Space Transistor Data Structures

S. de Graaf

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

Report EWI-ENS 10-02
June 15, 2010

Last revision: July 14, 2010.

1. INTRODUCTION

With the term *transistor* in this report i want only describe the Field Effect Transistor (FET) implementation as specified in the "fets" section of the *space* technology file.
The FET is an unipolar transistor which has in general three connections: the source (S), the drain (D) and the gate (G). In some occasions there is a fourth connection for the substrate or bulk (B).

The FET contains a conducting channel between the source (S) and the drain (D), of which the conductance can be controlled by the electric field of the voltage on the gate (G). When conducting, the current can flow in both directions, from source to drain or drain to source. Thus, the source and drain connections are equal and can be interchanged. This transistor is most times used as a switch in digital circuits. The P-channel type FET is only conducting by a low or zero voltage on the gate and the N-channel type is only conducting by a high enough positive voltage on the gate.

The connections (or pins) of the transistor are connected to circuit nodes. Nodes are conducting parts in the circuit. The smallest part of a node is a subnode, which can be in a layout tile or on the edges of the layout tile in the case when resistors are extracted.
In case nodes are connected by resistors, we speak about a node group. The node group stands for a complete conducting wire or path. When the last node of the group becomes ready, then the group becomes ready. After that it can be reduced (node reduction). Important nodes can not be deleted. That are nodes connected with terminals or labels or connected with transistor pins.

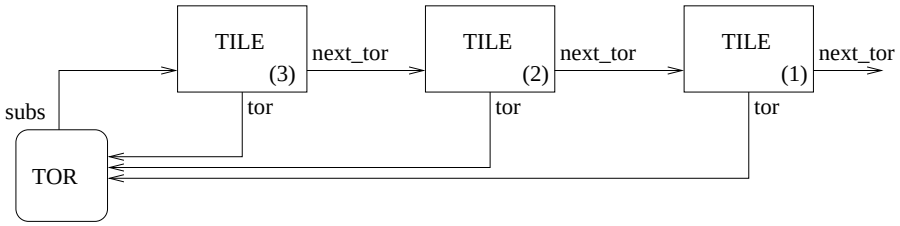
When a layout is extracted, the layout masks are combined and scanned from left to right. The layout is split up in tiles of which the edges first are processed. Function enumPair is called for each edge between two tiles and is used to recognize edge elements. When all edges of a tile are done the tile becomes ready and function enumTile is called. This function handles all surface elements and does the resistor mesh generation.

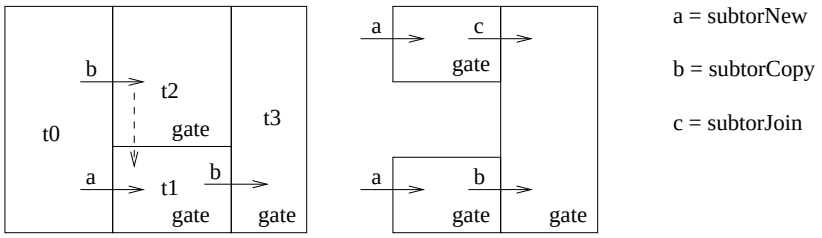
After enumTile is ready function clearTile is called. This function deletes the tile, but because the tile contains subnodes, first all the subnodes of the tile must be deleted (function subnodeDel). In case a tile also contains a part of a transistor, also subtorDel must be done. If subnodeDel deletes last subnode of a node, the node becomes ready. Function readyNode is called, it decrements the group node counter and shall put the node in the delay queue (if it is unimportant and can be eliminated). Only if the delayed queue becomes full, one node with the lowest degree is eliminated. This can be a node of one of the not ready node groups.

When the last node of a group becomes ready (the group notReady counter becomes zero), then function readyGroup is called. This function shall first eliminate all delayed nodes in lowest degree order. After that a number of reduction heuristics are done on the remaining nodes. The group is outputted (calling function outGroup and outNode) when all neighbor groups are ready. This, because the reduction heuristics must first be done for neighbor groups. Only thereafter no connections are more changed.

