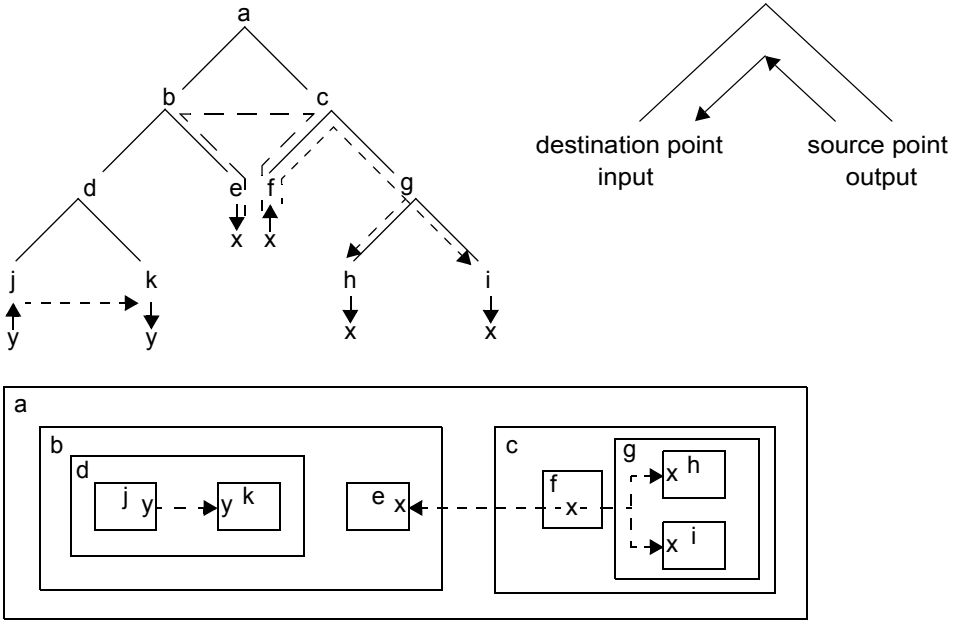
VERILOG CODE

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

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

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

module c;
 output out;
endmodule
```

```
module d;
 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)



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 attempt to connect the signals using NCA and additional information which specifies whether are not the signal is connected to the upper interface, the local scope or the lower level units

1.3.3.1 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 1.2 Direction inference

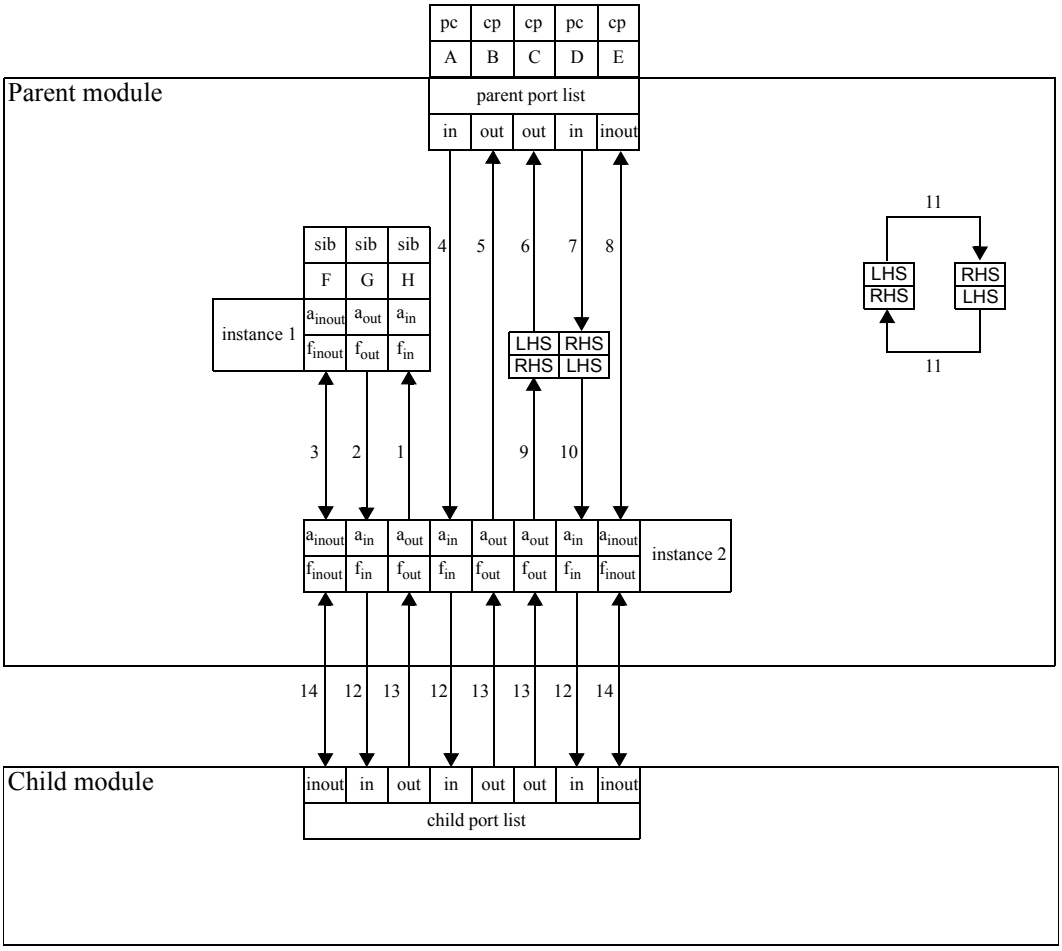
relationship	unit a	unit b	port dir unit a	port dir unit b	wire	Q	Error	Done
sib	in	-		out	x	x		
p c	in	-		in		x		
c p	in	-		in		x		
sib	out	-		in	x	x		
p c	out	-		out		x		
c p	out	-		out		x		
sib	inout	-		inout	x	x		
p c	inout	-		inout		x		
c p	inout	-		inout		x		
sib	in	in					x	x
p c	in	in						x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

