CHAPTER 1 CSL Auto Router

TABLE 1.1 Chapter Overview

1.1 CSL Auto Router Command Summary

1.1 CSL Auto Router Command Summary

```
(hid[.ps] | expression).connect_by_name((hid[.ps] | expression)
[,f2a_name]);
scope.connect_units(scope [,”f2a_prefix”]);
hid[.ps].connect_by_pattern(hid_pattern[.ps] [,f2a_name]);
```

1.2 CSL Auto Router Commands

[***CSL Auto Router Command Summary***]

.

```
(hid[.ps] | expression).connect_by_name((hid[.ps] | expression)
[,f2a_name]);
```

[***CSL Auto Router Command Summary***]

DESCRIPTION :

The method makes a connection between two endpoints. The complete path is given relative to the scope where the connect command is called:
hid = identifier(.identifier)*,
ps =part select,
expression = concatenation, replications, operators expression;
Last identifier in the hid is a connectivity object (port/signal/interface/sg). If the f2a option is used, there will be a port generated (not an interface.port, even if the connection is between ports from an interface) with the name "f2a_name" in each intermediate scope between the two connection end-points. The names of the generated ports may come in conflict with the names of the existing ports, in which case an error is shown.
The optional parameter is also explained on ***F2A option***.

TABLE 1.2 Possible connections for *connect_by_name()* method

RHS / LHS	port	port.ps	sig	sig.ps	ifc	signal group	op_ expr	cc_expr _var	cc_expr _num	rep_exp r_var	rep_exp r_num
port	L	L	L	L	I	I	L	L	L	L	L
port.ps	L	L	L	L	I	I	L	L	L	L	L
signal	L	L	I	I	I	I	L	L	I	L	I
signals.ps	L	L	I	I	I	I	L	L	I	L	I
interface	I	I	I	I	L	L	I	I	I	I	I
signal group	I	I	I	I	L	I	I	I	I	I	I
op_expr	I	I	I	I	I	I	I	I	I	I	I
cc_expr_v ar	L	L	L	L	I	I	L	L	L	L	L
cc_expr_n um	I	I	I	I	I	I	I	I	I	I	I
rep_expr_ var	I	I	I	I	I	I	I	I	I	I	I
rep_expr_ num	I	I	I	I	I	I	I	I	I	I	I

Where : L =legal case;
I =illegal case;

LHS =Left hand side;
RHS = Right hand side;
ps =part select;
op_expr = operators expression;
cc_expr_var = concatenation expression with variables (ports,signals);
cc_expr_num =concatenation expression with numbers;
rep_expr_var =replication expression with variables (ports,signals);
rep_expr_num=replication expression with numbers;

When *connect_by_name()* command is used for connect two ports can appear an error because of ports directions and ports positions. The following table show the rigth directions for the ports that will be connected.

TABLE 1.3 Possible connections between two ports

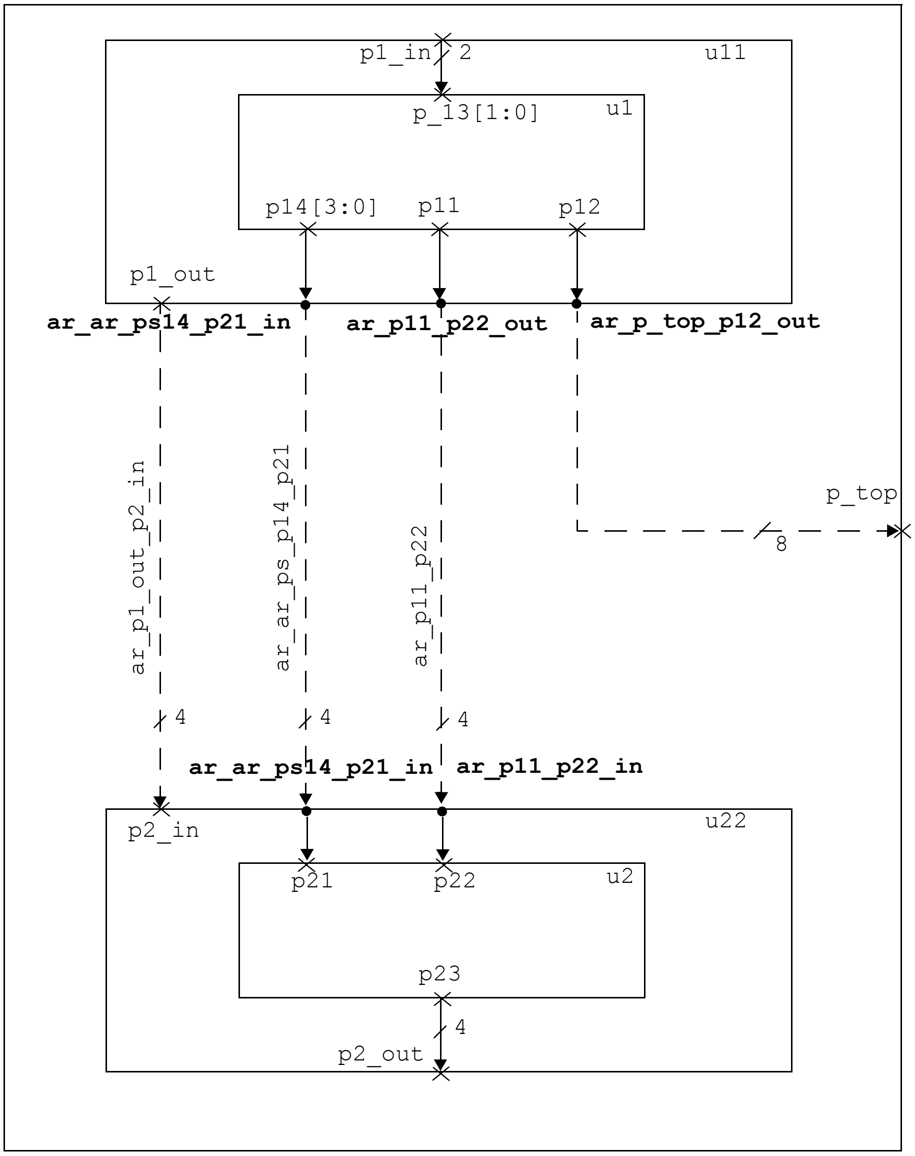
possition / direction	parent - child	child - parent	child -child
input - input	L	L	I
input - output	I	I	L
output - input	I	I	L
output - output	L	L	I

