Using Space Resistance and Capacitance Extraction Filters

S. de Graaf

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

Report EWI-ENS 14-02
Feb. 27, 2014

Last revision: Feb. 28, 2014.

1. INTRODUCTION

When requested with option **-r**, the *space* layout to circuit extractor extracts resistances for all high resistive conductors in the layout. Note that parameter "low_sheet_res", default 1 ohm, controls which conductors are low/high resistive. Note that parameter "low_sheet_res" is forced to 0 ohm, when requesting an extraction with all conductors being high resistive (if not zero) with option **-%R**.

When we look into the technology file of the "scmos_n" example process, we can find the following resistivities for the conductors:

```
conductors :
 # name     : condition      : mask : resistivity : type
   cond_mf  : cmf            : cmf  : 0.045       : m  # first metal
   cond_ms  : cms            : cms  : 0.030       : m  # second metal
   cond_pg  : cpg            : cpg  : 40          : m  # poly interconnect
   cond_pa  : caa !cpg !csn  : caa  : 70          : p  # p+ active area
   cond_na  : caa !cpg  csn  : caa  : 50          : n  # n+ active area
   cond_well : cwn           : cwn  : 0           : n  # n well
```

Thus, when we do a flat resistance extraction of for example the "switchbox4" demo layout, we normally extract only resistances for the "cpg" and "caa" masks (if the given condition is matched). Note that "caa" represents the active (drain / source) areas of the "nenh" and "penh" fets. And that "cpg" represents the poly gate and interconnect areas.

The following command is normally used to run a resistance extraction:

```
% space -Fr switchbox4
```

The following command can be used to run an all resistance extraction:

```
% space -%FR switchbox4
```

However, it is equal to the following resistance extraction command:

```
% space -Fr -Slow_sheet_res=0 switchbox4
```

In the last case we extract also resistances for the "cmf" and "cms" masks (the metal interconnect). However, when we don't want to extract resistance for example for all "cpg", then we must change the condition for the "cpg" conductor. For example, we can use the "cx" filter mask to split out the condition:

```
   cond_pg : cpg !cx       : cpg  : 40      : m  # poly interconnect
   cond_pgx: cpg  cx       : cpg  :  0      : m  # poly gate
```

But now, with a filter, we can also get the above result with following clause:

```
filters : # masks1 : masks2 : fmask : type
  r_cpg :   cpg    :        :  cx   : r
```

However, the filter becomes only active, when following command is given:

```
% space -Fr -Senable_filter=r_cpg switchbox4
```

Thus, you don't need to split the condition of the "cond_pg" conductor element.

When there is also a "r_caa" filter, which must be enabled, you can give the command:

```
% space -Fr -Senable_filter=r_cpg,r_caa switchbox4
```

However, you can keep it sort, by using wildcards. For example:

```
% space -Fr -Senable_filter=r* switchbox4
```

Note however, when giving this command directly to a command shell, you must escape the wildcard character. You can use a backslash before the '*' character or you can place the part between single quotation marks:

```
% space -Fr -Senable_filter='r*' switchbox4
```

Other wildcard possibilities are brackets and question marks. A question mark and the brackets match one character on a string position. Thus, if you have for example 3 filters called "filter1", "filter2" and "filter3". You can select all filters with:

```
% space -Fr -Senable_filter='*' switchbox4
% space -Fr -Senable_filter='filter?' switchbox4
% space -Fr -Senable_filter='filter[1-3]' switchbox4
```

When you want to select only filters "filter1" and "filter3", you can specify:

```
% space -Fr -Senable_filter='filter[13]' switchbox4
```

More about the technology file filters section.