TABLE 1.2 Direction inference

relationship	unit a	unit b	port dir unit a	port dir unit b	wire	Q	Error	Done
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



Details.

TABLE 1.3 Notations

i	instance
c	child
p	parent
a	actual

TABLE 1.3 Notations

f	formal
l	local

TABLE 1.4 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		

LHS = RHS

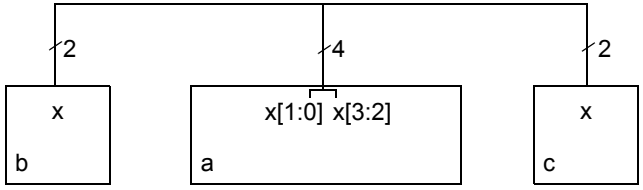
output | input

actual in | actual out

TABLE 1.5

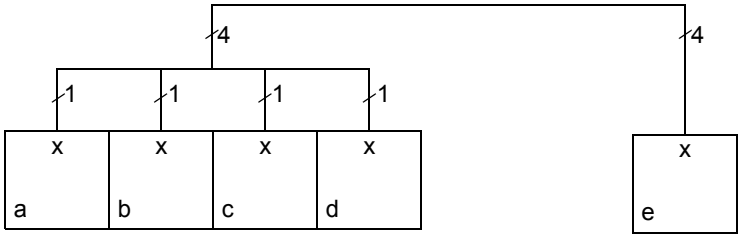
			RHS														
			Parent module			Child module			Actual instance			Formal instance			Local		
			IN	O	IO	IN	O	IO	IN	O	IO	IN	O	IO	IN	O	IO
LHS	Parent module	IN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		O	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
		IO	-	-	-	-	-	-	-	-	4	-	-	-	-	-	13
	Child module	IN	-	-	-	-	-	-	-	-	-	11	-	-	-	-	-
		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



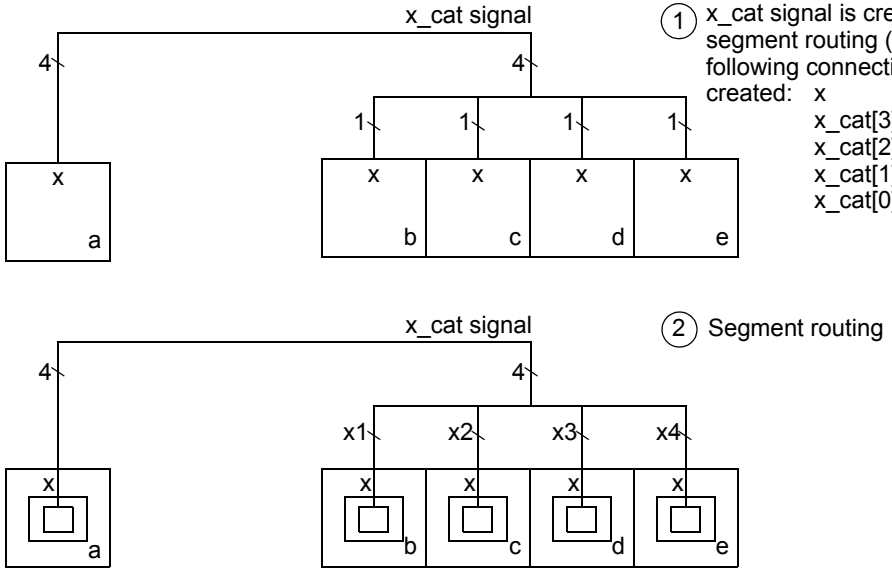

```
a.x(1,0).connect(b.x);
a.x(3,2).connect(c.x);
```