2. THE TOR DATA STRUCTURE

A tile can only point to one transistor (tor) data structure. When the gate of the transistor is split in pieces, then different tiles points to the same transistor. Function subtorNew is used when a tor element surface condition match is found. Note that this mask condition must be the condition for the gate of the transistor. In cases that a neighbor tile already contains the transistor function subtorCopy is used. In some occasions subtorJoin must be used, this can only happen by some fancy gate layouts. When a tile is finished function clearTile is called. When a tor is in the tile, clearTile shall call function subtorDel for it. Function subtorDel shall unlink the tile from the tile tor list. When the last tile of the gate becomes ready, tor->subs becomes NULL, then subtorDel shall call function outTransistor.





When doing enumPair of two tiles, doing the edge between tile t0 and t1, tile t0 does not contain a transistor gate and t1 does contain one. Therefor subtorNew is done for tile t1. For tile t2 subtorCopy can be done, however not from tile t0, but from tile t1. This is possible[1] because tile t2->stb (stitch bottom) points to tile t1.

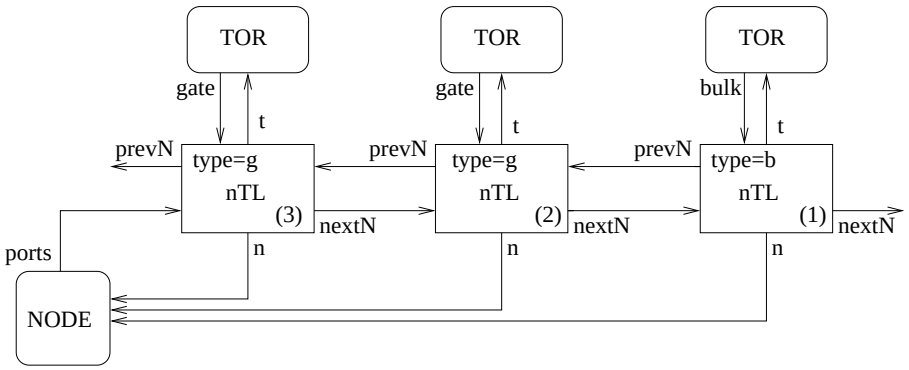
In the second figure are two separate tor structures allocated. When both tors become together, they must be joined. One tor structure is deallocated and its tile is linked in the list of the other tor and it must point to the other tor.

1. Using the stitch bottom pointer eliminates in most cases the use of function subtorJoin. I have tested this with the FreePDK45demo "I8051_ALL" example. This gives more than 2 % time improvement.

3. THE NODE-TOR-LINK DATA STRUCTURE

Between the TOR data structure and the NODE data structure there is made a nodeTorLink (nTL) data structure. This is done by function portAdd, which is called by enumTile. The portAdd is done for the gate node and optional for the bulk node. Note that for the drain and source nodes no portAdd is done (obsolete code). Note that the TOR data structure only is made in the extract pass (not in a prepass). Note that by resistance extraction resEnumTile is called. In that case a tile can contain different nodes at each tile corner. If tile split is true, then always the top-right node is used, and portAdd is always called to replace a previous nodeTorLink by a new one. This is not done by enumTile, because the nodeTorLink points always to the correct node and by node joins function nodeRelJoin takes care to relink it correctly.

