PARALLEL RESISTANCE REDUCTION IN THE SPACE EXTRACTION PROGRAM

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 04-02
June 3, 2004

Last revision: June 7, 2004.

1. INTRODUCTION

The parallel resistance reduction in the *space* extraction program is a resistance network elimination method, which is done with the *space* function reducMaxParRes. The function removes large resistance elements that are short-circuited by a much lower resistance path. The ratio between ‘‘large’’ and ‘‘much lower’’ can be specified with parameter "max_par_res". The "Space User’s Manual" gives as suggestion a value of 25 for this ratio. The "Space User’s Manual" names this reduction (in section 2.8.4) the max_par_res heuristic. This heuristic prevents the occurrence of high-ohmic shunt paths between two nodes. And the manual notes, that the order of elimination is arbitrary. Note that also large negative resistors can be removed with function reducMaxParRes.

The reducMaxParRes function is only called by function reducGroupI. It is called, after calls to functions reducArtDegree, reducMinRes and reducMinSepRes. Function reducGroupI is only called by function readyGroup. And function readyGroup is called, when the last node of a node group becomes ready.

Function reducMaxParRes is only done, when variable optRes is true. Variable optRes is true in the extract pass, when interconnect and/or substrate resistance extraction is performed. Thus, function reducMaxParRes is only done in the extract pass.

Function reducMaxParRes is also only done, when max_par_res ≥ 0. Variable max_par_res is default -1, but can be set with parameter "max_par_res". Thus, the parallel resistance reduction is only done, when requested.

Function reducMaxParRes calls function min_R_path to find the minimum resistance path between nodes. Note that function findMaxRes is also using function min_R_path.

1.1 How does function min_R_path work?

See for the source code appendix A. Function min_R_path uses three node members (help, flag and flag3). Before calling min_R_path, these node members must be inited for the set nodes of the group, which must be evaluated. The help member is used to store the minimum resistance value and must be inited to -1 for each evaluated node. The flag member for each evaluated node must initial be 0. This member is set by min_R_path, when a minimum resistance path is found. The flag3 member must be set for each node, which must be evaluated. Note that the function can not find a minimum path via negative resistors (≤ 0 conductor values), or go to nodes connected with negative resistor.

1.2 How does function reducMaxParRes work?

See for the source code appendix A. Function reducMaxParRes can be divided in two parts. Part I makes cliques for non-area nodes and process these node cliques. Part II handles area nodes, which are connected by resistance elements and which are possibly not processed in part I.

Note that this function possibly remove resistance elements from the resistance network, but that the function shall never remove nodes from the network. Thus, the node group queue (qn) and the number of nodes (n_cnt) is unchanged. Function reducMaxParRes (in current implementation) needs only to be done, when there are more nodes (n_cnt > 1).

Note that reducMaxParRes uses a number of node members (area, flag2, flag3) and that it must also set/reset a number of node members (help, flag, flag3) for min_R_path.

1.2.1 Part I of function reducMaxParRes

Part I makes node cliques for all non-area nodes, which are not yet part of a clique before. At least one clique is made, when one of more non-area nodes exist in the node group. When all nodes are area nodes, only part II needs to be done.

Before using function min_R_path, the to evaluated set of nodes (a clique) must be initiated. Before starting part I, all flag2 and flag3 node members of the group are set to 0. Note that all node "flag" members of the group are already set to 0. Note that the flag2/flag3 members are set to 0, because they are never set before. Note that the flag2 members are only used in this part (part I) to flag that the node has already been evaluated (i.e. was in a clique). Note that the flag3 needs be set for each node of the current clique.

Thus, part I makes a node clique for the first non-area node (n) found. Each node, which is connected to node n by a resistor element, is put in the clique for node n. Thus, also area nodes can be put in the clique. Node member flag3 is set for each node in the clique. A node can only once be put in the clique queue (QC), because flag3 is tested. This test must be done, because there can be more resistor elements between two nodes (only when different resistor sorts are used, when resSortTabSize > 1).

After the clique is build, first all resistor elements between node "n" and all other clique nodes are evaluated, calling function min_R_path. After this first clique evaluation round, there comes another round for each other clique node. Before each clique round the node help members must be inited to -1. And after each clique round the node flag members must be reset to 0. Note that the r_flag is used by each round to speed-up the searching, because previous minimum resistance results are remembered by min_R_path. In each round is r_flag by the first call to min_R_path 0. See for example appendix B.