First you must know that the filters section must be placed after the conductors section. You can specify filters for conductors, to force the resistivity value to be zero, when the filter mask is enabled and is laying over the conductor in the layout. When in the conductor filter specification "masks2" is specified, then the contacts between "masks1" and "masks2" are also filtered. For example, mask "cpg" can only have a contact with mask "cmf". When you also want to filter these contacts, you need to specify:

```
filters : # masks1 : masks2 : fmask : type
  r_cpg :   cpg    : cmf    :  cx   : r
```

Note that this specification does not filter the "cmf" mask. Because the "masks2" part is for resistance filters only used for contacts. The "masks2" part can contain "@sub", but not "@gnd", because there don't exist contacts with ground "@gnd". For example

```
filters : # masks1 : masks2 : fmask : type
  r_cpg : cpg cmf  : cmf @sub :  cx : r
```

shall filter the masks "cpg" and "cmf" and also the contacts between "cpg"/"cmf" and between "cmf"/"@sub" and (if possible) between "cpg"/"@sub". Note that this specification can also be specified with two separate filters (using different filter names), but by using the same filter mask "cx".

2. CAPACITANCE FILTERS

First, when requested with option **-C**, the *space* layout to circuit extractor shall extract edge and surface couple capacitances for the layout masks. And, when using option **-l**, shall also extract lateral couple capacitances. Of course the capacitances need to be specified in the "capacitances" section of the used technology file.

The following command is normally used to run a capacitance extraction:

```
% space -FC switchbox4
```

When you want to use a capacitance filter, for example "c_cpg", you must enable it by specifying the parameter "enable_filter". For example:

```
% space -FC -Senable_filter=c_cpg switchbox4
```

With a capacitance filter you can filter out certain capacitances. The capacitance filter must be placed in the used technology file after the "conductors" section. For example:

```
filters : # masks1 : masks2         : fmask : type
  c_cpg : cpg      : cpg @gnd @sub :  cx   : c
```

Because a capacitance has always two pins, both the "masks1" and "masks2" parts need to be specified. Of course, the specified masks need to be conductor masks. Only the "masks2" part may contain also "@gnd" and/or "@sub". The above filter shall filter out edge and surface capacitances between "cpg"/"@gnd" and "cpg"/"@sub" and (if possible) also filter out lateral capacitances between "cpg"/"cpg".

More about 3D capacitance extraction.

The above filter works also for 3D capacitance extraction. Normally the "capacitances" section of the used technology file is in this case not more used. But a "vdimensions" section for the to use conductor masks needs to be specified and also a "dielectrics" section. To do the 3D capacitance extraction, you need to use the 3D version of the layout to circuit extractor. For example, with filter, use the following command:

```
% space3d -C3 -Senable_filter=c_cpg switchbox4
```

Note that the "flat" extraction option **-F** does not need to be specified, because a 3D extraction needs always to be flat (except when parameter "allow_hierarchical_cap3d" is specified). When also a resistance extraction must be done (with filters), run for example the following command:

```
% space3d -C3r -Senable_filter='c*,r*' switchbox4
```

Or, when all filters must be used, the following command:

```
% space3d -C3r -Senable_filter='*' switchbox4
```

Note that you can always use this command, also when there are resistance filters and resistance extraction is not used. Because, in that case the resistance filters are skipped.

3. IMPLEMENTATION

Function "readTechFile" shall read the used technology file and shall try to read the filter section when "optCap" or "optIntRes" is specified. On forehand, a list of filter data structures in allocated (with maximum possible count). Parameter "enable_filter" is read and only the to use filters are installed in the list. First, if needed, the capacitance filters are read. The number of found capacitance filters is counted with "nrCapFilters". Note that only the first 60 capacitance filters may contain "@gnd" and/or "@sub", because bitmasks "gndFilters(2)" and "subFilters(2)" are used for registration. Note that only the first 32 resistance filters may contain "@sub", because a bitmask "rsubFilter" is used for the registration. The number of resistance filters is counted with "nrResFilters". Pointer "cap_filter" points to the first capacitance filter in the filter-list. Pointer "res_filter" points to the first resistance filter in the filter-list. The pointer is NULL, when there is no cap/res filter. Each layout mask has a unique mask color bit. The color bits of all cap filter masks are saved in "cap_filter_masks" and for the res filter masks in "res_filter_masks". For capacitance extract, the color of a tile is tested against the "cap_filter_masks", but only when "cap_filter" is set. The used conductors are saved in bitmasks "con1", "con2" and "con". Note that "con" contains "con1 | con2". Thus, only the first 32 conductors can be used for the filters. For the capacitance filters, not more than 32 different filter masks may be used. Variable "nrCapFilterMasks" counts the used number. Two arrays are used, to save info about the capacitance filter masks. Array "fmConBM" contains for each filter mask a bitmask of the used "con". Array "fmColor" contains for each filter mask the used mask color.

