Lateral Capacitance Extraction

S. de Graaf

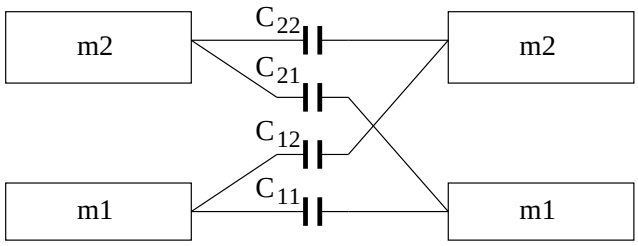
Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 12-02
May 7, 2012

Last revision: July 3, 2012.

1. INTRODUCTION

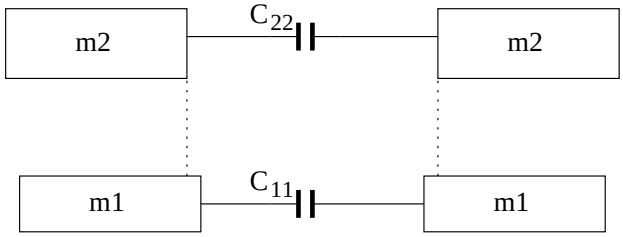
Besides the extraction of area and edge coupling capacitances, the *space* extractor can optional also extract lateral coupling capacitances. Of course you must specify these lateral capacitances in the *space* technology file and to extract you must specify the **-l** option. Lateral capacitances are edge-to-edge capacitances between opposite edges of two conductors. For example, the following figure shows four lateral capacitances:

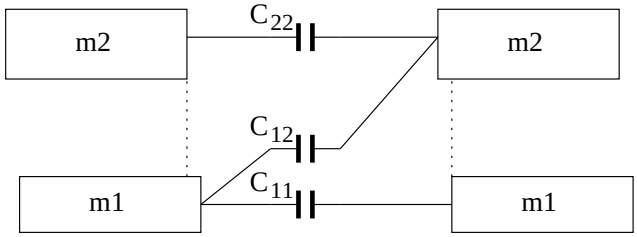


These lateral capacitances can be specified in the technology file with the following lines:

```
lcap_m1   : -m1 !m1 =m1     : -m1 =m1 : 1.0e-16
lcap_m2   : -m2 !m2 =m2     : -m2 =m2 : 1.2e-16
lcap_m1m2 : -m1 !m1 !m2 =m2 : -m1 =m2 : 0.8e-16
```

A ’-’ sign before the mask name means "edge" and a ’=’ sign means "other-edge".
The "lcap_m1m2" line specifies both capacitances C12 and C21.
Next figures show what happens when the edges of both masks are not exact inline:



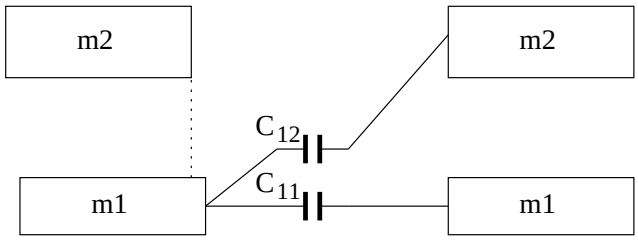


Some lateral capacitances are not found and not extracted.

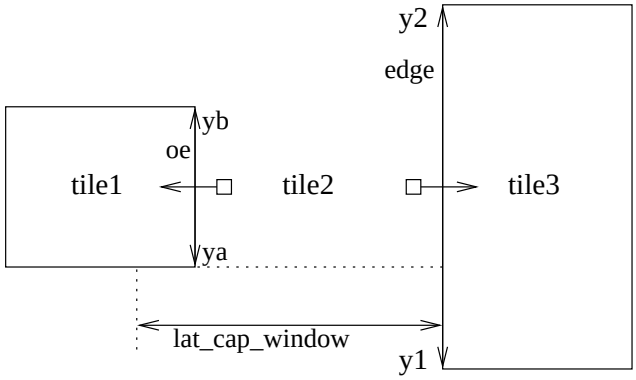
The following technology file specification gives another problem:

```
lcap_m2_a : -m2 !m1 !m2 =m2 : -m2 =m2 : 1.2e-16
lcap_m2_b : -m2  m1 !m2 =m2 : -m2 =m2 : 1.0e-16
```

The problem is that there is no lateral capacitance C22 extracted when mask m1 is partial found below mask m2 (see the figure below).



Lateral capacitances can be found when the edges of a tile are evaluated by function *enumPair* (see the figure below).

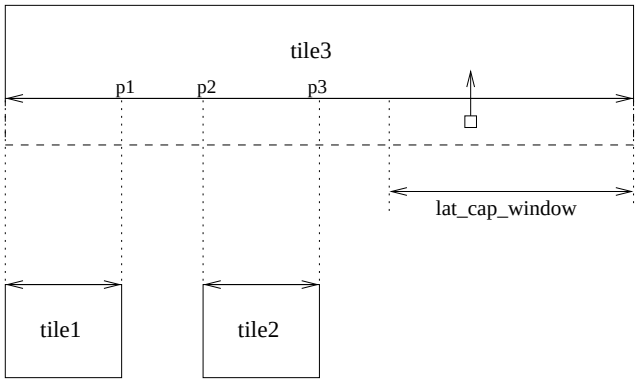


On a stateruler x-position the vertical edge between tile2 and tile3 is processed. For edge element recognition, tile2 is the surface mask side and tile3 is the edge mask side. When tile3 contains mask m1 and tile2 not, then there is possible a lateral capacitance C11. An opposite edge (oe) must be found, this edge must be in the **lat_cap_window** (this is a parameter which must be specified). The extractor is searching back and finds tile1 and a matching other-edge lateral element condition (when tile1 contains mask m1).