Function min_R_path returns the minimum resistance value found between two nodes of the clique. This value can be the value of the resistor element between the two nodes, when there is no smaller path available. If the value of the resistor element is greater than the found value multiplied by max_par_res, then the resistor element is deleted. Note that node pointers are compared, to exclude double calls to min_R_path for symmetrical cases. Note that for a resistor element with conductance value 0 the call/search is not done. This, because a divide by zero is not allowed. However, this element should be deleted, when a min_R_path exist. On the other side, the connections between the nodes of the group may not be broken.

After the node clique is processed, the node flag3 members are reset and the flag2 members are set. A node of a processed clique can be part of a second clique, but not be the first node.

After that, part I tries to make a new clique for a non-area node of which flag2 is not yet set.

1.2.2 Part II of function reducMaxParRes

In part II, only min_R_path for area nodes are evaluated. Because cliques with only area nodes are still uninvestigated now. Therefore, evaluate the resistances between the area node pairs. Note: Now, min_R_path will not find the minimum parallel path if this path is via a non-area node. However, in that case, the resistance between the two nodes will already have been computed correctly in the previous part.

First set for all area nodes the flag3. Note that for all other nodes of the group flag3 is 0. Note that flag2 is not used, each area node is done ones. Note that the evaluated set of area nodes is not a clique as in part I, because we don't know if there are resistor elements between these nodes.

In each round an area node is processed and the min_R_path determined. Before each round, all node help members are inited to -1. And after each round all node flag members are reset to 0. Thus, can possible in each round too large resistor elements between two area nodes be deleted.

Note that part II needs only to be done, when there are two or more area nodes. And these area nodes are directly connected too each other and have not been in the same clique of part I.

1.3 What is a clique?

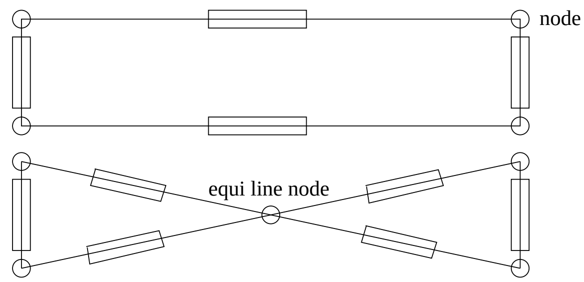
A clique is a set of nodes. The first node of the clique (the starting node) may not be an area node. All other nodes of the clique are connected to the first node by a resistor element. Thus, the clique is only one node level depth. However, when all nodes in the resistor network are connected to each other, then all nodes are part of the clique.

Note that all nodes are members of the same node group (and are all in the qn node-queue). However, there can be resistors of type 'S', which connects to nodes of other groups. These nodes may not be part of the clique. Note that resistor elements of type 'S' can only be substrate resistors. Thus, different substrate nodes are not part of the same group. These nodes can only become part of the same group, when they are connected via interconnect to each other. The clique queue is maintained by array QC and QC_cnt. Function putQueueClique (code not included) is used to put nodes in the clique. The function is also maintaining the size of the queue.

1.4 What is an area node?

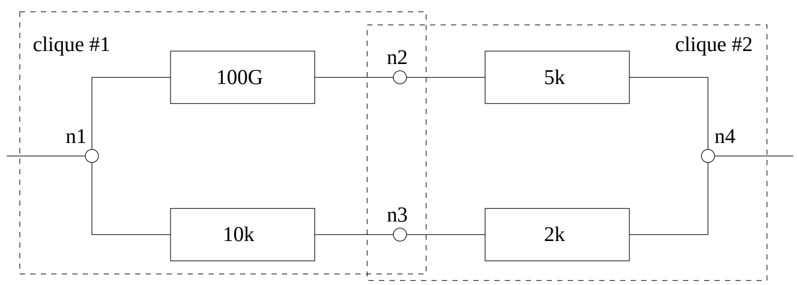
Each node has an area member, which flags, when a node is an area node. There are two possible types, ‘‘line’’ (area = 1) and ‘‘area’’ (area = 2) nodes. Real area nodes (area = 2) have higher priority than ‘‘line’’ nodes (see nodeJoin). Area nodes of type 2 are created, when there are equipotential area's. This happens, when the interconnect resistance of the area is zero. Area nodes of type 1 are only created, when parameter "equi_line_ratio"

is specified (value > 0). Note, type 1 creation can be put off with parameter "equi_line_area". Note that the *space* program can give statistics about equi line and area nodes with option **-i**.
The following figure gives an example of a created equi line node:



1.5 Comments

Because the node cliques are one level depth, function reducMaxParRes can not always remove too large resistors from the resistor network. Example:



When possible, first a node elimination round needs to be done.