L =legal case;
I =illegal case;

EXAMPLE :

In this example the command connect_by_name() is used for ports connections. This shows the legal cases for ports connections, the ports and signals created by auto router.

FIGURE 1.1 Auto router connections



NOTE: On the figure, the black dots show the ports created by auto router and the names are bolded. The x indicate the ports declared by user.

NOTE: The signals created by auto router are dashed and the names are vertical. The arrows show the directions.

CSL CODE

```
csl_unit u1{
  csl_port p11(output,4);
  csl_port p12(output,8);
  csl_port p13(input,16);
  csl_port p14(output,7);
  u1(){
}
};
csl_unit u2{
  csl_port p21(input,4);
  csl_port p22(input,4);
  csl_port p23(output,4);
  u2(){
}
};
csl_unit u11{
  csl_port p1_in(input,2);
  csl_port p1_out(output,4);
  u1 u1_i;
  u11(){}
};
csl_unit u22{
  csl_port p2_in(input,4);
  csl_port p2_out(output,4);
  u2 u2_i;
  u22(){}
};
csl_unit top{
  csl_port p_top(output,8);
  u11 u11_i;
  u22 u22_i;
  top(){
    u11_i.p1_out.connect_by_name(u22_i.p2_in);      //c-c connection
    u11_i.u1_i.p11.connect_by_name(u22_i.u2_i.p22); //c-c connection
    p_top.connect_by_name(u11_i.u1_i.p12);          //p-c connection
    u22_i.u2_i.p23.connect_by_name(u22_i.p2_out);   //c-p connection
    u11_i.p1_in.connect_by_name(u11_i.u1_i.p13[1:0]);    //p-c with ps
```

```
      u11_i.u1_i.p14[3:0].connect_by_name(u22_i.u2_i.p21); //c-c with ps
    }
  };
```