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)



1.3.4 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.

1.3.5 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:

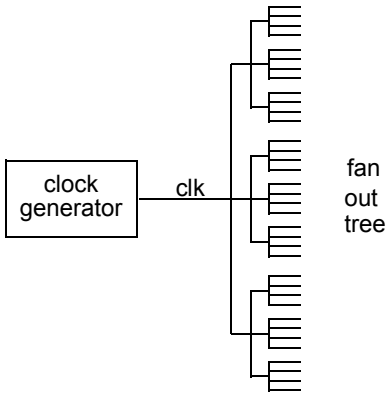
- build the CSLOm
- run the AR
- check CSLOm connections

1.3.6 Creating Unit Instances//incomplete explanation

top is a unit which is declared using CSL. We declare other units then we add them as children to their parent units using the **add_instance** keyword.
The signal list contains signals which have CSL signal type attributes and drive fan out trees.

EXAMPLE :

FIGURE 1.13 Fan out tree



CSL CODE:

```
csl_signal clk;
csl_clock clkn;
clkn.add_output(clk);
clk.set_attr(clock);//clock is a tree drive
```

1.3.6.1 Connectivity object//incomplete explanation

signal, signal group, port, interface, unit, unit instance, signal concatenation, expr

signal type: port, wire, reg //, tristate

signal attribute: en, stall, pe, ps, ms, dec, clk, rst

1.3.6.2 Signal connected to Auto Connect Template with instances//rewrite this section

If a signal is connected to a module and the module has an instance list then traverse the instance list and connect signal to catch instance.

The CDOM is created with the different signal connection assignments. Then the OMDconnection engine connects the objects that are in the CDOM. Commands such as autoroute can be queued up in a CSL command queue and executed after all of the CDOM signals have been read in.
For each connect statement which is an auto conect add the signals to the global auto route map

TABLE 1.6

	Map value
*namespace_signal	vector <namespace_signal>

Then run the connect route.
namespace_signal map key for each map value and map module.

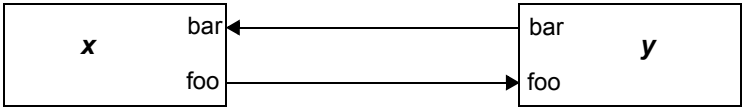
```
csl_signal instance_name.port(signal_name);
```

EXAMPLE :

DESCRIPTION :

//

FIGURE 1.14 Reversing portlists



CSL CODE:

```
bar.reverse;
foo.reverse;
```

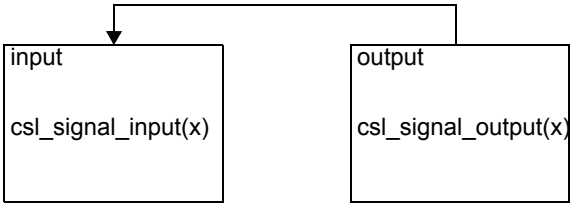
VERILOG CODE:

```
module x (a, b);
  output a;
  input b;
endmodule
module y (a, b);
  input a ;
  output b;
endmodule
```

1.3.6.3 Auto routing signals using csl_connect from one module to another in Verilog

The name of the module is optional when the csl_signal construct is used in verilog module.