1.6 Improvements

For the improvements see appendix C and D.

There where two main reasons to change function reducMaxParRes.
The first reason is, that for max_par_res < 1 the function always removes all resistors from the network and when no other path is found, sometimes it removes also the resistor for max_par_res ≡ 1.
The second reason is, that the compare of node pointers is a bad choice. Because node structures are arbitrary allocated. By the use of different extract passes, the resulting network (the extract result) may be each time different.

1.6.1 Improvement 1

Parameter max_par_res must be ≥ 1. Also for max_par_res ≡ 1 there can be a problem, therefor we add an arbitrary small value (1e-9) to the result. Thus, the element is only deleted, when another path is found.

1.6.2 Improvement 2
The node flag3 is set to 2, to flag that the node was already the start node of a clique round. Because the other node 'c_n' may not be a start node, this prevents symmetrical cases. Thus, the compare of node pointers is not more needed. And the extract result shall possibly not more be depending on the number of extract passes to be involved. This improvement is also done for part II.

1.6.3 Improvement 3
The group node count (n_cnt) must be ≥ 2, else the function is skipped.

1.6.4 Improvement 4
An area nodes counter is added (area_nodes). Thus, part II is only done, when there are at least 2 area nodes.

1.6.5 Improvement 5
The last node in a clique does not need to be done. Thus, we do only QC_cnt - 1 clique rounds.

1.6.6 Improvement 6
We use the clique queue also for part II. Thus, we don't need to evaluate all nodes of the group. We do only QC_cnt - 1 clique rounds. We init only the area nodes (in place of all nodes).

1.6.7 Improvement 7
We set all clique nodes with flag2, not only the not area nodes.

1.6.8 Improvement 8
We give now to function min_R_path the QC_cnt (in place of n_cnt). This is the size of the array, that we really need.

1.6.9 Improvement 9
Zero conductor values are now also processed. We take for that value an arbitrary large resistance value (1e99). When another minimum resistance path is found, the resistor element is now deleted. Note that we need improvement 10, to let it work.

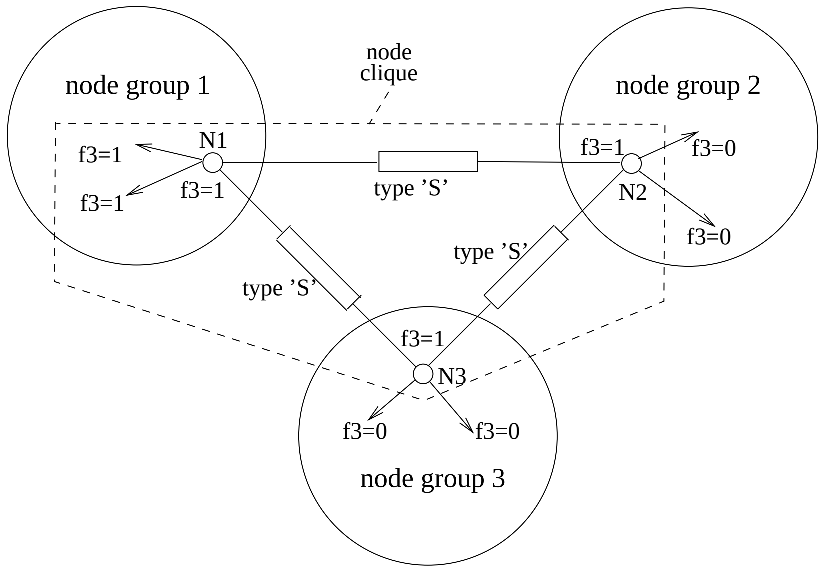
1.6.10 Improvement 10
The return value of function min_R_path is changed. The function does not more return an arbitrary large resistance value, when no resistance path could be found. Now, it returns a negative value (which is the initial set help value). Thus, the return value of function min_R_path is only valid, when greater than zero.

Note that this implementation is also a better choice for function findMaxRes. Maybe we need to inform Optem about this new choice?