VERILOG CODE

```
  module u1(p11,
            p12,
            p13,
            p14,
            ar_ps_p13,    // port created by auto router
            ar_ps_p14);   // port created by auto router
    input [16 - 1:0] p13;
    input [1:0] ar_ps_p13; //port used to connect a part select
    output [4 - 1:0] p11;
    output [8 - 1:0] p12;
    output [7 - 1:0] p14;
    output [3:0] ar_ps_p14;        //port used to connect a part select
    assign   ar_ps_p13 = p13[1:0]; // assigned by auto router
    assign   ar_ps_p14 = p14[3:0]; // assigned by auto router
    `include "u1.logic.v"
  endmodule
  module u2(p21,
            p22,
            p23);
    input [4 - 1:0] p21;
    input [4 - 1:0] p22;
    output [4 - 1:0] p23;
    `include "u2.logic.v"
  endmodule
  module u11(p1_in,
             p1_out,
             ar_p11_p22_out,        //created by auto router
             ar_p_top_p12_out,      //created by auto router
             ar_ar_ps_p14_p21_out); //created by auto router
    input [2 - 1:0] p1_in;
    output [4 - 1:0] p1_out;
    output [4 - 1:0] ar_p11_p22_out;
    output [8 - 1:0] ar_p_top_p12_out;
    output [4 - 1:0] ar_ar_ps_p14_p21_out;
    u1 u1_i(.ar_ps_p13(p1_in),
            .ar_ps_p14(ar_ar_ps_p14_p21_out),
         .p11(ar_p11_p22_out),
         .p12(ar_p_top_p12_out));
  `include "u11.logic.v"
endmodule
module u22(p2_in,
          p2_out,
          ar_p11_p22_in,
          ar_ar_ps_p14_p21_in);
  input [4 - 1:0] p2_in;
  input [4 - 1:0] ar_p11_p22_in;
  input [4 - 1:0] ar_ar_ps_p14_p21_in;
  output [4 - 1:0] p2_out;
  u2 u2_i(.p21(ar_ar_ps_p14_p21_in),
         .p22(ar_p11_p22_in),
         .p23(p2_out));
  `include "u22.logic.v"
endmodule
module top(p_top);
  output [8 - 1:0] p_top;
  wire [4 - 1:0] ar_p1_out_p2_in;
  wire [4 - 1:0] ar_p11_p22;
  wire [4 - 1:0] ar_ar_ps_p14_p21;
  u11 u11_i(.ar_ar_ps_p14_p21_out(ar_ar_ps_p14_p21),
           .ar_p11_p22_out(ar_p11_p22),
           .ar_p_top_p12_out(p_top),
           .p1_out(ar_p1_out_p2_in));
  u22 u22_i(.ar_ar_ps_p14_p21_in(ar_ar_ps_p14_p21),
           .ar_p11_p22_in(ar_p11_p22),
           .p2_in(ar_p1_out_p2_in));
  `include "top.logic.v"
endmodule
```

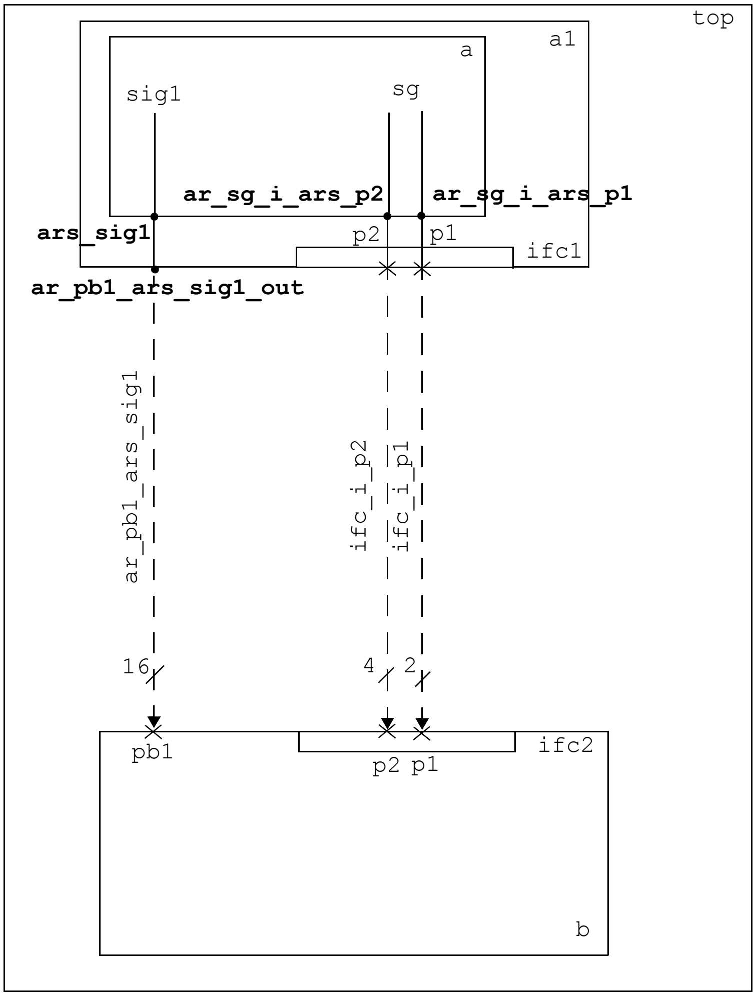
NOTE: 1) When the connect_by_name() command is used to connect two interfaces, the ports from interfaces which will be connected should have the same name and width.

NOTE: 2) When the connect_by_name() command is used to connect an interface with a signal group, the ports from interface should have the same name and width with the signals from signal group.

EXAMPLE :

In this example *connect_by_name()* command is used to connect the interface with interface, signal group with interface and signal with port.

FIGURE 1.2 Auto router connections and the created ports and signals.



NOTE: On the figure, the black dots show the ports created by auto router and the names are bolded. The x indicate the ports declared by user.

NOTE: The signals created by auto router are dashed and the names are vertical. The arrows show the directions.