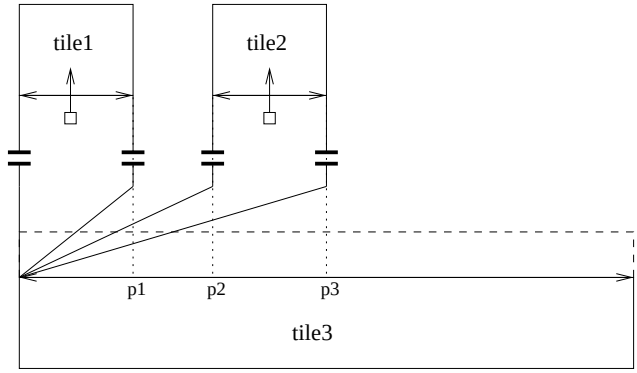
When only one lateral capacitance value is specified in the technology file, then this value is for a meter edge length and a meter distance. For a half edge length the value is divided by two and for a half distance the value is multiplied by two (thus there is a linear dependance). (You can use an interpolation method by specifying distance,cap pairs.)
By resistance extraction, when tile3 and/or tile1 contain node points, the calculated lateral edge capacitance is divided by two. One half is assigned to the top corners and one half is assigned to the bottom corners of the tiles.

Because of the **lat_cap_window** the tiles may not be cleared before they become outside this window. In that case function *clearTile* is called, which deletes the existing subnodes of the tile and deletes the tile.

For horizontal edges there may be the following problem situation. When the horizontal edge is processed by function *enumPair*, then the opposite edge can not more be found. This because the tile of the opposite edge is already outside the **lat_cap_window** distance. The following figure shows this problem:



You may say that the edge is normally split on the positions p1, p2 and p3. Yes this is normally true, but what if there is another edge between the tiles which is split in stead of the edge of tile3. The opposite situation, see the figure below, does not give this problem. But there can be a node assignment problem, because the tile is not yet processed.



In the above figure you see that all lateral capacitances are assigned to the upper-left corner of tile3, because the complete edge of tile3 is not yet processed. This is only the case when resistance is extracted and the conductor mask is not a low_sheet_res conductor.
To repair this problem we need some extra mesh refinement and for the first problem also when no resistance is extracted.

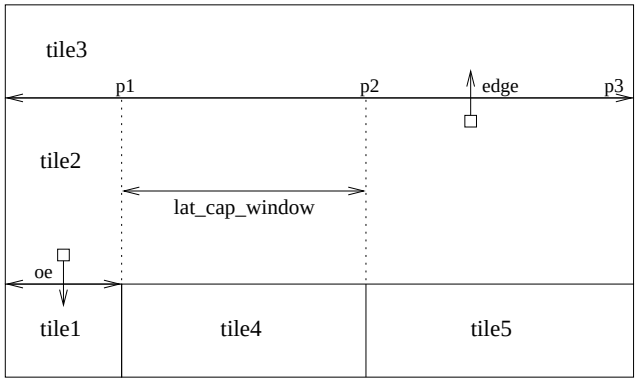
The following code is added to function *enumPair* to repair the first problem:

```
if (extrEdgeCaps) { /* update edge capacitances */
    if (tHasConduc) {
        elem = RecogE (newerTile -> color, tile -> color);
        i = updateEdgeCap (elem, newerTile, tile);
        if (i >= 0 && edgeOrien == 'h') newerTile -> content |= 8; // NEW1
    }
    if (nHasConduc) {
        elem2 = RecogO (tile -> color, newerTile -> color, ...);
        i = updateEdgeCap (elem2, tile, newerTile);
        if (i >= 0) { /* optLatCap */
            updateLateralCap (newerTile, tile, elem2, edgeOrien, i);
            if (edgeOrien == 'h'
                && tile -> content >= 8) tile -> content -= 8; // NEW2
        }
    }
}
```

And the following code is added to function *tileCrossEdge* (in scan/update.c):

```
if (!split) split = !COLOR_EQ_COLOR (edge -> tile -> color, color);

if (freeTile) { /* two tiles below edge */
    tile_t *fT = freeTile;                                  // NEW3
    TR (thisY, freeTile, ebwd -> tile, edge -> tile);
    ebwd -> tile -> tl = thisY; /* TL */
    if (edge -> tile -> content & 8) {                      // NEW4
        double w = Y (edge -> fwd, thisX) - thisY;
        if (edge -> yl != edge -> yr) w *= sqrt (2.);
        if (w <= effectDist) split = 1;
        else edge -> tile -> content -= 8;
    }
    else if (fT -> content & 8) {                           // NEW5
        double w = Y (edge -> fwd, thisX) - thisY;
        for (;;) {
           w += fT -> tr - fT -> br;
           fT = fT -> stb;
           while (fT -> xr < thisX) fT = fT -> str;
           if (fT -> content < 8) {
              if (fT -> tl != fT -> tr) w *= sqrt (2.);
              if (w <= effectDist) { edge->tile->content |= 8; split = 1; }
              break;
           }
        }
    }
}

if (split) { /* two tiles above edge */
    BR (thisY, tile, ebwd -> tile);
    freeTile = edge -> tile;
    edge -> tile = createTile (thisX, thisY, ...);
}
```

Note that more precisely the tile of an opposite edge (oe) is cleared after the scanline is finished and function *tileAdvanceScan* is called. And that *tileAdvanceScan* calls *clearTile* for every tile with a xr <= thisX - bandWidth (bandWidth is **lat_cap_window**). The following figure shows a problem example:



Because there is no reason to make a tile split in tile2 on positions p1 and p2, therefor the bottom "edge" of tile3 is processed on scanline position p3. Because there is a previous scanline position p2, which causes a *clearTile* of tile1, therefor the lateral capacitance between tile1 and tile3 cannot be extracted. The following new code is now used in function *enumPair* to get a tile split on position p1 in tile3 (i am now using a global variable lastY in place of the tile -> content).