1.7 Suggestions for improvements

Too large substrate resistors can not be removed with function reducMaxParRes. However, this can be made possible. Note, by the introduction of the special resistor type

’S’ is this functionality gone. For example:



To let it work, the nodes of the type ’S’ resistors must also be put in the clique. The flag3 node members must all be inited by function createNode, because it was only done for one group. Now, for n_cnt ≡ 1 must function reducMaxParRes also be done. Some ’S’ type resistors can possibly be deleted, because there is another minimum resistance path (see picture above). The ’S’ type nodes (the nodes in another group) need not self be done for min_R_path. They are skipped, when flag3 is set to 2. Note that the number of nodes in the clique can be more than n_cnt. Maybe we need to include new parameters (a separate ratio for substrate and a substrate res reduction enable). Maybe we must also take into account the resistors to node SUBSTR.

Note that in some cases, when there are substrate contacts with interconnect, the ’S’ type resistors are changed into ’G’ type resistors. In that case, they are already taken into account, and possibly deleted.

1.8 Test Results

I have tested the improvements with the "infineon/coilgen" layout (for max_par_res = 1). The extracted circuit of this layout consist out of two separate resistor networks. One for the interconnect part and one for the substrate part. Only implementing improvement 2 shows a complete different result. But both the old and new results seem to be wrong. Only implementing improvement 1 shows that the result is much better. Thus, the main reason for bad results is the delete of good resistors. The extraction result shows, that the substrate part is not reduced (see appendix E and F). I have tried to implement the substrate improvement suggestion. And that result removes from the substrate resistor network all negative resistors and other large resistors.