FIGURE 1.15 CSL signal used inline in RTL code



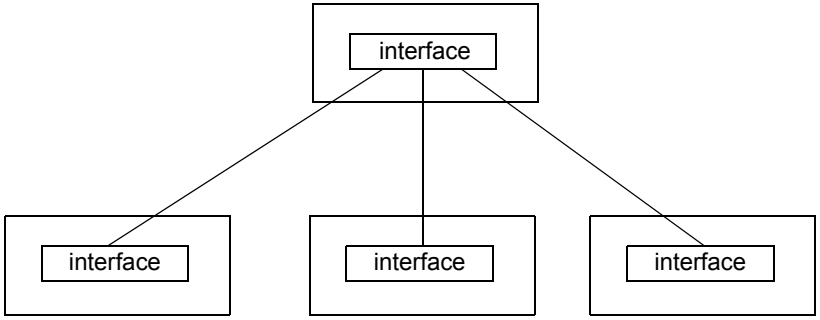
CSL treats design components as objects which can be copied and operated on. CSL objects include numeric values (constants), varables, bitranges, signals, portlists and modules Chapter 1.16 , “Interfaces used as objects” shows connected module hierarchies objects connected by interface objects.

EXAMPLE :

DESCRIPTION :

//

FIGURE 1.16 Interfaces used as objects



The signal connection which is to be autorouted is expressed by specifying the endpoints (e.g. top and bottom signals) and any formal to actual mappings,signal type information and signal attribute.

TABLE 1.7 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))
port select	w w0 (.x(p[7:4]))

1.4 CSL Auto Router Rules

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.

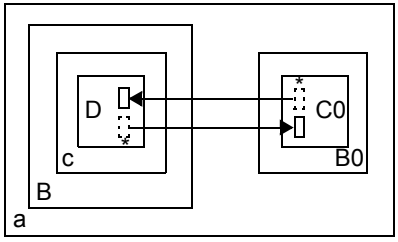
3.When connecting ports on different branches of the tree, the autorouter will infer ports for each level until it reaches the scope of the nearest common ancestor where it will infer wires.

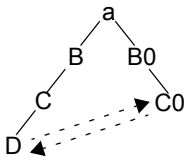
4.Each unit or instance should store a vector of numbers(according to the path from top to that of unit instance and considering the numbers given by the unique number build factory)

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

<ADD>

FIGURE 1.17





C0 & D point to each others' regs
autoconnect the two

</ADD>

1.5 CSL Auto Router Command Summary

Unit: autoconnect

```
unit_object_name.set_auto_router(ar_flag);
unit_object_name.auto_connect_filter(auto_connect_filter_enum);
connection_object_name0.connect(connection_object_name1);
connect(connection_object_name0,connection_object_name1);
connection_object_name.connect({.formal_connection_object_name(
actual_connection_object_name)}+);
autorouter_connect_bus_to_split_bus(ON|OFF);
```

1.6 CSL Auto Router Commands

unit_object_name.**set_auto_router(***ar_flag***);**

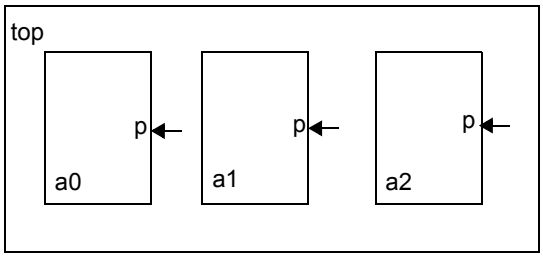
DESCRIPTION :

The *ar_flag* can be *on* or *off*.If *ar_flag* is *off*,the auto_router doesn't autoconnect the ports from that unit.If *ar_flag* is *on*,it does.By default the *ar_flag* is *on*.

EXAMPLE :

In the example below the unit named top contains three instances of the unit A.This unit can communicate to the outside through the input port p Because we have set to off the auto_router for a1,the auto_router will connect only the ports from a0 and a1.

FIGURE 1.18 Figure before autorouting



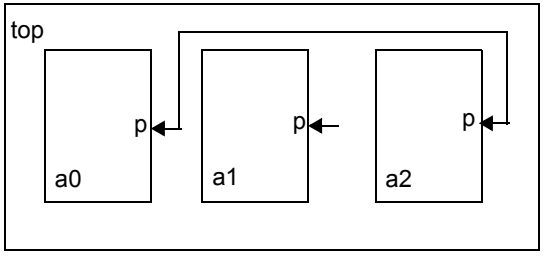
CSL CODE:

```
csl_unit top;
csl_unit a;
scope a{
  add_port(input,4,p);
}
scope top
{
  add_instance(a,a0);
  add_instance(a,a1);
  add_instance(a,a2);
  a1.set_auto_router(off);
autoconnect_width_inference();
}
```

VERILOG CODE:

//

FIGURE 1.19 Figure after autorouting



unit_object_name.**auto_connect_filter(**auto_connect_filter_enum**);**

DESCRIPTION :