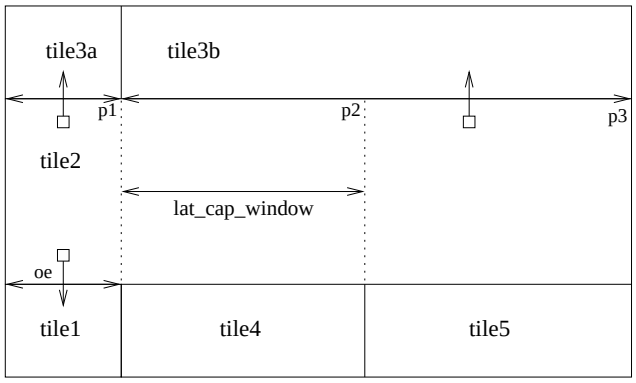
```
if (extrEdgeCaps) { /* update edge capacitances */
   if (tHasConduc) {
      elem = RecogE (newerTile -> color, tile -> color);
      i = updateEdgeCap (elem, newerTile, tile);
      if (i >= 0 && edgeOrien == 'h') lastY = bdr -> y2;  // NEW1
   }
   if (nHasConduc) { ... }
}
```

And the following code is added to function *tileCrossEdge* (in scan/update.c):

```
if (freeTile) { /* two tiles below edge */
   TR (thisY, freeTile, ebwd -> tile, edge -> tile);  ...
}
else if (!split && lastY != INF) { /* latcap "oe" */      // NEW2
   w = thisY - lastY;
   if (edge -> yr != edge -> yl) w /= sqrt (2.);
   if (w > effectDist) lastY = INF;
   else if ((edge -> xc - thisX) > effectDist)
        if (hasLatCap (ebwd->tile->color, edge->tile->color)) split = 1;
}
```

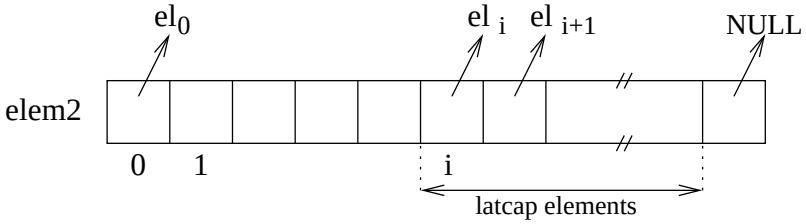
The "split" is only set when it is really needed. By a slanting edge the 'w' must be divided by the square root of 2 (only 45 degree slanting edges can be used). Note that function *tileAdvanceScan* also sets lastY back to INF.

Because of the new code tile3 is split into tile3a and tile3b (see figure below).

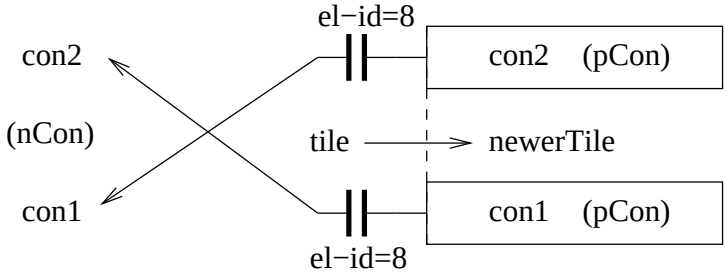


How does updateLatEdgeCap select lateral capacitances?

Function *enumPair* shall call function *updateLateralCap* when *updateEdgeCap* has found lateral capacitances and returns i >= 0. Variable i is the index of the first latcap element in the elem2 array (see figure below).



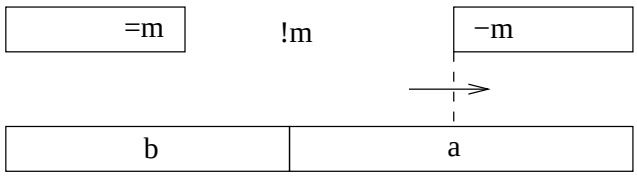
Note that only the lateral cap elements are placed in the array, which match the surface (tile) and edge (newerTile) mask color condition. Note that also the elements with a same element id are placed. This can happen, when there is an or condition and there are more possibilities. This can also happen, when there is an inverse lateral cap added, because it is an lateral cap between to different conductors. Note that both can be matched and found in the array. But these must separately be handled (see figure below).



Each lateral cap element has a done flag, which tells of the element is not yet finished on the level where we are. When all elements are finished, we don’t need to invest any futher. The procedure has the following shortcoming, that a lateral cap with different surface conditions possible is not found. For example the following definition:

```
lcap_m_a_or_b : -m !m (a | b) =m : -m =m : 1.2e-16
```

When we start with "-m !m a b", we have no problem, because we start with both caps. However, when we start with "-m !m a !b" or "-m !m !a b", we start only with one cap and cannot match the other cap element with same id anymore (see figure below).



This can also happen with the following definition:

```
new : a b : m
lcap_m : -m !m =m : -m =m : 1.2e-16
```

We get:

```
lcap_m : -(a b) !(a b) =(a b) : -m =m : 1.2e-16
lcap_m : (-a -b) (!a | !b) (=a =b) : -m =m : 1.2e-16
```

Is equal with:

```
lcap_m : (-a -b) (!a | !b) (=a =b) : -m =m : 1.2e-16
```

And we get:

```
lcap_m (1) : (-a -b) !a (=a =b) : -m =m : 1.2e-16
lcap_m (2) : (-a -b) !b (=a =b) : -m =m : 1.2e-16
```

Thus, when starting with "-m a !b" or "-m !a b", we cannot match the other anymore.