2. APPENDICES

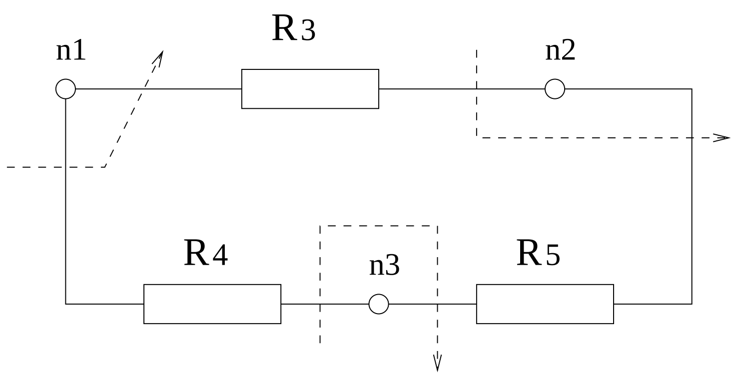
APPENDIX A -- File lump/reduc.c version 4.45

```
Private void reducMaxParRes (qn, n_cnt) node_t **qn; int n_cnt;
{
    int QC_cnt;
    node_t *QC[MAX_NODES];
    int i, j, k, l, r_flag;
    node_t *n, *c_n;
    element_t *con, *next;

    if (!optRes || max_par_res < 0) return;

    /* remove resistances that are
     * short-circuited by a much lower resistance path
     */
    for (i = 0; i < n_cnt; i++) {
        qn[i] -> flag2 = 0;   /* if flag2 == 1 clique already done */
        qn[i] -> flag3 = 0;   /* if flag3 == 1 part of current clique */
    }

    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (n -> area || n -> flag2) continue;

        /* First, search clique of n */

        QC_cnt = 0;
        n -> flag3 = 1;
        QC[ QC_cnt++ ] = n; // putQueueClique

        for (k = 0; k < resSortTabSize; k++) {
            for (con = n -> con[k]; con; con = NEXT (con, n)) {
                if (con -> type == 'S') continue;
                c_n = OTHER (con, n);
                if (c_n -> flag3 == 0) {
                    c_n -> flag3 = 1;
                    QC[ QC_cnt++ ] = c_n; // putQueueClique
                }
            }
        }

        /* Then, evaluate resistances between all pairs of nodes in the clique */

        for (l = 0; l < QC_cnt; l++) {
            n = QC[l];
            ASSERT (n -> flag3);

            for (j = 0; j < QC_cnt; j++) QC[j] -> help = -1;

            /* r_flag is used to speed-up the searching with min_R_path().
             * As long as we start searching from the same node, we can use
             * the previous results obtained so far.
             */
            r_flag = 0;
            for (k = 0; k < resSortTabSize; k++) {
                for (con = n -> con[k]; con; con = next) {
                    next = NEXT (con, n);
                    if (con -> type == 'S') continue;
                    c_n = OTHER (con, n);
```

```
                    if (c_n -> flag3 && (unsigned)n < (unsigned)c_n) {
                        /* Compare pointers to exclude symmetrical cases
                           (obtained by swapping n and c_n)
                           that would yield the same result. */
                        if (con -> val != 0) {
                            if (min_R_path (n, c_n, n_cnt, r_flag) * max_par_res
                                 < 1 / Abs (con -> val)) {
                                elemDel (con, k);
                            }
                            r_flag = 1;
                        }
                    }
                }
            }

            for (j = 0; j < QC_cnt; j++) QC[j] -> flag = 0;
        }

        for (l = 0; l < QC_cnt; l++) {
            QC[l] -> flag3 = 0;
            if (!QC[l] -> area) QC[l] -> flag2 = 1;
        }
    }

    /* Cliques with only area nodes are still uninvestigated now.
       Therefore, evaluate the resistances between the area node pairs.
       Now, min_R_path() will not find the minimum parallel path if
       this path is via a non-area node.  However, in that case,
       the resistance between the two nodes will already have
       been computed correctly in the previous part.
    */

    for (i = 0; i < n_cnt; i++) {
        if (qn[i] -> area) qn[i] -> flag3 = 1;
    }

    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (!n -> area) continue;

        for (j = 0; j < n_cnt; j++) qn[j] -> help = -1;

        r_flag = 0;
        for (k = 0; k < resSortTabSize; k++) {
            for (con = n -> con[k]; con; con = next) {
                next = NEXT (con, n);
                if (con -> type == 'S') continue;
                c_n = OTHER (con, n);
                if (c_n -> flag3 && (unsigned)n < (unsigned)c_n) {
                    if (con -> val != 0) {
                        if (min_R_path (n, c_n, n_cnt, r_flag) * max_par_res
                             < 1 / Abs (con -> val)) {
                            elemDel (con, k);
                        }
                        r_flag = 1;
                    }
                }
            }
        }

        for (j = 0; j < n_cnt; j++) qn[j] -> flag = 0;
    }
}
```

```
static node_t *frontNodes[MAX_NODES];

Private double min_R_path (ns, nt, n_cnt, doContinue)
node_t *ns, *nt;
int n_cnt;
int doContinue;   /* use previous results (ns must be unchanged) */
{
    static node_t *n;
    static int f_cnt;
    int i, f, k;
    double min;
    node_t * c_n;
    element_t * con;

    if (!doContinue) {
        n = ns;
        n -> help = 0;
        f_cnt = 0;
    }
    else if (nt -> flag) {
        return (nt -> help);
    }

    while (n && n != nt) {

        for (k = 0; k < resSortTabSize; k++) {
            for (con = n -> con[k]; con; con = NEXT (con, n)) {
                if (con -> type == 'S') continue;
                c_n = OTHER (con, n);
                if (c_n -> flag3 && con -> val > 0) {
                    min = 1 / con -> val + n -> help;
                    /* ASSERT (min > 0); */
                    if (c_n -> help < 0) {
                        frontNodes[ f_cnt++ ] = c_n;
                        c_n -> help = min;
                    }
                    else if (min < c_n -> help) {
                        c_n -> help = min;
                    }
                }
            }
        }

        if (f_cnt > 0) {
            min = frontNodes[f = 0] -> help;
            for (i = 0; ++i < f_cnt;) {
                if (frontNodes[i] -> help < min) min = frontNodes[f = i] -> help;
            }
            n = frontNodes[f];
            frontNodes[f] = frontNodes[--f_cnt];
            n -> flag = 1; /* minimum resistance path has been determined */
        }
        else
            n = NULL;
    }

    if (!n) return (1e+30); /* no path (along positive R's) */
    return (n -> help);
}
```

APPENDIX B -- Example of reducMaxParRes part I




```
n_cnt = 3; qn = { n1, n2, n3 };

part I:
    QC_cnt = 3; QC = { n1, n3, n2 }; // clique #1

    round 1:  n = n1;
            c_n = n3; // R4 > max_par_res * (R3 + R5) ?
            c_n = n2; // R3 > max_par_res * (R4 + R5) ?

    round 2:  n = n3;
            c_n = n1; // symm. case
            c_n = n2; // R5 > max_par_res * (R3 + R4) ?

    round 3:  n = n2;
            c_n = n1; // symm. case
            c_n = n3; // symm. case

part I:
    // All nodes are done, no second clique is needed.

    // Note that only one resistor can be deleted.
    // There must always be a resistor path between the nodes.
```