In function "enumPair" in a new tile the "HasCapFilter" bit is set in the "known" field, when "cap_filter_masks" is present in the tile "color". In function "updateEdgeCap" and other functions, macro "HasCapFilter" is used to check the tile "known" field, if true function "filterThisCap" is called for the capacitor conductor pins to do a more accurate test if this capacitor must be skipped.

```
if (HasCapFilter(tile) && filterThisCap(tile,cap->pCon,cap->nCon)) continue;
```

Also, in function "resEnumPair" (if res is present) in a new tile the "HasResFilter" bit is set in the "known" field, when "res_filter_masks" is present in the tile "color". Later on, in function "connectPoints" the macro "HasResFilter" is used for high res conductors to filter them out by not setting the subnode highres flag to 1. Note that "psn" is a subnode in the "otherTile" and is not a subnode of a node point.

```
j = 1;
if (optPrick) { /* test for selective res */ }
if (j && HasResFilter(otherTile) && filterThisRes(otherTile,i,-1)) j = 0;
if (j) { psn -> highres = 1; SET_KNOW1(otherTile); }
else   { psn -> highres = 2; }
```

Note that function "filterThisRes" is also used in function "resEnumTile" to possible filter out contacts (by changing the value into 0) between two conductors.

Here follows a part of the code of function "filterThisCap":

```
for (i = 0; i < nrCapFilters; ++i) {
   if (COLOR_ABSENT (tile -> color, cap_filter[i].mask)) continue;
   if (cap_filter[i].con1 & (1 << c1)) {
      if (c2 < 0) {
         if (i < 28) {
            if (c2 == -1) { if (gndFilters & (1 << i)) return 1; }
            else if (subFilters & (1 << i)) return 1;
         } else if (i < 60) {
            if (c2 == -1) { if (gndFilter2 & (1 << (i-28))) return 1; }
            else if (subFilter2 & (1 << (i-28))) return 1;
         } else return 0;
      }
      else if (cap_filter[i].con2 & (1 << c2)) return 1;
   }
   if (c2 >= 0 && (cap_filter[i].con1 & (1 << c2)))
      if (cap_filter[i].con2 & (1 << c1)) return 1;
}
return 0;
```

In function "resEnumTile" is only resistance extract done for conductors when the tile has more than one node point (pTRb != pTR). This indicate, that there is at least one conductor with high resistivity. The resistive conductors are done in function "triangular", we make a sort array of these conductors by using "lastA" and "conNums". Here a small part of the code used:

```
pTR  = tile -> rbPoints;
pTRb = tile -> tlPoints;
if ((split = (pTRb != pTR))) { /* high res tile */
   lastA = 0;
   for (con = 0; con < nrOfConductors; ++con)
   if ((sn = tile -> cons[con])) {
      sn1 = pTR  -> cons[con];
      sn2 = pTRb -> cons[con];
      subnodeJoin (sn1, sn2);
      if (sn -> highres == 1) { /* HIGH_RES */
         conNums[lastA++] = con;
         conVal[con]  = sn -> cond -> val;
         conSort[con] = sn -> cond -> sortNr;
      } else { /* LOW_RES */
         for(p = pTRb->next; p; p = p->next) subnodeJoin(sn1,p->cons[con]);
         for(p = pTR ->next; p; p = p->next) subnodeJoin(sn1,p->cons[con]);
         makeAreaNode (sn1);
      }
   }
   ASSERT(lastA > 0);
   triangular (tile);
}
```

From above code, you see, that the subnodes of the last two points must be joined together. And you see, that the tile can contain low resistive conductors, which node points must be joined together.