CSL CODE

```
csl_interface ifc1{
  csl_port p1(output,8);
  csl_port p2(output,4);
  ifc1(){}
};
csl_interface ifc2{
  csl_port p1(input,8);
  csl_port p2(input,4);
  ifc2(){}
};
csl_signal_group sg{
  csl_signal p1(8);
  csl_signal p2(4);
  sg(){}
};
csl_unit a{
  csl_signal sig1(16);
  sg sg_i;
  a(){}
};
csl_unit b{
  csl_port pb1(input,16);
  ifc2 ifc_i;
  b(){}
};
csl_unit a1{
  ifc1 ifc_i;
  a a_i;
  a1(){}
};
csl_unit top{
  a1 a1_i;
  b b_i;
  top(){
    a1_i.a_i.sig1.connect_by_name(b_i.pb1);  // signal-port
    a1_i.a_i.sg_i.connect_by_name(a1_i.ifc_i); //signal group - inter-
face
```

```
      a1_i.ifc_i.connect_by_name(b_i.ifc_i); //interface - interface
    }
  };
```

VERILOG CODE

```
module a(ars_sig1,           //created by AR for signal sig1 connection
         arifc_sg_i_ars_p1, //created by AR for signal p1 from sg
                             //connection
         arifc_sg_i_ars_p2); //created by AR for signal p2 from sg
                             //connection

  output [16 - 1:0] ars_sig1;
  output [8 - 1:0] arifc_sg_i_ars_p1;
  output [4 - 1:0] arifc_sg_i_ars_p2;
  wire [16 - 1:0] sig1;
  wire [8 - 1:0] sg_i_p1;
  wire [4 - 1:0] sg_i_p2;
  assign   ars_sig1 = sig1;
  assign   arifc_sg_i_ars_p1 = sg_i_p1;
  assign   arifc_sg_i_ars_p2 = sg_i_p2;
  `include "a.logic.v"
endmodule
module b(pb1,
         ifc_i_p1,
         ifc_i_p2);
  input [16 - 1:0] pb1;
  input [8 - 1:0] ifc_i_p1;
  input [4 - 1:0] ifc_i_p2;
  `include "b.logic.v"
endmodule
module a1(ifc_i_p1,
          ifc_i_p2,
          ar_pb1_ars_sig1_out);
  output [8 - 1:0] ifc_i_p1;
  output [4 - 1:0] ifc_i_p2;
  output [16 - 1:0] ar_pb1_ars_sig1_out;
  a a_i(.arifc_sg_i_ars_p1(ifc_i_p1),
        .arifc_sg_i_ars_p2(ifc_i_p2),
        .ars_sig1(ar_pb1_ars_sig1_out));
  `include "a1.logic.v"
endmodule
```

```
module top();
  wire [16 - 1:0] ar_pb1_ars_sig1; //signal created by AR
  wire [8 - 1:0] ifc_i_p1;
  wire [4 - 1:0] ifc_i_p2;
  a1 a1_i(.ar_pb1_ars_sig1_out(ar_pb1_ars_sig1),
          .ifc_i_p1(ifc_i_p1),
          .ifc_i_p2(ifc_i_p2));
  b b_i(.ifc_i_p1(ifc_i_p1),
        .ifc_i_p2(ifc_i_p2),
        .pb1(ar_pb1_ars_sig1));
  `include "top.logic.v"
endmodule
```

EXAMPLE :

The example contains a set of interfaces and a spreadsheet is used to determine the units they were instantiated in and the connections between them. A script then adds these interfaces to the right units, reverses the interfaces used in the LHS for the SS connections and then makes the actual connections using the *connect_by_name()* method.