The following definition is also nice:

```
new : a | b : m
lcap_m : -m !m =m : -m =m : 1.2e-16
```

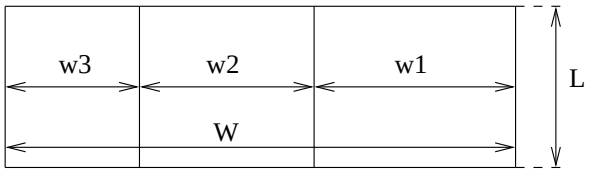
This case gives no problem, because we get:

```
lcap_m (1) : -a (!a !b) =a : -m =m : 1.2e-16
lcap_m (2) : -a (!a !b) =b : -m =m : 1.2e-16
lcap_m (3) : -b (!a !b) =a : -m =m : 1.2e-16
lcap_m (4) : -b (!a !b) =b : -m =m : 1.2e-16
```

Four lateral caps with the same id, but with equal surface condition.

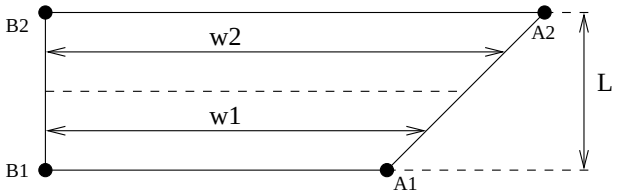
2. NEW LATCAP IMPLEMENTATION

1) A new data-structure is used to remember that possible different latcap element values are used for the same element. In that case the mean value can be calculated after an other-edge match is found. Example:



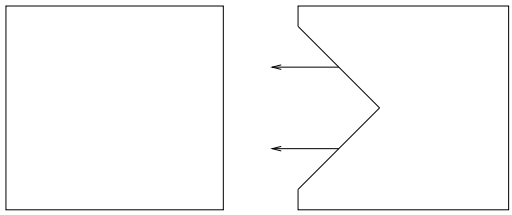
Clat = (w1/W*Clat1(W) + w2/W*Clat2(W) + w3/W*Clat3(W))*L

2) Another calculation method is used for possible slanting edges. Now all slanting edge situations are calculated (not only the parallel ones). A mean value is calculated for two distances. Example:



ClatA2B2 = Clat(w2)*L/2
ClatA1B1 = Clat(w1)*L/2

3) Now, for slanting horizontal edges there is also looked back in the vertical direction. Thus, for all cell use cases (mirror/rotate) the same latcaps can be found. See figure:

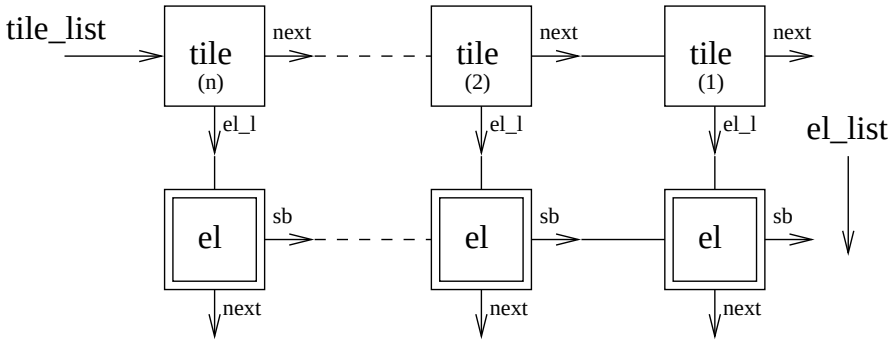


4) Depending on the start situation (latcap edge match) a set latcap elements is found and some latcap elements are not found. This is not more a problem, because we remember always only one element for reference. But for other-edge matching we try now always all elements of the reference.

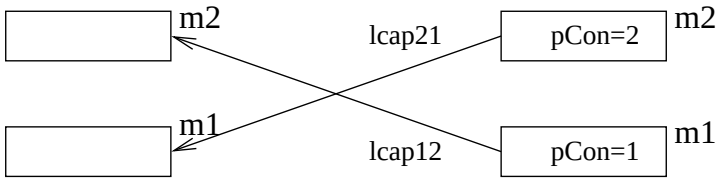
5) Besides this (see 4), for the surface side (gap) matching, we try also to find new latcap elements using the hasLatCap function for the new (otherTile, startTile) combination. If found, this latcap can have a different cap. value and is placed in the data-struct (see 1).

3. DATA STRUCTURE

A very simple data structure is used to remember the tiles and latcap elements:

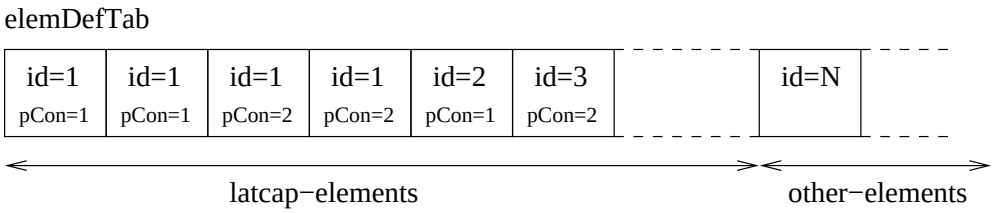


Each tile has at least one latcap element reference, but it can have a latcap element list. Only one latcap element with same id and same pCon is saved. A latcap with same id and different pCon can happen and is another element, see figure:



lcap12_21: −m1 =m2 !m1 !m2 : −m1 =m2 : val

We search the elemDefTab to find the possible other latcap elements with same id and pCon. We know that these elements are placed in the following order by tecc:

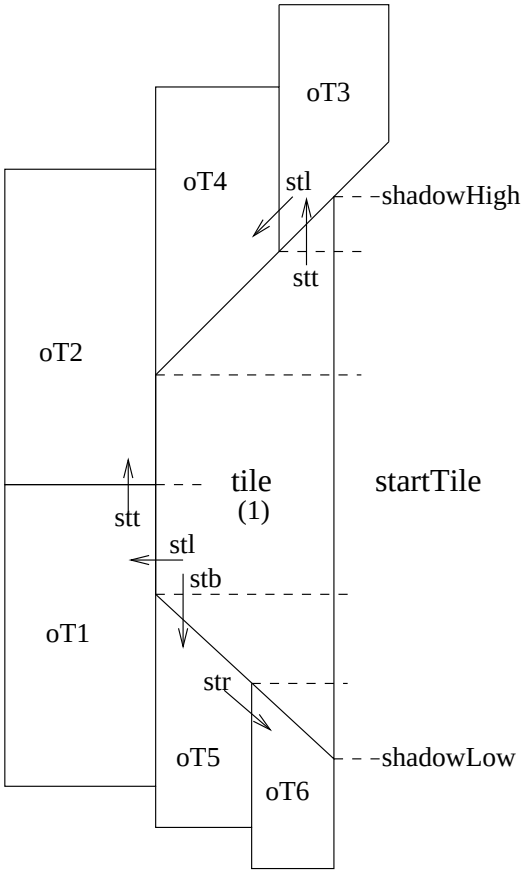


We start building an initial el_list and first tile-ref structure in the *updateLatCap* function. Function *updateLatCap* shall call function *updateLatEdgeCap* with level=0. For level=0 we know that there cannot be an other-edge match in the tile(1) and we can directly go to the part in *updateLatEdgeCap* to find an otherTile. We do only the otherTile if it is on a distance > 0 and <= lat_cap_window (or < lat_cap_window, if it is a slanting situation).
For each otherTile we call *updateLatEdgeCap* again with level=1. Each otherTile is

tested for a lateral other-edge match. When a match is not found (also not for elements with same id and pCon), we look of otherTile matches the gap (surface) condition. When non of the elements match the gap condition, we try at last to find another latcap element with the *hasLatCap* function, which matches the (otherTile, startTile) condition. Only if we can find a match we can go futher and we install the otherTile in the tile_list and the found element in the el_list of otherTile.

Note that at the end (before return) function *updateLatEdgeCap* shall dispose the finished tile-ref structure and its el_list. Thus, when returning to the initial level in the *updateLatCap* function, we know that the data structure is already disposed.

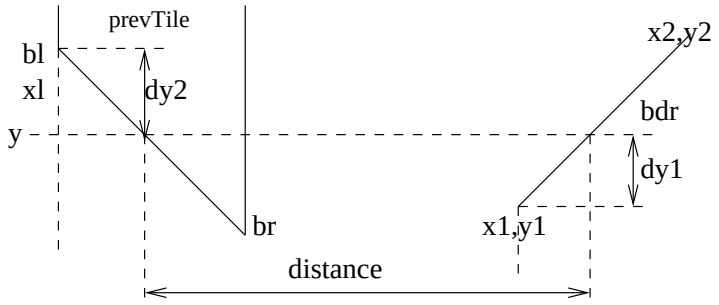
Ones again we shall show which otherTiles are possible when we start with the first tile on level=0 in v_mode, see figure below:



You see that in this case 6 otherTiles can be found. Each of these otherTiles can have a matching other-edge condition. Note that (tile, startTile) has a matching lateral edge condition.

4. NOTES

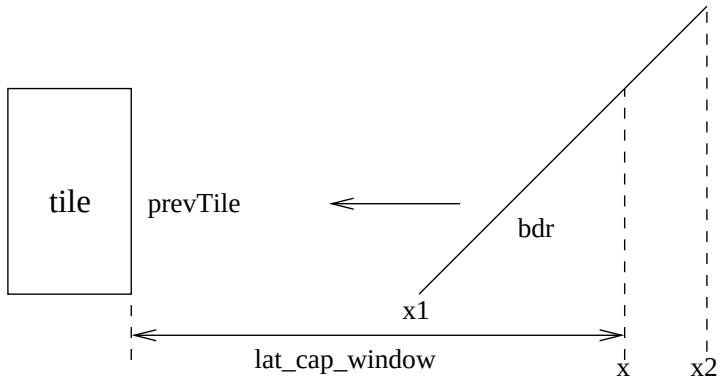
When an other-edge match is found, in most cases we don't need to use the new data structure and tile_list. Because there don't need to be different capacitance values. In that case the distance for some y-value can be calculated with (only 45 degree slanting):



distance = bdr−>x1 − prevTile−>xl + dy1 − dy2

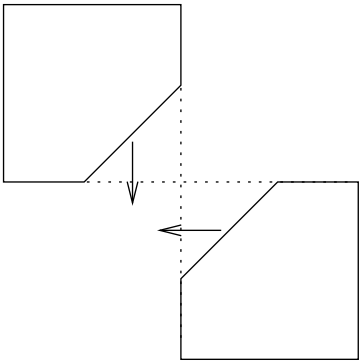
However, in most cases there are no slanting edges, in that case for the above v_mode situation the distance is: bdr->x1 - prevTile->xl.

In the above situation when we look back for v_mode, because we start with a slanting edge, there can be a problem to find the other-edge. The **tile** of the other-edge can already be cleared because it is outside the *lat_cap_window* at the position x2, when the starting slanting edge is processed (see figure below).



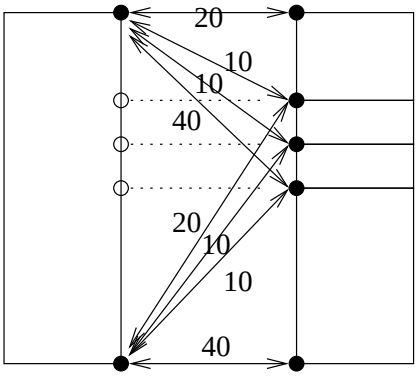
This happens when there is a state ruler position at x or between x and x2. To overcome this problem, we must split each slanting latcap edge at each state ruler position found between x1 and x2. Because we don't know if there is a **tile** and we don't know at which x-position the right side (xr) is laying. Maybe we can remember these tiles, which have a matching latcap edge.