4. IMPLEMENTATION FOR 3D CAP

Function "spiderPair" is called by function "enumPair" or "resEnumPair". There is a left tile "tile_l" and a right tile "tile_r", both tiles are checked for "HasCapFilter". If true, there is made a list of filter masks, see code part for "tile_l":

```
if (HasCapFilter (tile_l)) /* which filtermasks are in tile_l */
   for (i = 0; i < nrCapFilterMasks; ++i)
      if (!COLOR_ABSENT (tile_l->color, fmColor[i])) nF_l[nFilters_l++] = i;
```

Now, for each mesh conductor (< 32) we find in "tile_l" or "tile_r", we make a filter bitmap. Note "filter1" for "tile_l" and "filter2" for "tile_r", see code part for "tile_l":

```
filter1 = filter2 = isGate = isGat2 = 0;
con = 1 << conductor;
if (nFilters_l && m -> solid_l)
   for (j = 0; j < nFilters_l; ++j)
      if (fmConBM[nF_l[j]] & con) filter1 |= (1 << nF_l[j]);
```

After that, we test if a filter is found in "tile_l" or "tile_r" and set the "isGate" and "isGat2" bitmask. A special bit (28) in "isGate" is used to flag cap-filter existance. Note that bit 30 is used to flag a diffusion conductor and bit 31 to flag a gate conductor. See following code fragment:

```
if (filter1 || filter2) { /* cap filter found in (tile_l || tile_r) */
   filter12 = filter1 | filter2;
   for (j = 0; j < nrCapFilters; ++j)
     if (filter12 & (1 << cap_filter[j].nr)) {
        if (cap_filter[j].con & con) { /* conductor found in filter j */
           if (j < 28) isGate |= (1 << j); else isGat2 |= (1 << (j - 28));
        }
     }
   if (isGate || isGat2) isGate |= 1 << 28;
}
```

For each new created conductor spider "isGate" and "isGat2" is set, see the functions "spiderFindNew" and "spiderNew":

```
spiderFindNew (tile_t *tile, ..., int level, int conductor, ...)
{
    if (!sp1) sp1 = spiderNew (x1, y1, z, conductor, tile);
    else if (isGate) sp1 -> isGate |= isGate;
    sp1 -> isGat2 |= isGat2;
    ...
}
```

Note that the global variable "filter" is used in function "tryFace" and "spiderFindFace" to set the "filter" field in a new face. But first, variable "filter" is set to "filter1" or "filter2". In function "meshRefine", in the called "reconstructFace" functions, the face "filter" and "type" fields are used for the face split and merge operations. Note that only faces with the same "type" and "filter" fields may be merged.