Filters can be setup within a unit hierarchy and can be then passed as arguments to **auto_connect_filter()** to achieve more control or impose restrictions over the auto connect process

TABLE 1.8 Auto connect enums

Enum	Description
CSL_CONNECT BY_NAME	Connects interfaces, signal groups, signals and ports by name
CSL_CONNECT_PORTS_BY_NAME	Connects ports by name
CSL_CONNECT_SIGNALS_BY_NAME	Connect signals and signal groups by name (is this feasible?)
CSL_CONNECT_INTERFACES_BY_NAME	Connect interfaces by name
CSL_CONNECT_FUNCTION	Connects units, instances, interfaces, signal groups, signals by function connect()
VERILOG_CONNECT_BY_NAME	Connect only Verilog modules/instances/ports within the design
VERILOG_CONNECT_PORTS_BY_NAME	Connect only Verilog ports within the design
ALL_CONNECT_BY_NAME	Connect both CSL and Verilog connection objects
CREATE_NEW_ENDPOINT_PORTS	Connection objects which do not exist in a scope are infered to be ports. A new port objecft is generated in the scope. isn't this already being done above ?

EXAMPLE :

//

CSL CODE

//

VERILOG CODE

//

```
connection_object_name0.connect(connection_object_name1);
```

DESCRIPTION :

All elements in the table below are objects (except elements that start with “list_of”).
The signal connection is used to create connections between signals. The signals can be ports or local signals. Connections to ports create the formal to actual mapping between the the port name and signal in the upper level unit. Note that actual names are really expressions. The types of expressions supported for the actual expressions are as follows:

- signal
- signal with bit range
- expression (boolean operation on signals)
- concatentation
- signal name change
- constant

The table Table 1.9 contains the list of connection objects and the allowed connections between connection object types.

TABLE 1.9 Valid connections between connection objects

Nr	LHS \ RHS	1 s	2 sg	3 p	4 ifc	5 u	6 ui	7 sc	8 e
1	signal	X	-	X	X	X	X	X	X
2	signal group	-	X	-	X	X	X	X	-
3	port	X	-	X	X	X	X	X	X
4	interface	-	X	-	X	X*	X*	X*	-
5	unit	X	X	X	X	X	X	X	X
6	unit instance	X	X	X	X	X	X	X	X
7	signal concat	X	X	-	X	X	X	X	X
8	expr	-	-	-	-	-	-	-	-

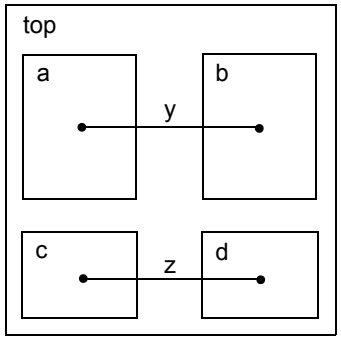
x? = to be discussed
Connection needs support for signal subranges and expressions
The following may not be needed anymore
Coonect can be used to ‘link’ signals, ports, groups of signals, or unit interfaces (when connect is apllied to unit names it’s still the interfaces of those units that get connected)

```
signal_name.connect(signal_name);
port_name.connect(port_name);
group_of_signals.connect(group_of_signals);
interface.connect(interface);
```

unit_name.connect(unit_name);

EXAMPLE :

FIGURE 1.20



//small description of the example

CSL CODE

```
csl_unit top;
csl_unit a, b, c, d;
top.add_instance(a,a1);
top.add_instance(b,b1);
b1.add_signal(y);
scope a1 {
  add_signal(y);
  y.connect(b1.y);
}
csl_unit c, d;
top.add_instance(c,c1);
top.add_instance(d,d1);
c1.add_signal(z);
d1.add_signal(t);
c1.z.connect(d1.t);
```

VERILOG CODE

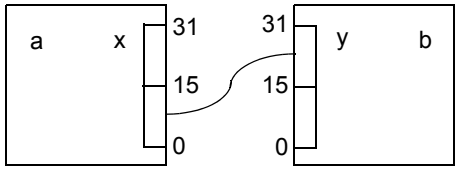
```
//AV
module top;
    wire y,z;
    ab a1(y);
    ab b1(y);
    c c1(z);
    d d1(z);
endmodule
module ab(y);
    inout y;
endmodule
```

```
module c(z);
    inout z;
endmodule
module d(z);
    inout z;
    wire t;
    assign t = z;
endmodule
```

.