Note that not in all slanting cases a latcap can be found, see figure:

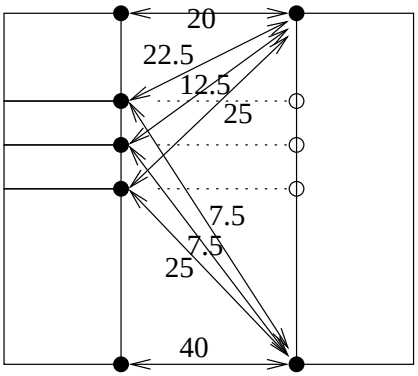


Note that a mirror of a cell does not always give the same latcap result, see figures:

test7h

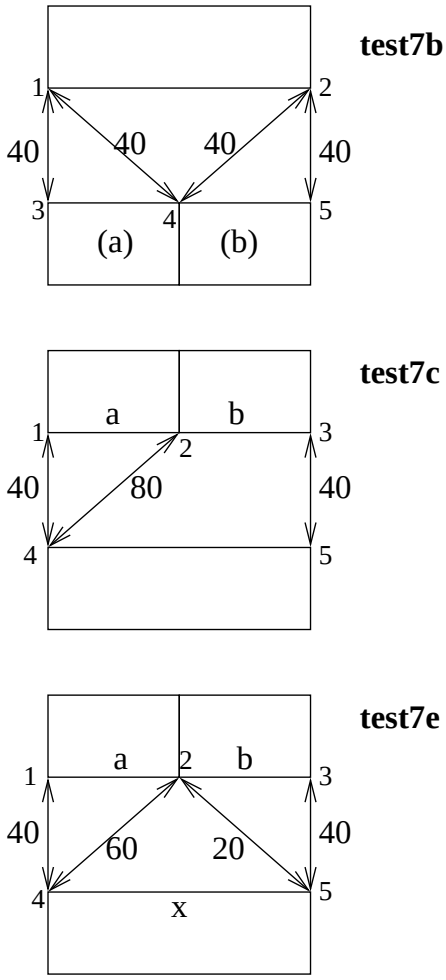


test7i



In cell **test7h** the extra nodes are first removed before latcap assignment.
In cell **test7i** the extra nodes are removed after latcap assignment.

Some other test cell examples (cells: **test7** , **test7a** t/m **test7g**) Almost all cells give the same result (like cell **test7b**). However, cells **test7c** and **test7e** give another result.

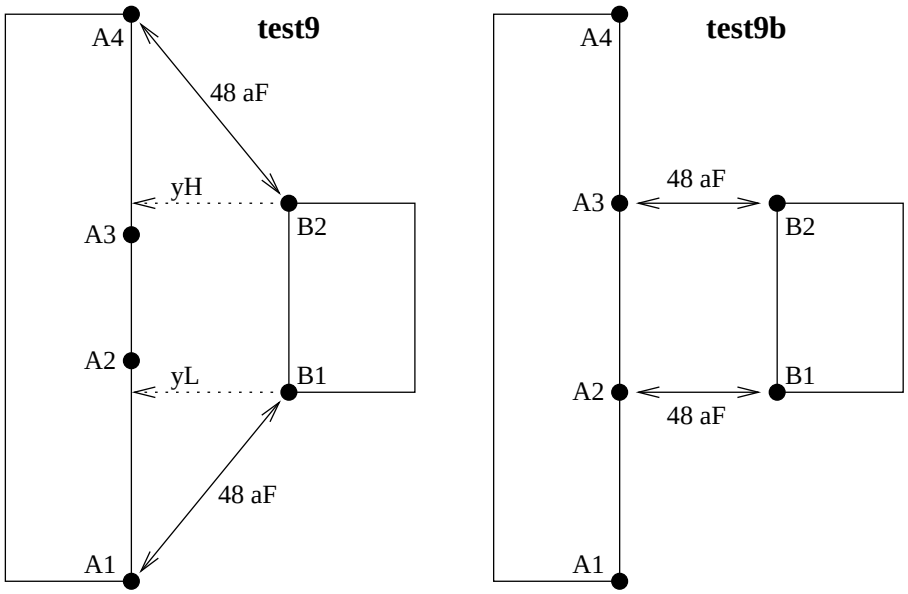


In **test7b** the latcap is done for the edge between points 1 and 2. It is internally split in a latcap for tile(a) and tile(b). The latcaps for tile(a) are assigned between points (1,3) and (2,4). And the latcaps for tile(b) are assigned between points (1,4) and (2,5).

In **test7c** the latcaps are done for edges a and b. For edge a the latcaps are assigned between points (1,4) and (2,4). For edge b the latcaps are assigned between points (2,4) and (3,5). There is no extra point on the edge between 4 and 5.

In **test7e** the latcaps are done for edges a and b. For edge a the latcaps are assigned between points (1,4) and (2,x). However, when the tile becomes ready, this extra point is removed and the latcap of 40 is split and assigned between points (2,4) and (2,5). For edge b the latcaps are assigned between points (2,4) and (3,5).

Note about the node assignment, see figures:



The lateral capacitance value is always assigned to one node. For cell **test9** the cap values are assigned to node **A1** because the node point is laying <= yL and to node **A4** because the node point is laying >= yH. Nothing is assigned to nodes A2 and A3. But by cell **test9b** the assignment can be to nodes A2 and A3.