CSL CODE

```
// interfaces
csl_interface fabric_ifc {
   csl_port addr  (output, 18);
   csl_port data  (output, 32);
   csl_port nid  (output,  4);
   csl_port ready (input,   1);
   csl_port reject(output,  1);
   csl_port type  (output,  4);
   csl_port valid (output,  1);

   fabric_ifc () {}
};

csl_interface cdrv_ifc {
   csl_port  cmd  (input    );    // emac rx ix bus
   csl_port  data (input, 32);
   csl_port  ready(input    );
   csl_port  valid(input    );
};

csl_interface sb_ifc {
   csl_port start(output), busy(input);
   sb_ifc () {}
};

csl_interface sber_ifc {
   csl_port start(output), busy(input), empty(output), result(output);
   sber_ifc () {}
};

csl_interface sbhp_ifc {
   csl_port start(input), busy(output), hit(output), port(output) ;
   sbhp_ifc () {}
};

csl_interface vrr_ifc {
   csl_port valid(output), rd(input), result(output, 32);
   vrr_ifc () {}
};

csl_interface vsr_ifc {
   csl_port read_response_valid(output), fabric_start(input),
read_resonse(output, 32);
   vsr_ifc () {}
};

csl_interface adrv_ifc {
   csl_port addr(output,18), data(output,32), ready(input), valid(out-
put);
   adrv_ifc () {}
};

csl_interface adwr_ifc {
   csl_port addr(output,18), data(output,32), wr(output), rd(output);
   adwr_ifc () {}
};

csl_interface dw_ifc {
   csl_port data(output,32), wr(input);
   dw_ifc () {}
};
```

```
csl_interface adw_ifc {
   csl_port addr(output,18), data(output,32), wr(input);
   adw_ifc () {}
};


csl_interface malt_ifc {
   csl_port busy(output), hit(output), port(output), start(input);
   malt_ifc () {}
};


csl_interface ram_ifc {
  csl_port address(input, 11);
  csl_port byteena(input, 4);
  csl_port clock(input, 1);
  csl_port data(input, 32);
  csl_port enable(input, 1);
  csl_port q(output, 16);
  csl_port wren(input, 1);
  ram_ifc () {}
};

csl_interface nios_tcm_ifc {
   csl_port address(output, 18);
   csl_port byteena(output,  4);
   csl_port chipsel(output,  1);
   csl_port enable (output,  1);
   csl_port dout   (input , 32);
   csl_port wren   (output,  1);
   csl_port din    (output, 32);
   nios_tcm_ifc () {}
};

csl_interface nios_cii_ifc {
   csl_port a     (output,  1);
   csl_port b     (output,  1);
   csl_port c     (output,  1);
   csl_port clk_en  (output,  1);
   csl_port clk     (output,  1);
   csl_port dataa   (output, 32);
   csl_port datab   (output, 32);
   csl_port done    ( input,  1);
   csl_port n     (output,  8);
   csl_port readra  ( input, 32);
   csl_port readrb  ( input, 32);
   csl_port reset   (output,  1);
   csl_port result  ( input, 32);
   csl_port start   (output,  1);
   csl_port writerc ( input, 32);
   nios_cii_ifc () {}
};

csl_interface nios_int_ifc {
   csl_port irq_i_to_the_interrupt_0(input);
   csl_port irq_i_to_the_interrupt_1(input);
   csl_port irq_i_to_the_interrupt_2(input);
   csl_port irq_i_to_the_interrupt_3(input);
   nios_int_ifc () { }
};

// units, interfaces instantiated in the units and the connections
between the interfaces generated with the script

csl_unit pie_aud_nq1 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_aud_nq1 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit pie_ex_dq1 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_ex_dq1 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};
```

```
csl_unit pie_aud_dq1 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_aud_dq1 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit inst_tcm {
  inst_tcm () {
  }
};

csl_unit doorbell {
  vrr_ifc vrr; // from ifc
  dw_ifc dw; // to ifc
  doorbell () {
    dw.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit fabric_drop {
  dw_ifc db_dw; // from ifc
  dw_ifc dreg_dw; // from ifc
  adw_ifc maclu_adw; // from ifc
  adw_ifc qm_adw; // from ifc
  adwr_ifc adwr; // from ifc
  fabric_drop () {
  }
};

csl_unit fabric_interface {
  fabric_interface () {
  }
};

csl_unit pie_eth_nq01 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_eth_nq01 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit qm {
  adw_ifc dw; // to ifc
  qm () {
    dw.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit fabric_add {
  fabric_add () {
  }