APPENDIX C -- File lump/reduc.c version 4.46

```
Private void reducMaxParRes (qn, n_cnt) node_t **qn; int n_cnt;
{
    int QC_cnt;
    node_t *QC[MAX_NODES];
    int i, j, k, l, r_flag, area_nodes;
    node_t *n, *c_n;
    element_t *con, *next;
    double min, res;

    if (!optRes || max_par_res < 1 || n_cnt < 2) return;

    /* remove resistances that are
     * short-circuited by a much lower resistance path
     */
    for (i = 0; i < n_cnt; i++) {
        qn[i] -> flag2 = 0;   /* if flag2 == 1 clique already done */
        qn[i] -> flag3 = 0;   /* if flag3 == 1 part of current clique */
    }

    area_nodes = 0;
    for (i = 0; i < n_cnt; i++) {
        n = qn[i];
        if (n -> area) { ++area_nodes; continue; }
        if (n -> flag2) continue;

        /* First, search clique of n */

        QC_cnt = 0;
        n -> flag3 = 1;
        QC[ QC_cnt++ ] = n; // putQueueClique

        for (k = 0; k < resSortTabSize; k++) {
            for (con = n -> con[k]; con; con = NEXT (con, n)) {
                if (con -> type == 'S') continue;
                c_n = OTHER (con, n);
                if (c_n -> flag3 == 0) {
                    c_n -> flag3 = 1;
                    QC[ QC_cnt++ ] = c_n; // putQueueClique
                }
            }
        }

        /* Then, evaluate resistances between all pairs of nodes in the clique */

        for (l = 0; l < QC_cnt-1; l++) {
            n = QC[l];
            n -> flag3 = 2;

            for (j = 0; j < QC_cnt; j++) QC[j] -> help = -1;

            /* r_flag is used to speed-up the searching with min_R_path().
             * As long as we start searching from the same node, we can use
             * the previous results obtained so far.
             */
            r_flag = 0;
            for (k = 0; k < resSortTabSize; k++) {
                for (con = n -> con[k]; con; con = next) {
                    next = NEXT (con, n);
                    if (con -> type == 'S') continue;
                    c_n = OTHER (con, n);
```

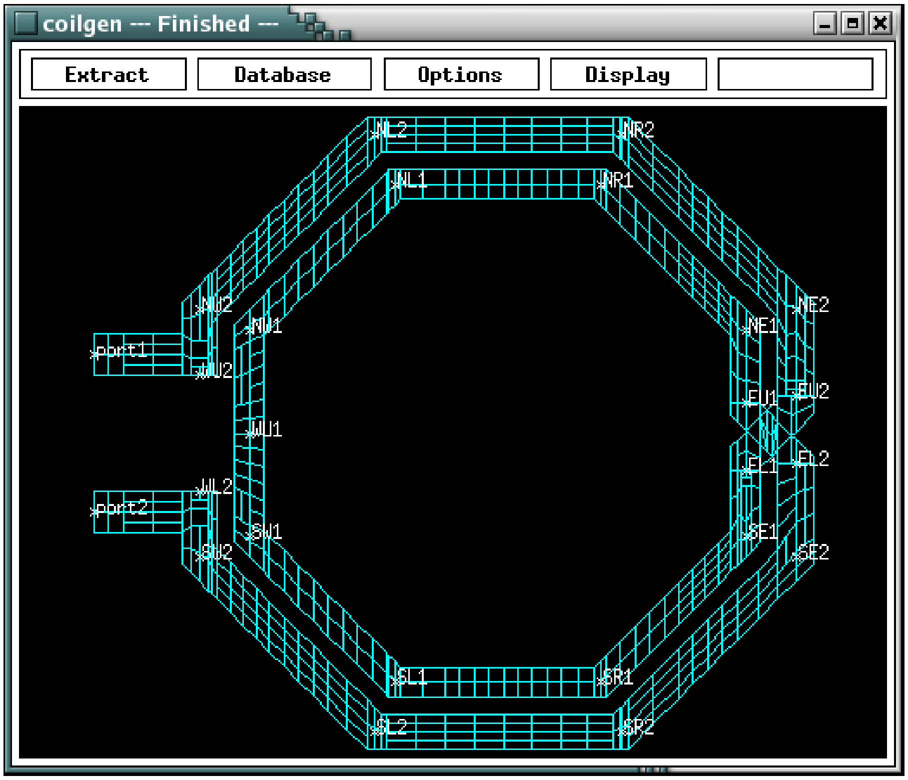
```
                if (c_n -> flag3 == 1) {
                    if (con -> val == 0) res = 1e99;
                    else if ((res = 1 / con -> val) < 0) res = -res;
                    min = min_R_path (n, c_n, QC_cnt, r_flag);
                    if (min > 0 && min * max_par_res + 1e-9 < res) {
                        elemDel (con, k);
                    }
                    r_flag = 1;
                }
            }
        }

        for (j = 0; j < QC_cnt; j++) QC[j] -> flag = 0;
    }

    for (l = 0; l < QC_cnt; l++) {
        QC[l] -> flag3 = 0;
        QC[l] -> flag2 = 1;
    }
}

/* Cliques with only area nodes are still uninvestigated now.
   Therefore, evaluate the resistances between the area node pairs.
*/
if (area_nodes < 2) return;

QC_cnt = 0;
for (i = 0; i < n_cnt; i++) {
    if (qn[i] -> area) {
        qn[i] -> flag3 = 1;
        QC[ QC_cnt++ ] = qn[i]; // putQueueClique
    }
}

for (l = 0; l < QC_cnt-1; l++) {
    n = QC[l];
    n -> flag3 = 2;

    for (j = 0; j < QC_cnt; j++) QC[j] -> help = -1;

    r_flag = 0;
    for (k = 0; k < resSortTabSize; k++) {
        for (con = n -> con[k]; con; con = next) {
            next = NEXT (con, n);
            if (con -> type == 'S') continue;
            c_n = OTHER (con, n);
            if (c_n -> flag3 == 1) {
                if (con -> val == 0) res = 1e99;
                else if ((res = 1 / con -> val) < 0) res = -res;
                min = min_R_path (n, c_n, QC_cnt, r_flag);
                if (min > 0 && min * max_par_res + 1e-9 < res) {
                    elemDel (con, k);
                }
                r_flag = 1;
            }
        }
    }

    for (j = 0; j < QC_cnt; j++) QC[j] -> flag = 0;
}
}
```