EXAMPLE :

FIGURE 1.21



CSL CODE

```
csl_unit a, b;
a.add_signal(wire, 32, x);
```

SEE ALSO :

VERILOG CODE

```
//see code directory

module top;
  wire [31:0] x;
  a a0(.x(x));
  b b0(.x(x));
endmodule

module a;
 input [31:0] x;
endmodule

module b;
 output [31:0] y;
endmodule
```

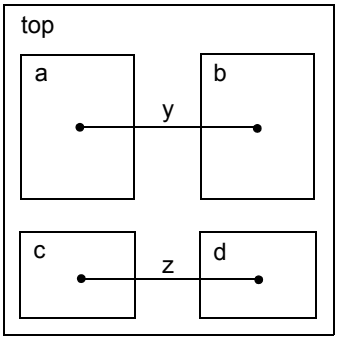
connect(*connection_object_name0,connection_object_name1***);**

DESCRIPTION :

Global connect function.

EXAMPLE :

FIGURE 1.22



//small description of the example

CSL CODE

```
csl_unit top;
csl_unit a, b, c, d;
top.add_instance(a,a1);
top.add_instance(b,b1);
a1.add_signal(y);
scope b1 {
  add_signal(y);
  connect(y,a1.y);
}
csl_unit c, d;
top.add_instance(c,c1);
top.add_instance(d,d1);
c1.add_signal(z);
d1.add_signal(t);
connect(c1.z,d1.t);
```

VERILOG CODE

```
//AV
module top;
    wire y,z;
    ab a1(y);
    ab b1(y);
    c c1(z);
```

```
    d d1(z);
endmodule
module ab(y);
    inout y;
endmodule
module c(z);
    inout z;
endmodule
module d(z);
    inout z;
    wire t;
    assign t = z;
endmodule
```

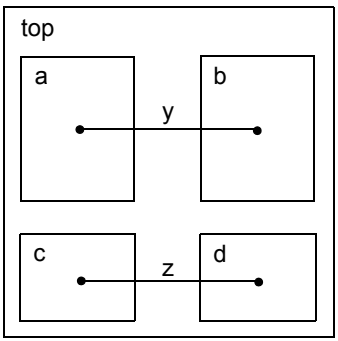
connection_object_name.**connect(**{.*formal_connection_object_name*(*actual_connection_object_name*)}+**);**

DESCRIPTION :

Connections are specified by *formal_connection_object_name* and *actual_connection_object_name* which can be ports or interfaces; *connection_object_name* can be a unit, instance or interface.

EXAMPLE :

FIGURE 1.23



//small description of the example

CSL CODE

```
csl_unit top;
csl_unit a, b, c, d;
top.add_instance(a,a1);
top.add_instance(b,b1);
b1.add_signal(y);
scope a1 {
  add_signal(y);
  connect(.y(b1.y));
}
csl_unit c, d;
top.add_instance(c,c1);
top.add_instance(d,d1);
c1.add_signal(z);
d1.add_signal(t);
c1.connect(.z(d1.t));
```

VERILOG CODE

```
//AV
module top;
    wire y,z;
    ab a1(y);
    ab b1(y);
    c c1(z);
    d d1(z);
endmodule
module ab(y);
    inout y;
endmodule
module c(z);
    inout z;
endmodule
module d(z);
    inout z;
    wire t;
    assign t = z;
endmodule
```

autorouter_connect_bus_to_split_bus(ON|OFF);

DESCRIPTION :

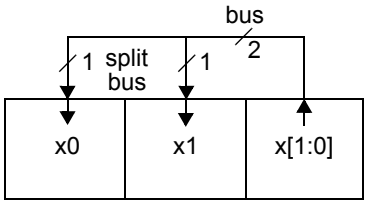
convert the bus into strings ??

EXAMPLE :

x[1:0]