};

csl_unit put_dma {
  cdrv_ifc cdrv; // from ifc
  sb_ifc sb; // to ifc
  put_dma () {
    sb.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit sram0 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  sram0 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit pie_eth_dq1 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  fabric_dma fabric_dma0  ;
  put_dma put_dma0  ;
  fabric_dma fabric_dma1  ;
  fabric_interface fabric_interface  ;
  fabric_drop fabric_drop  ;
  qm qm  ;
  nios nios  ;
  arb arb  ;
  maclu maclu  ;
  inst_tcm inst_tcm  ;
  cil cil  ;
  doorbell doorbell  ;
  reg reg  ;
  data_tcm data_tcm  ;
  put_dma put_dma1  ;
  fabric_add fabric_add  ;
  pie_eth_dq1 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit ipc {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  ipc () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit nios {
  nios () {
  }
};

csl_unit pie_ex_nq1 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_ex_nq1 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit pie_eth_nq11 {
  fabric_ifc fab_add; // from ifc
  fabric_ifc fab_drop; // to ifc
  pie_eth_nq11 () {
    fab_drop.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit cil {
  sb_ifc sb; // from ifc
  sb_ifc sbfd; // from ifc
  malt_ifc malt; // to ifc
  vrr_ifc vrr; // to ifc
  cdrv_ifc cdrv; // to ifc
  cil () {
    malt.reverse(); // this is an endpoint so reverse the interface
    vrr.reverse(); // this is an endpoint so reverse the interface
    cdrv.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit fabric_dma {
  sb_ifc sbfd; // to ifc
  fabric_dma () {
    sbfd.reverse(); // this is an endpoint so reverse the interface
  }
};



csl_unit reg {
  dw_ifc dw; // to ifc
  reg () {
    dw.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit arb {
  adwr_ifc adwr; // to ifc
  arb () {
    adwr.reverse(); // this is an endpoint so reverse the interface
  }
};
```

```
csl_unit maclu {
  malt_ifc malt; // from ifc
  adw_ifc adw; // to ifc
  maclu () {
    adw.reverse(); // this is an endpoint so reverse the interface
  }
};

csl_unit data_tcm {
  data_tcm () {
  }
};

csl_unit proc_ring {
  pie_eth_nq11 pie_eth_nq11  ;
  pie_eth_nq01 pie_eth_nq01  ;
  pie_ex_nq1 pie_ex_nq1  ;
  pie_aud_dq1 pie_aud_dq1  ;
  ipc ipc  ;
  pie_ex_dq1 pie_ex_dq1  ;
  sram0 sram0  ;
  pie_aud_nq1 pie_aud_nq1  ;
  pie_eth_dq1 pie_eth_dq1  ;
  proc_ring () {
    pie_eth_dq1.fab_add.connect_by_name(pie_eth_nq01.fab_drop
,rn1_rn2);
    pie_eth_nq01.fab_add.connect_by_name(pie_eth_nq11.fab_drop
,rn2_rn3);
    pie_eth_nq11.fab_add.connect_by_name(pie_aud_nq1.fab_drop
,rn3_rn4);
    pie_aud_nq1.fab_add.connect_by_name(pie_aud_dq1.fab_drop
,rn4_rn5);
    pie_aud_dq1.fab_add.connect_by_name(ipc.fab_drop ,rn5_rn6);
    ipc.fab_add.connect_by_name(pie_ex_nq1.fab_drop ,rn6_rn7);
    pie_ex_nq1.fab_add.connect_by_name(pie_ex_dq1.fab_drop ,rn7_rn8);
    pie_ex_dq1.fab_add.connect_by_name(sram0.fab_drop ,rn8_rn9);
    sram0.fab_add.connect_by_name(pie_eth_dq1.fab_drop ,rn9_rn1);
    pie_eth_dq1.cil.sb.connect_by_name(pie_eth_dq1.put_dma0.sb
,cil_pdma0);
```

```
    pie_eth_dq1.cil.sb.connect_by_name(pie_eth_dq1.put_dma1.sb
,cil_pdma1);
    pie_eth_dq1.maclu.malt.connect_by_name(pie_eth_dq1.cil.malt
,maclu_cil);
    pie_eth_dq1.doorbell.vrr.connect_by_name(pie_eth_dq1.cil.vrr
,db_cil);
    pie_eth_dq1.put_dma0.cdrv.connect_by_name(pie_eth_dq1.cil.cdrv
,pdma0_cil);
    pie_eth_dq1.put_dma1.cdrv.connect_by_name(pie_eth_dq1.cil.cdrv
,pdma1_cil);
    pie_eth_dq1.cil.sbfd.connect_by_name(pie_eth_dq1.fabric_dma0.sbfd
,cil_fdma0);
    pie_eth_dq1.cil.sbfd.connect_by_name(pie_eth_dq1.fabric_dma1.sbfd
,cil_fdma1);
    pie_eth_dq1.fabric_drop.db_dw.connect_by_name(pie_eth_dq1.door-
bell.dw ,fd_db);
   pie_eth_dq1.fabric_drop.dreg_dw.connect_by_name(pie_eth_dq1.reg.dw
,fd_reg);

pie_eth_dq1.fabric_drop.maclu_adw.connect_by_name(pie_eth_dq1.maclu.ad
w ,fd_maclu);
    pie_eth_dq1.fabric_drop.qm_adw.connect_by_name(pie_eth_dq1.qm.dw
,fd_qm);
    pie_eth_dq1.fabric_drop.adwr.connect_by_name(pie_eth_dq1.arb.adwr
,fd_arb);
  }
};
```

```
scope.connect_units(scope [,"f2a_prefix"]);
```