APPENDIX D -- Diffs between file reduc.c 4.45 and 4.46

```
7c7
<     int i, j, k, l, r_flag;
---
>     int i, j, k, l, r_flag, area_nodes;
9a10
>     double min, res;
11c12
<     if (!optRes || max_par_res < 0) return;
---
>     if (!optRes || max_par_res < 1 || n_cnt < 2) return;
20a22
>     area_nodes = 0;
23c25,26
<         if (n -> area || n -> flag2) continue;
---
>         if (n -> area) { ++area_nodes; continue; }
>         if (n -> flag2) continue;
44c47
<         for (l = 0; l < QC_cnt; l++) {
---
>         for (l = 0; l < QC_cnt-1; l++) {
46c49
<             ASSERT (n -> flag3);
---
>             n -> flag3 = 2;
56,62c59,64
<                     if (c_n -> flag3 && (unsigned)n < (unsigned)c_n) {
<                         if (con -> val != 0) {
<                             if (min_R_path (n, c_n, n_cnt, r_flag) * max_par_res
<                                  < 1 / Abs (con -> val)) {
<                                 elemDel (con, k);
<                             }
<                             r_flag = 1;
---
>                     if (c_n -> flag3 == 1) {
>                         if (con -> val == 0) res = 1e99;
>                         else if ((res = 1 / con -> val) < 0) res = -res;
>                         min = min_R_path (n, c_n, QC_cnt, r_flag);
>                         if (min > 0 && min * max_par_res + 1e-9 < res) {
>                             elemDel (con, k);
63a66
>                         r_flag = 1;
73c76
<             if (!QC[l] -> area) QC[l] -> flag2 = 1;
---
>             QC[l] -> flag2 = 1;
77a81
>     if (area_nodes < 2) return;
78a83
>     QC_cnt = 0;
80c85,88
<         if (qn[i] -> area) qn[i] -> flag3 = 1;
---
>         if (qn[i] -> area) {
>             qn[i] -> flag3 = 1;
>             QC[ QC_cnt++ ] = qn[i]; // putQueueClique
>         }
83,85c91,93
<     for (i = 0; i < n_cnt; i++) {
<         n = qn[i];
<         if (!n -> area) continue;
```

```
---
>     for (l = 0; l < QC_cnt-1; l++) {
>         n = QC[l];
>         n -> flag3 = 2;
87c95
<         for (j = 0; j < n_cnt; j++) qn[j] -> help = -1;
---
>         for (j = 0; j < QC_cnt; j++) QC[j] -> help = -1;
95,101c103,108
<                 if (c_n -> flag3 && (unsigned)n < (unsigned)c_n) {
<                     if (con -> val != 0) {
<                         if (min_R_path (n, c_n, n_cnt, r_flag) * max_par_res
<                              < 1 / Abs (con -> val)) {
<                             elemDel (con, k);
<                         }
<                         r_flag = 1;
---
>                 if (c_n -> flag3 == 1) {
>                     if (con -> val == 0) res = 1e99;
>                     else if ((res = 1 / con -> val) < 0) res = -res;
>                     min = min_R_path (n, c_n, QC_cnt, r_flag);
>                     if (min > 0 && min * max_par_res + 1e-9 < res) {
>                         elemDel (con, k);
102a110
>                     r_flag = 1;
107c115
<         for (j = 0; j < n_cnt; j++) qn[j] -> flag = 0;
---
>         for (j = 0; j < QC_cnt; j++) QC[j] -> flag = 0;
167,168c175
<     if (!n) return (1e+30); /* no path (along positive R's) */
<     return (n -> help);
---
>     return (nt -> help); /* if (help < 0) no path (along positive R's) */
```