x[1] becomes x_1 and x[0] becomes x_0

FIGURE 1.24



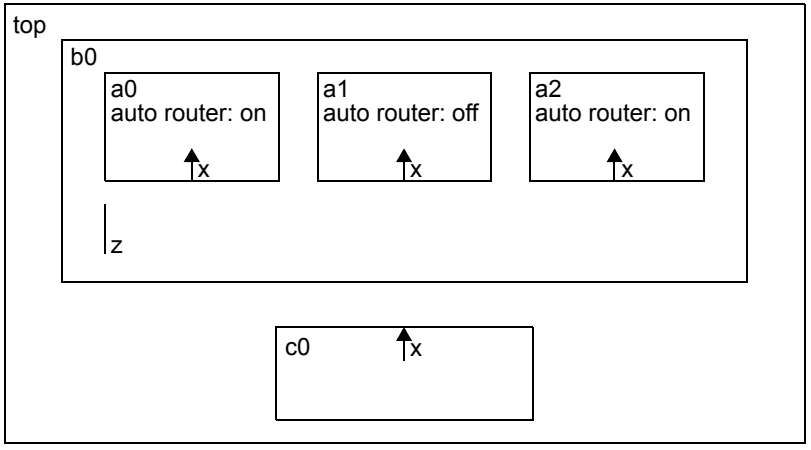
//commands

- turn {on/off} (port,signal,interface) inference - specified for or what can be inferred
- connect by (name,function) {on/off}

1.7 CSL Auto Router Examples

The first example focuses on using the autorouter on/off filter on unit instances to control port inference. Figure 1.25 shows how the autorouter can infer ports and connect them using name inference; the two instances which allow this will have the ports connected, while the unit which doesn't allow for this will not have it's ports connected by the auto router. Usually this is needed when a certain port is required to connect differently than other ports with which it shares the same name.

FIGURE 1.25 Initial layout before the auto router operation



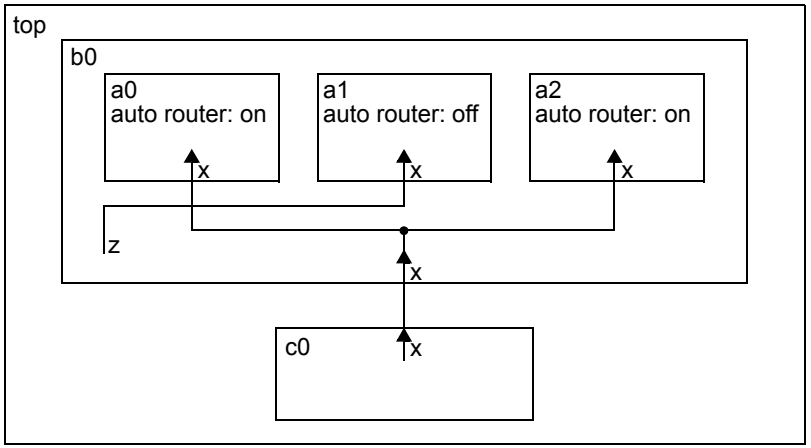
CSL CODE

```
csl_unit top,a,b,c;
c.add_port(output,x);
a.add_port(input,x);
scope b {
  /* no need to specifically turn on the auto router
  on a0 and a2 since it is on by default */
  add_instance(a,a0);
  add_instance(a,a1);
  a0.set_auto_router(off);
  add_instance(a,a2);
  add_signal(wire,z);
  z.connect(a0.x);
}
top.add_instance(b,b0);
top.add_instance(c,c0);
```

VERILOG CODE

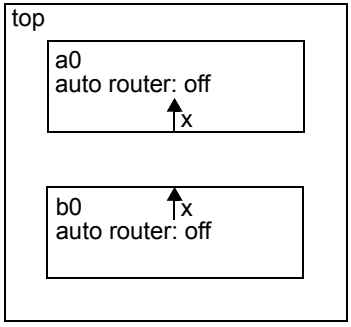
```
module top();
  wire y;
  b b0(.x(y));
  c c0(.x(y));
endmodule
module c(x);
  output x;
endmodule
module b(x);
  input x;
  wire z;
  a a0(.x(x));
  a a1(.x(z));
  a a2(.x(x));
endmodule
module a(x);
 input x;
endmodule
```

FIGURE 1.26 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 1.27 Initial two unconnected sibling units



When the connect function is called on connecting the two units in Figure 1.27 the connection is being made because no ports need to be inferred.

CSL CODE

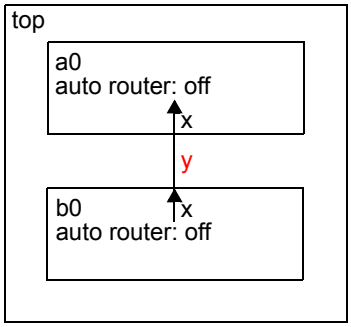
```
csl_unit top,a,b;
a.add_port(input,x);
b.add_port(output,x);
scope top {
  add_instance(a,a0);
  add_instance(b,b0);
  /* this works even if the auto router is set to off because
  the two units are siblings and no ports need to be inferred */
  a0.x.connect(b0.x);
}
```