At last "addCap", this function does use the functions "extractGnd" and "extractCoup" to decide of the calculated capacitance value "val" may be added by function "capAdd". See the following code fragments from the source file "cap3d.c":

```
if (!extractGateGndCap3d)   { gndFilters |= 1 << 31; subFilters |= 1 << 31; }
if (!extractDiffusionCap3d) { gndFilters |= 1 << 30; subFilters |= 1 << 30; }

#define hasFilter(sp) (sp->isGate & (1 << 28))
#define hasDiff(sp) (sp -> isGate & (1 << 30))
#define hasGate(sp) (sp -> isGate & (1 << 31))

int extractGnd (spider_t *s1) {
  if (s1 -> isGate) {
     if (s1 -> isGate & (s1 -> subnode2 ? subFilters : gndFilters)) return 0;
     if (s1 -> isGat2 & (s1 -> subnode2 ? subFilter2 : gndFilter2)) return 0;
  }
  return 1;
}

int extractCoup (spider_t *s1, spider_t *s2) {
  if (s1 -> isGate) {
    if (hasFilter (s1) && hasFilter (s2)) {
      con1 = (1 << s1 -> conductor);
      con2 = (1 << s2 -> conductor);
      f = s1 -> isGate & s2 -> isGate & 0x0fffffff;
      for (i = 0; f; ++i, f >>= 1) if (f & 1) {
        if((cap_filter[i].con1 & con1) && (cap_filter[i].con2 & con2)) return 0;
        if((cap_filter[i].con1 & con2) && (cap_filter[i].con2 & con1)) return 0;
      }
      f = s1 -> isGat2 & s2 -> isGat2;
      for (i = 28; f; ++i, f >>= 1) if (f & 1) {
        if((cap_filter[i].con1 & con1) && (cap_filter[i].con2 & con2)) return 0;
        if((cap_filter[i].con1 & con2) && (cap_filter[i].con2 & con1)) return 0;
      }
    }
    if (hasDiff (s1)) {
      if (hasDiff (s2)) return extractDiffusionCap3d;
      else if (hasGate (s2)) return extractGateDsCap3d;
    }
    else if (hasGate (s1) && hasDiff (s2)) return extractGateDsCap3d;
  }
  return 1;
}

void addCap (spider_t *s1, spider_t *s2, schur_t val) {
  if (s1 != s2) {
    if (s1->subnode->node != s2->subnode->node)
       if (extractCoup (s1, s2)) capAdd (s1->subnode, s2->subnode, -val, 0);
    if (extractGnd (s2)) capAdd (s2->subnode, s2->subnode2, val, 0);
  }
  if (extractGnd (s1)) capAdd (s1->subnode, s1->subnode2, val, 0);
}
```