APPENDIX E -- Test layout coilgen



APPENDIX F -- Test result coilgen

```
network coilgen (terminal NL1, port1, port2, NR1, SL1, SE1, SW1, NW1, WU1, NW2,
                 SR1, SW2, EL2, SE2, EU1, WL2, WU2, NE2, NL2, NR2, WL1, EU2,
                 SL2, SR2, NE1, EL1)
{
    net {WU1, WL1};

    /* substrate resistor part */
    res 350.2276k (1, 3);
    res 1.524709M (1, 9);
    res 446.8717k (1, 4);
    res 1.454077M (1, 10);
    res 535.2308k (1, 8);
    res 472.8638k (1, 5);
    res 228.648k (1, 6);
    res 36.24017k (1, 7);
    res 13.24587k (1, SUBSTR);
    res 348.1347k (2, 15);
    res 443.3744k (2, 18);
    res 1.524709M (2, 11);
    res 467.6385k (2, 17);
    res 533.951k (2, 16);
    res 1.454077M (2, 12);
    res 240.4154k (2, 14);
    res 36.21573k (2, 13);
    res 13.23007k (2, SUBSTR);
```

```
    res 4.248458M (3, 21);
    res 717.7612k (3, 23);
    res 374.0033k (3, 22);
    res 352.2195k (3, 19);
    res 15.99485k (3, 20);
    res 30.00612k (3, 4);
    res -24.29189M (3, 5);
    res 2.466167M (3, 6);
    res 2.153876M (3, 7);
    res 78.60294k (3, 8);
    res 294.2514k (3, 9);
    res -22.92306M (3, 10);
    res 295.8869k (3, SUBSTR);
    ...
    ...
    res 83.47287k (120, SUBSTR);
    res 155.8883k (121, 124);
    res 1.755354M (121, 122);
    res 15.44243k (121, 125);
    res 146.7759k (121, 123);
    res 36.94201k (121, 126);
    res 51.06214k (121, SUBSTR);
    res 412.3285k (122, 124);
    res 238.5616k (122, 123);
    res 1.063008M (122, 125);
    res 739.5383k (122, SUBSTR);
    res 936.5304k (123, 124);
    res 406.6315k (123, 125);
    res 794.9201k (123, SUBSTR);
    res 17.70575k (124, 125);
    res 276.6507k (124, 126);
    res 63.37967k (124, SUBSTR);
    res 201.2622k (125, 126);
    res 76.88452k (125, SUBSTR);
    res 25.52495k (126, SUBSTR);

    /* interconnect resistor part */
    res 181.1118m (port2, SW2);
    res 207.1595m (port2, WL2);
    res 58.99385m (WL2, SW2);
    res 259.3106m (SL2, SR2);
    res 206.8761m (SL2, SW2);
    res 256.9423m (SR2, SE2);
    res 1.975785 (EL1, EU2);
    res 65.20366m (EL1, SE1);
    res 299.3041m (NW1, NL1);
    res 8.86235m (NW1, WU1);
    res 8.86235m (WU1, SW1);
    res 299.3041m (SW1, SL1);
    res 263.668m (SL1, SR1);
    res 190.576m (SR1, SE1);
    res 181.1799m (port1, NW2);
    res 208.8902m (port1, WU2);
    res 59.42574m (WU2, NW2);
    res 206.8129m (NW2, NL2);
    res 259.3741m (NL2, NR2);
    res 196.5181m (NR2, NE2);
    res 76.90015m (NE2, EU2);
    res 263.7301m (NL1, NR1);
    res 178.3569m (NR1, NE1);
    res 93.04827m (EU1, EL2);
    res 56.56543m (EU1, NE1);
    res 99.7248m (SE2, EL2);
}
```