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
```

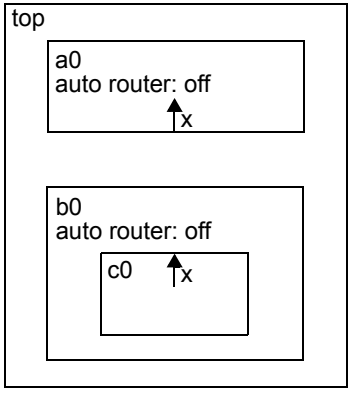
The connection is shown in Figure 1.28

FIGURE 1.28 Two connected sibling units



In Figure 1.29 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 1.29 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 :

FIGURE 1.30

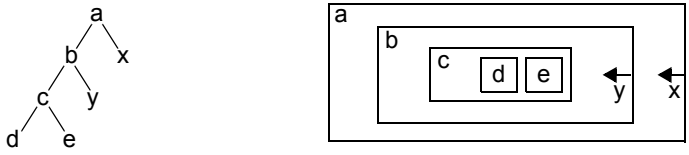


TABLE 1.10

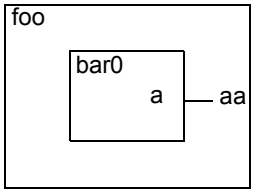
b	add output port x
c	wire x and connect to b b0(.x(x)) and d d0(.x(x))
d	add input port x
e	add input port x

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
```

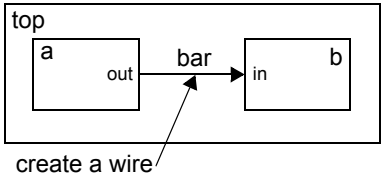
Signals can be auto routed between modules.

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 1.31 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
```

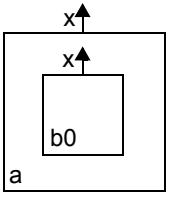
EXAMPLE :

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 1.32 Port to port autorouter connection



VERILOG CODE:

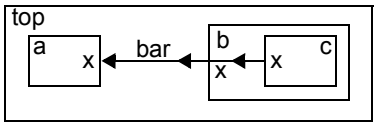
```
module a(x);
  output x;
  b b0(.x(x));
endmodule
module b(x);
  output x;
endmodule
```

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 1.33



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:

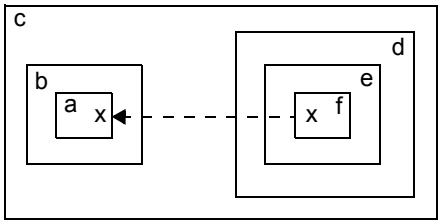
```
module top();
  wire x;
  a a(.x(x));
  b b(.x(x));
endmodule
module a(x);
  input x;
endmodule
module c(x);
  output x;
endmodule
module b(x);
  output x;
  c c(.x(x));
endmodule
```

EXAMPLE :

DESCRIPTION :

//

FIGURE 1.34 Figure before autorouting



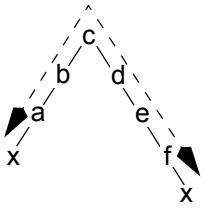
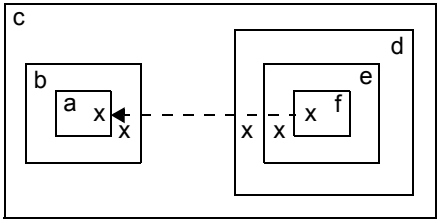


FIGURE 1.35 Figure after autorouting



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.

1.7.0.0.1 Design Hierarchy

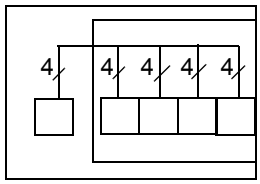
The CSL interconnect specification is used to define the design hierarchy for a chip design. A hierarchical file is a file which is part of the design hierarchy.The hierarchy file specifies the hierarchy for the chip. The design hierarchy files can be nested. The design hierarchy file lists the instance 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(1(b,c,d))
is equivalent to
a.connect(b)
a.connect(c)
a.connect(d)

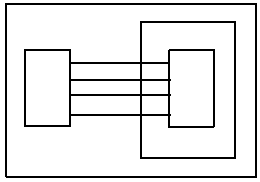


2. multi to multi

bus to bus connection

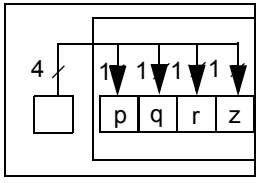
1(a,b,c).connect(list(p,q,r))

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