5. TESTING

Using technology file "latcap.s", lateral capacitance extraction of the metal masks of cell top_DS01 (project PXProblemTorex5) where tested (using a lat_cap_window of 6.0 um). Part of the contents of latcap.s:

```
capacitances :
  lcap_M1    : !M1 -M1 =M1 : -M1 =M1 : 0.0522
  lcap_M2    : !M2 -M2 =M2 : -M2 =M2 : 0.0571
  lcap_M3    : !M3 -M3 =M3 : -M3 =M3 : 0.0632
```

Before extract i added 15 terminals to cell top_DS01flatM1 (M0, M1, ..., M14), and 15 terminals to cell top_DS01flatM2 (N0, N1, ..., N14), and 20 terminals to cell top_DS01flatM3 (A0, A1, ..., A13, AA0, AB0, AC0, AD0, AE0, AF0). Only mask M3 contains some slanting layout parts. To test all possible situations i used rotate and mirrow for cell top_DS01flatM3 (cell names: top_DS01flatM3mxflat, ...mxmyflat, ...myflat, ...r3flat, ...r3mxflat, ...r3mxmyflat and ...r3myflat). I did not use resistance extraction, but verified only that in each situation an equal total amount of lateral capacitances were extracted between the terminals. The compared results where all equal.

I compared the results of the OLD code against the NEW code and found for all 8 cases the same results (looks correct because OLD can not do slanting edges):

```
diff top_DS01flatM3.sls_old top_DS01flatM3.sls
<  cap 555.9122e-18 (A11, A13); >  cap 556.1359e-18 (A11, A13);
<  cap 130.5973e-18 (A0,  A13); >  cap 130.5976e-18 (A0,  A13);
<  cap 8.397989e-18 (AB0, A13); >  cap 8.514067e-18 (AB0, A13);
```

When the NEW code w/o NEW_V is compared with the NEW code, then there were some differences found. Thus the NEW_V addition to the code makes that the results are equal for all 8 cases. For 4 cases there were no differences, cells top_DS01flatM3(mx) and top_DS01flatM3r3(mx). For 2 cases there was one difference found, for the cells top_DS01flatM3my(mx):

```
<  cap 555.9122e-18 (A11, A13); >  cap 556.1359e-18 (A11, A13);
```

For 2 cases there were two differences, for cells top_DS01flatM3r3my(mx):

```
<  cap 130.5973e-18 (A0,  A13); >  cap 130.5976e-18 (A0,  A13);
<  cap 8.397989e-18 (AB0, A13); >  cap 8.514067e-18 (AB0, A13);
```

6. TESTING2

Using technology file "latcap2.s", lateral capacitance extraction of the metal masks of cell top_DS01 (project PXProblemTorex5) where tested (using a lat_cap_window of 6.0 um). Part of the contents of latcap2.s:

```
capacitances :
  lcap_M1aa  : !M1 !M2 !M3 -M1 =M1 : -M1 =M1 : 0.0522
  lcap_M1ab  : !M1 !M2  M3 -M1 =M1 : -M1 =M1 : 0.0522
  lcap_M1ba  : !M1  M2 !M3 -M1 =M1 : -M1 =M1 : 0.0522
  lcap_M1bb  : !M1  M2  M3 -M1 =M1 : -M1 =M1 : 0.0522

  lcap_M2aa  : !M2 !M1 !M3 -M2 =M2 : -M2 =M2 : 0.0571
  lcap_M2ab  : !M2 !M1  M3 -M2 =M2 : -M2 =M2 : 0.0571
  lcap_M2ba  : !M2  M1 !M3 -M2 =M2 : -M2 =M2 : 0.0571
  lcap_M2bb  : !M2  M1  M3 -M2 =M2 : -M2 =M2 : 0.0571

  lcap_M3aa  : !M3 !M1 !M2 -M3 =M3 : -M3 =M3 : 0.0632
  lcap_M3ab  : !M3 !M1  M2 -M3 =M3 : -M3 =M3 : 0.0632
  lcap_M3ba  : !M3  M1 !M2 -M3 =M3 : -M3 =M3 : 0.0632
  lcap_M3bb  : !M3  M1  M2 -M3 =M3 : -M3 =M3 : 0.0632
```

For this test we use cell top_DS01flatM123, which contains the 3 metal masks together. Using technology file "latcap.t" we see that this new cell has exact the same latcap differences:

```
diff sls_old_latcap sls_new_latcap
<  cap 555.9122e-18 (A11, A13); >  cap 556.1359e-18 (A11, A13);
<  cap 130.5973e-18 (A0,  A13); >  cap 130.5976e-18 (A0,  A13);
<  cap 8.397989e-18 (AB0, A13); >  cap 8.514077e-18 (AB0, A13);
```

Using technology file "latcap2.t" we see that the NEW extractor can handle it without any differences:

```
diff sls_new_latcap sls_new_latcap2
   NO DIFFs
```

Using technology file "latcap2.t" we see (as expected) that the OLD extractor cannot handle it, because many differences were found:

```
diff sls_old_latcap sls_old_latcap2
   MANY DIFFs
```

Conclusion, when the metal layers overlap each other on some places, then the lateral definitions in the technology file must be prepared for it. But also the extractor must be prepared for it, that it can use different latcaps (with possible different values) between the same conductor pair.

7. TESTING LAST_Y

For the OLD code we want to know what the impact of the implementation of lastY gives. This change was checked-in at 14 May 2012. This change was done for horizontal lateral capacitance extraction.