[***CSL Auto Router Command Summary***]

DESCRIPTION :

A connection between 2 scopes. The complete path is given relative to the scope where the connect command is called. The connection is made between connectivity elements with the same name from the two scopes.
scope = identifier(.identifier)*
Last identifier in the scope sequence is a scope (unit/reg/rf/fifo/memory instance).
The f2a_prefix is applied only to the top connectivity elements used in the connection between the two scopes. For example, if there is an interface within an interface only the top interface will be prefixed.
The optional parameter is also explained on ***F2A prefix***.

TABLE 1.4 Possible connections for *connect_units()* method

	unit	memory	fifo	register	register file
unit	L	L	L	L	L
memory	L	L	L	L	L
fifo	L	L	L	L	L
register	L	L	L	L	L
register file	L	L	L	L	L

Where: L -legal;

EXAMPLE :

In the example there is a connection between c0 and c0.b0.a0 using the *connect_units()* method. The interfaces from the 2 units have the same name, ifc0, and this way the connection is made between them.

CSL CODE

```
csl_interface ifc {
  csl_port x(input);
  csl_port y(output);
  ifc () {}
};
csl_unit a {
  ifc ifc0;
  a () {}
};
csl_unit b {
  a a0;
  a a1;
  b () {}
};
csl_unit c {
  b b0;
  ifc ifc0;
  c () {}
};
csl_unit d {
  c c0;
  d () {
    c0.connect_units(c0.b0.a0);
  }
};
```

VERILOG CODE

```
module a(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
endmodule
module b(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
  a a0(.ifc0_x(ifc0_x),
       .ifc0_y(ifc0_y));
  a a1();
endmodule
module c(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
  b b0(.ifc0_x(ifc0_x),
       .ifc0_y(ifc0_y));
 endmodule
module d();
  c c0();
endmodule
```

```
hid[.ps].connect_by_pattern(hid_pattern[.ps] [,f2a_name]);
```

[***CSL Auto Router Command Summary***]

DESCRIPTION :

The method creates a connection between two endpoints. The hid is the path to the first endpoint relative to where the command is called. The second endpoint is given as a path pattern (Not the full path, but only a part of the path that has to be matched in the hierarchy).

hid = path to connectivity object;
hid_pattern = path pattern to connectivity object;
ps =part select;
The optional parameter is explained on ***F2A option***.

TABLE 1.5 Possible connections for *connect_by_pattern()* method

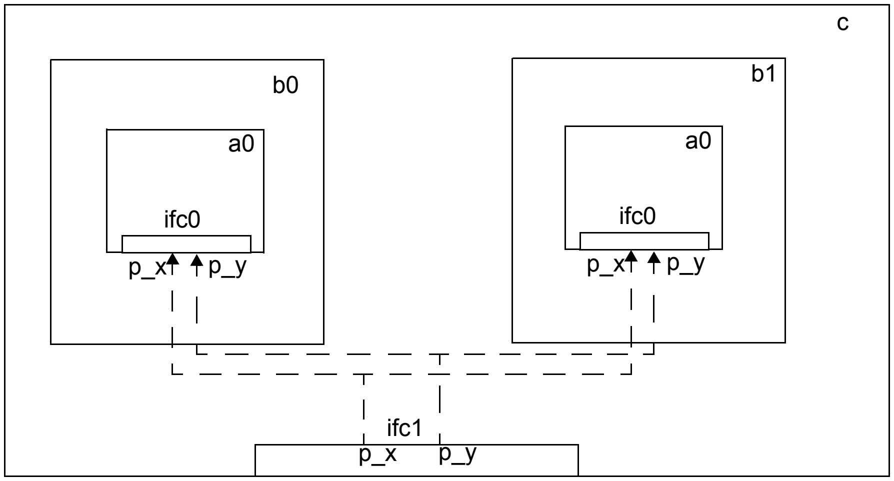
	port	signal	interface	signal group	expression*
port	L	L	I	I	I
signal	L	I	I	I	I
interface	I	I	L	L	I
signal group	I	I	L	I	I
expression*	I	I	I	I	I

expression = concatenation, replication, operators expression
L = legal
I = illegal

EXAMPLE :

The example below connects first ifc1 and ifc0 interfaces, from both b0 and b1 unit instances and then ifc2 and ifc0, from c0.b0, c0.b1, c1.b0 and c1.b1.

FIGURE 1.3



CSL CODE

```
 csl_interface ifc {
   csl_port p_x(input);
   csl_port p_y(input);
   ifc () {}
};

csl_unit a {
  ifc ifc0;
  a () {}
};
csl_unit b {
  a a0;
  b () {}
};
csl_unit c {
  b b0;
  b b1;
  ifc ifc1;
  c () {
// 2 matches : b0.a0.ifc0 and b1.a0.ifc0
    ifc1.connect_by_pattern(a0.ifc0);
  }
};
```

VERILOG CODE

```
module a(ifc0_p_x,
         ifc0_p_y);
  input ifc0_p_x;
  output ifc0_p_y;
endmodule
module b(ifc1_p_x,
         ifc1_p_y);
  input ifc1_p_x;
  output ifc1_p_y;
  a a0(.ifc0_p_x(ifc1_p_x),
       .ifc0_p_y(ifc1_p_y));
  endmodule
module c(ifc1_p_x,
         ifc1_p_y);
  input ifc1_p_x;
  output ifc1_p_y;
  b b0(.ifc1_p_x(ifc1_p_x),
       .ifc1_p_y(ifc1_p_y));
  b b1(.ifc1_p_x(ifc1_p_x),
       .ifc1_p_y(ifc1_p_y));
endmodule
```

EXAMPLE :

This example show the connections between clock ports, using *connect_by_pattern()* method.