5. EXAMPLE TECHNOLOGY FILE

```
# space element definition file for scmos_n example process
# with transistor bulk connections and substrate terminals
# for substrate contacts and nmos bulk connections, and
# with information for 3D capacitance extraction.

unit resistance    1     # ohm
unit c_resistance  1e-12 # ohm um^2
unit a_capacitance 1e-6  # aF/um^2
unit e_capacitance 1e-12 # aF/um
unit capacitance   1e-15 # fF
unit vdimension    1e-6  # um

conductors :
 # name    : condition     : mask : resistivity : type
   cond_mf : cmf           : cmf  : 0.045       : m  # first metal
   cond_ms : cms           : cms  : 0.030       : m  # second metal
   cond_pg : cpg           : cpg  : 40          : m  # poly interconnect
   cond_pa : caa !cpg !csn : caa  : 70          : p  # p+ active area
   cond_na : caa !cpg  csn : caa  : 50          : n  # n+ active area
   cond_well : cwn         : cwn  : 0           : n  # n well

filters :
 # name     : masks1      : masks2      : fmask : type
  cfilter_1 : caa cpg cwn : caa cpg cwn : cx    : c
  cfilter_1g: caa         :        @gnd : cx    : c
  rfilter_1 : caa cpg cwn :             : cx    : r
  rfilter_2 : cms         :        @sub : cx    : r

fets :
 # name : condition    : gate d/s  : bulk
   nenh : cpg caa  csn : cpg  caa  : @sub  # nenh MOS
   penh : cpg caa !csn : cpg  caa  : cwn   # penh MOS

contacts :
 # name   : condition         : lay1 lay2 : resistivity
   cont_s : cva cmf cms       : cmf  cms  :   1   # metal to metal2
   cont_p : ccp cmf cpg       : cmf  cpg  : 100   # metal to poly
   cont_a : cca cmf caa !cpg cwn !csn
          | cca cmf caa !cpg !cwn csn
                              : cmf  caa  : 100   # metal to active area
   cont_w : cca cmf cwn csn   : cmf  cwn  :  80   # metal to well
   cont_b : cca cmf !cwn !csn : cmf  @sub :  80   # metal to subs

junction capacitances ndif :
 # name    :  condition                 : mask1 mask2 : capacitivity
   acap_na :  caa       !cpg  csn !cwn  : @gnd  caa : 100  # n+ bottom
   ecap_na : !caa -caa !-cpg -csn !-cwn : @gnd -caa : 300  # n+ sidewall

junction capacitances nwell :
   acap_cw :  cwn                      : @gnd  cwn : 100  # bottom
   ecap_cw : !cwn -cwn                 : @gnd -cwn : 800  # sidewall
```

```
junction capacitances pdif :
   acap_pa :  caa       !cpg  !csn cwn      :  caa cwn : 500 # p+ bottom
   ecap_pa : !caa -caa !-cpg !-csn cwn -cwn : -caa cwn : 600 # p+ sidewall

capacitances :
 # polysilicon capacitances
   acap_cpg_sub :  cpg               !caa !cwn :  cpg @gnd : 49
   acap_cpg_cwn :  cpg               !caa  cwn :  cpg cwn  : 49
   ecap_cpg_sub : !cpg -cpg !cmf !cms !caa !cwn : -cpg @gnd : 52
   ecap_cpg_cwn : !cpg -cpg !cmf !cms !caa  cwn : -cpg cwn  : 52

 # first metal capacitances
   acap_cmf_sub :  cmf           !cpg !caa !cwn :  cmf @gnd : 25
   acap_cmf_cwn :  cmf           !cpg !caa  cwn :  cmf cwn  : 25
   ecap_cmf_sub : !cmf -cmf !cms !cpg !caa !cwn : -cmf @gnd : 52
   ecap_cmf_cwn : !cmf -cmf !cms !cpg !caa  cwn : -cmf cwn  : 52

   acap_cmf_caa :  cmf      caa !cpg !cca :  cmf  caa : 49
   ecap_cmf_caa : !cmf -cmf caa !cms !cpg : -cmf  caa : 59

   acap_cmf_cpg :  cmf      cpg !ccp :  cmf  cpg : 49
   ecap_cmf_cpg : !cmf -cmf cpg !cms : -cmf  cpg : 59

 # second metal capacitances
   acap_cms_sub :  cms      !cmf !cpg !caa !cwn :  cms @gnd : 16
   acap_cms_cwn :  cms      !cmf !cpg !caa  cwn :  cms cwn  : 16
   ecap_cms_sub : !cms -cms !cmf !cpg !caa !cwn : -cms @gnd : 51
   ecap_cms_cwn : !cms -cms !cmf !cpg !caa  cwn : -cms cwn  : 51

   acap_cms_caa :  cms      caa !cmf !cpg :  cms caa : 25
   ecap_cms_caa : !cms -cms caa !cmf !cpg : -cms caa : 54

   acap_cms_cpg :  cms      cpg !cmf :  cms cpg : 25
   ecap_cms_cpg : !cms -cms cpg !cmf : -cms cpg : 54

   acap_cms_cmf :  cms      cmf !cva :  cms cmf : 49
   ecap_cms_cmf : !cms -cms cmf      : -cms cmf : 61

   lcap_cms     : !cms -cms =cms    : -cms =cms : 0.07

vdimensions :
   v_caa_on_all : caa !cpg           : caa : 0.30 0.00
   v_cpg_of_caa : cpg !caa           : cpg : 0.60 0.50
   v_cpg_on_caa : cpg caa            : cpg : 0.35 0.70
   v_cmf        : cmf                : cmf : 1.70 0.70
   v_cms        : cms                : cms : 2.80 0.70

dielectrics :
 # Dielectric consists of 5 micron thick SiO2
 # (epsilon = 3.9) on a conducting plane.
   SiO2   3.9   0.0
   air    1.0   5.0
```