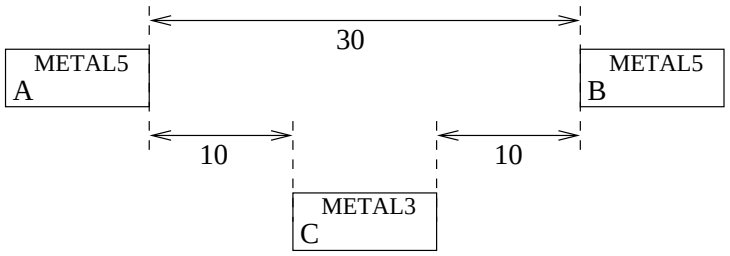
```
diff M3_sls_oldest M3_sls_old
<     cap 790.5087f (AF0, AD0); >     cap 924.5263f (AF0, AD0);
<     cap 347.0660f (AF0, AE0); >     cap 470.0272f (AF0, AE0);
<     cap 8.475926f (A0,  AE0); >     cap 72.63973f (A0, AE0);

diff M3my_sls_oldest M3my_sls_old
<     cap 913.5274f (AF0, AD0); >     cap 924.5263f (AF0, AD0);
<     cap 373.5307f (AF0, AE0); >     cap 470.0272f (AF0, AE0);
<     cap 0.776586f (A0,  AE0); >     cap 72.63973f (A0, AE0);

diff M3mx_sls_oldest M3mx_sls_old
<     cap 322.9358f (AF0, AD0); >     cap 924.5263f (AF0, AD0);
<     cap 198.2207f (AF0, AE0); >     cap 470.0272f (AF0, AE0);
<     cap 9.186187f (AD0, AB0); >     cap 19.88189f (AD0, AB0);

diff M3mxmy_sls_oldest M3mxmy_sls_old
<     cap 742.8821f (AF0, AD0); >     cap 924.5263f (AF0, AD0);
<     cap 230.6506f (AF0, AE0); >     cap 470.0272f (AF0, AE0);
<     cap 7.354047f (AD0, AB0); >     cap 19.88189f (AD0, AB0);

diff M3r3_sls_oldest M3r3_sls_old
<     cap 0.893983f (A0,  AD0); >     cap 2.297688f (A0, AD0);
<     cap 0.895214f (A0,  AB0); >     cap 9.850654f (A0, AB0);

diff M3r3my_sls_oldest M3r3my_sls_old
<     cap 0.616652f (A0,  AD0); >     cap 2.297688f (A0, AD0);
<     cap 0.718254f (A0,  AB0); >     cap 9.850654f (A0, AB0);

diff M3r3mx_sls_oldest M3r3mx_sls_old
   NO DIFFs

diff M3r3mxmy_sls_oldest M3r3mxmy_sls_old
   NO DIFFs
```

Conclusion, without lastY no consistent extraction for all 8 cell cases.

8. TEST PeterKaiser/problem1

In the project directory PX/px_results/example two cells can be found, tst2 and tst3. The lambda of this project is 0.01 um, the METAL parts have a length of 50 units (= 0.5 um). The other dimensions are shown in the figure below. The layout of cell tst2 does not contain the METAL3 part.

The layout of cell tst3:



Lateral extraction and results of cell tst2:

```
% space -l tst2
% xsls tst2

network tst2 (terminal A, B)
{
    cap 49.452e-18 (B, A);
    cap 12.60533e-18 (B, GND);
    cap 12.60533e-18 (A, GND);
}
```

Lateral extraction and results of cell tst3:

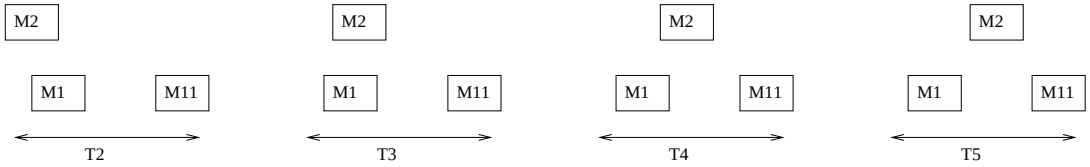
```
% space -l tst3
% xsls tst3

network tst3 (terminal A, B, C)
{
    cap 53.20305e-18 (C, GND);
    cap 47.84333e-18 (B, A);
    cap 12.60533e-18 (B, GND);
    cap 12.60533e-18 (A, GND);
}
```

The OLD space extractor could not extract the lateral capacitance between terminal A and B. The NEW space extractor extracts 47.84333e-18 in place of 49.452e-18 because this is a mean value. This mean value can be calculated, if we know what the latcap value for M5 with M3 below it is. That latcap value is 44.626e-18. Thus the mean value must be:

```
10 / 30 * 44.626e-18 + 20 / 30 * 49.452e-18 = 47.84333e-18
```

9. TEST PeterKaiser/quest24nov(2011)

In the project directory PX_test_2/px_results/example we can find some problem cells. The layout of cell example_M1_M2_M1:



Lateral extraction and results for OLD space:

```
% space_ -l -E simon.t example_M1_M2_M1
% xsls example_M1_M2_M1

network example_M1_M2_M1 (terminal T1_M11, T1_M1, ...)
{
    cap 2.521191f (T1_M11, T1_M1);
    cap 2.521191f (T2_M11, T2_M1);
    cap 35.23403e-18 (T3_M11, T3_M1);
    cap 27.40425e-18 (T4_M11, T4_M1);
    cap 24.79432e-18 (T5_M11, T5_M1);
}
```

Lateral extraction and results for NEW space:

```
% space -l -E simon.t example_M1_M2_M1
% xsls example_M1_M2_M1

network example_M1_M2_M1 (terminal T1_M11, T1_M1, ...)
{
    cap 2.521191f (T1_M11, T1_M1);
    cap 2.521191f (T2_M11, T2_M1);
    cap 2.502784f (T3_M11, T3_M1);
    cap 2.170358f (T4_M11, T4_M1);
    cap 1.929696f (T5_M11, T5_M1);
}
```

Lateral definition part of the technology file used:

```
capacitances: # lateral capacitances
  lcap_METAL1_none_none : !METAL1 -METAL1 =METAL1 !METAL2 : -METAL1 =METAL1 :
                          2e-07 0.11007 4e-07 0.068965 8e-07 0.042328 ...
  lcap_METAL1_none_M2   : !METAL1 -METAL1 =METAL1 METAL2 : -METAL1 =METAL1 :
                          2e-07 0.090354 4e-07 0.048477 8e-07 0.021222 ...
```