CSL CODE:

```
csl_unit u_low{
  csl_port p11(input);
  csl_port p12(output);
  csl_port p_clk(input);   //clock port
  u_low(){}
};
csl_unit u1{
  csl_port p21(input);
  csl_port p22(output);
  u_low u_i1(.p11(p21),.p12(p22));  //instance of the low level unit
u1(){}
};
```

```
csl_unit u2{
  csl_port p31(input);
  csl_port p32(output);
  csl_port p_clk(input);            //clock port for u2;
  u1 u1_i1(.p21(p31),.p22(p32));   //instances of unit u1;
  u1 u1_i2(.p21(p31),.p22(p32));
  u1 u1_i3(.p21(p31),.p22(p32));
  u2(){}
};
csl_unit u_top{
  csl_signal s_clk;      //clock signal
  csl_port pt1(input);
  csl_port pt2(output);
  u2 u2i(.p31(pt1),.p32(pt2));  //instance of unit u2;
  u_top(){
    s_clk.connect_by_pattern(p_clk); //connect clock signal with clock
ports with name p_clk;
  }
};
```

VERILOG CODE:

```
module u_low(p11,
             p12,
             p_clk);
  input p11;
  input p_clk;
  output p12;
  `include "u_low.logic.v"
endmodule
module u1(p21,
          p22,
          ar_p_clk_s_clk);
  input p21;
  input [1 - 1:0] ar_p_clk_s_clk;  //port created by auto router
  output p22;
  u_low u_i1(.p11(p21),
             .p12(p22),
             .p_clk(ar_p_clk_s_clk));
  `include "u1.logic.v"
endmodule
module u2(p31,
          p32,
          p_clk,
          ar_p_clk_s_clk);
  input p31;
  input p_clk;
  input [1 - 1:0] ar_p_clk_s_clk;  //port created by auto router
  output p32;
  u1 u1_i1(.ar_p_clk_s_clk(ar_p_clk_s_clk),
           .p21(p31),
           .p22(p32));
  u1 u1_i2(.ar_p_clk_s_clk(ar_p_clk_s_clk),
           .p21(p31),
           .p22(p32));
  u1 u1_i3(.ar_p_clk_s_clk(ar_p_clk_s_clk),
           .p21(p31),
           .p22(p32));
  `include "u2.logic.v"
endmodule
module u_top(pt1,
             pt2);
  input pt1;
  output pt2;
  wire s_clk;
  u2 u2i(.ar_p_clk_s_clk(s_clk),
         .p31(pt1),
         .p32(pt2),
         .p_clk(s_clk));
  `include "u_top.logic.v"
endmodule
```

NOTE:

F2A option
Connections are made through formal to actual connection pairs. The autorouter infers automatically the actual part of a connection unless the user specifies it with the f2a option. f2a_name is the name that will be used to create intermmediate ports/interfaces/signals/signal groups according to the types of connection endpoints involved.

F2A prefix
When specified it will prefix only to the top connectivity elements with the f2a_prefix.

EXAMPLE :

This is an example the optional parameter *f2a_name* is used.

CSL CODE

```
csl_interface ifc {
  csl_port x(input);
  csl_port y(output);
  ifc () {}
};

csl_unit a {
  ifc ifc0;
  a () {}
};

csl_unit b {
  a a0;
  b () {}
};

csl_unit c {
  b b0;
  c () {}
};

csl_unit d {
  c c0;
  ifc ifc1;
  d () {
// the generated ports will be p_a_x, p_a_y
    ifc1.connect_by_name(c0.b0.a0.ifc0,p_a);
  }
};
```

VERILOG CODE

```
module a(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
endmodule
module b(p_a_x,
         p_a_y);
  input p_a_x;
  output p_a_y;
  a a0(.ifc0_x(p_a_x),
       .ifc0_y(p_a_y));
endmodule
module c(p_a_x,
         p_a_y);
  input p_a_x;
  output p_a_y;
  b b0(.p_a_x(p_a_x),
       .p_a_y(p_a_y));
 endmodule
module d(ifc1_x,
         ifc1_y);
  input ifc1_x;
  output ifc1_y;
  c c0(.p_a_x(ifc1_x),
       .p_a_y(ifc1_y));
endmodule
```

EXAMPLE :

In this example is used the optional parameter *f2a_prefix.*

CSL CODE

```
csl_interface ifc {
  csl_port x(input);
  csl_port y(output);
  ifc () {}
};

csl_unit a {
  ifc ifc0;
  a () {}
};

csl_unit b {
  a a0;
  b () {}
};
```

```
csl_unit c {
  b b0;
  ifc ifc0;
  c () {}
};

csl_unit d {
  c c0;
  d () {
// connects the two ifc0 interfaces and adds “prefix” to the ports
    c0.connect_units(c0.b0.a0,"prefix");
  }
};
```

VERILOG CODE

```
module a(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
endmodule
module b();
  a a0();
endmodule
module c(ifc0_x,
         ifc0_y);
  input ifc0_x;
  output ifc0_y;
  b b0();
endmodule
module d();
  c c0();
 endmodule
```