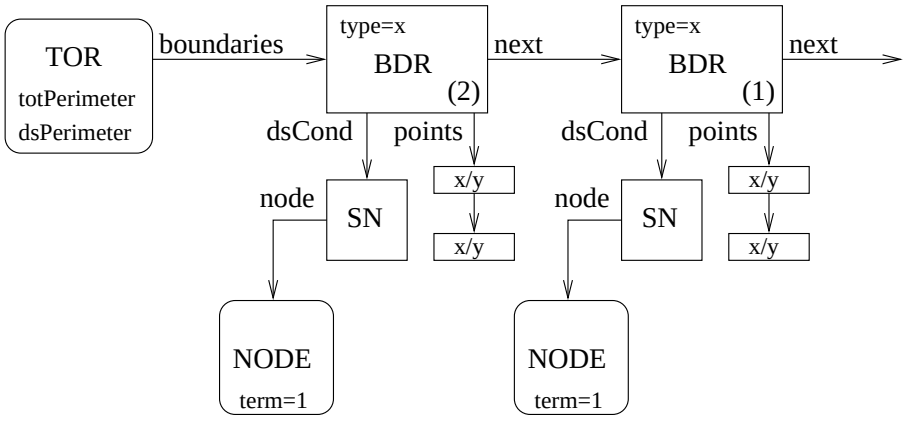
Function portAdd calls nTLinkAdd to add a nodeTorLink. There is only made a new nodeTorLink between the NODE and the TOR of the requested type (gate or bulk) if it is not already there. Because a TOR data structure only can point with member "gate" (or "bulk") to one nodeTorLink, therefor the nextT and prevT pointers of the nodeTorLink structure are obsolete. However, the NODE data structure can point with member "ports" to different nodeTorLink structures. There is made a double linked list with nextN and prevN nodeTorLink members. The nodeTorLink list can contain two nodeTorLinks to the same transistor if the gate and bulk are connected with each other. Note that a NODE can only have one nodeTorLink of a type (g/b) to one transistor.



Because the notReady counter of the node group is incremented for each nodeTorLink a node group can only become ready after all nodeTorLinks of the group are deleted. Thus, after the last gate tile of a transistor is done and subtorDel and outTransistor is called. Function outTransistor shall call function nTLinkDel to delete the gate/bulk nodeTorLink. Thus, the nodeTorLink data structure is also a lock which prevents that a node group becomes ready and also prevents that a ready node becomes delayed. Note that outTransistor also sets the "term" flag of the gate (and bulk) node. This is also enough for readyNode not to delay the node.

4. THE DRAIN/SOURCE-BOUNDARY DATA STRUCTURE

A transistor gate area must be surrounded by two drain/source areas. Function enumPair shall call function updateTorEdge for one or both of the tiles, if a transistor gate is in the tile. When it is not an internal gate edge, then the edge length is added to the total perimeter of the gate. When a drain/source conductor is found in the adjacent tile, then the edge length is also added to the total ds-perimeter around the gate and function torBoundary is called. Note that by resistance extraction the nodes on the edge are joined together. When the transistor uses a different conductor for the drain and source area, then the correct type (d or s) must be given to function torBoundary. But normally the drain and source conductor are equal and type 'x' is used.



Function torBoundary sets the "term" flag of the d/s node, because the node may not be delayed and eliminated. If the transistor already has a boundary data structure, then is checked of the existing boundary must be extended. It can also be possible, that two points are found for the new boundary edge. In that special case two existing boundaries must be joined together. Thus, it is also possible that the complete gate is surrounded by one closed boundary.

When it is a new boundary, a new boundary data structure is allocated and also two point data structures. Besides that, also a boundary subnode is allocated and this subnode points to the d/s node. This is done with function subnodeCopy. Note that this is a very special subnode, because it is not living in a tile. This subnode is a lock for the d/s node, because the d/s node can only become ready if this boundary subnode is deleted (with subnodeDel) and this is done by function outTransistor.
Thus, the node groups of all the transistor pins can only become ready after the transistor (gate area) is ready.

Function outTransistor shall attach a netEq data structure to the d/s nodes with pin type 'x'. The first d/s node which is outputted by function outNode receives type 'd' and the other node type 's'. This is possible because both netEq's are connected with each other (by a netEq ring).

5. THE NODE TERM FLAG

The node term flag is inited by function createNode normally to 0, but is inited to 2 when optFineNtw is true (option -%f). The node term flag is set to 1 for each terminal/label connected to the node (see function nameAdd). Also the term flag of nodes connected to transistor pins are set to 1. This make these nodes more important, see function readyNode, a node with the term flag set shall not be delayed and eliminated. Only by some reduction heuristics a ready node of a ready group can be eliminated (in that case a nodeRelJoin is done).

For drain/source pins the node term flag is set by function torBoundary. However, for the gate and bulk pins the node term flag is set in function outTransistor and only when the transistor is not skipped. Therefor the node term flag does not need to be cleared for these pins. Note that these nodes can only become ready after nTLinkDel, when the node "ports" member is cleared.

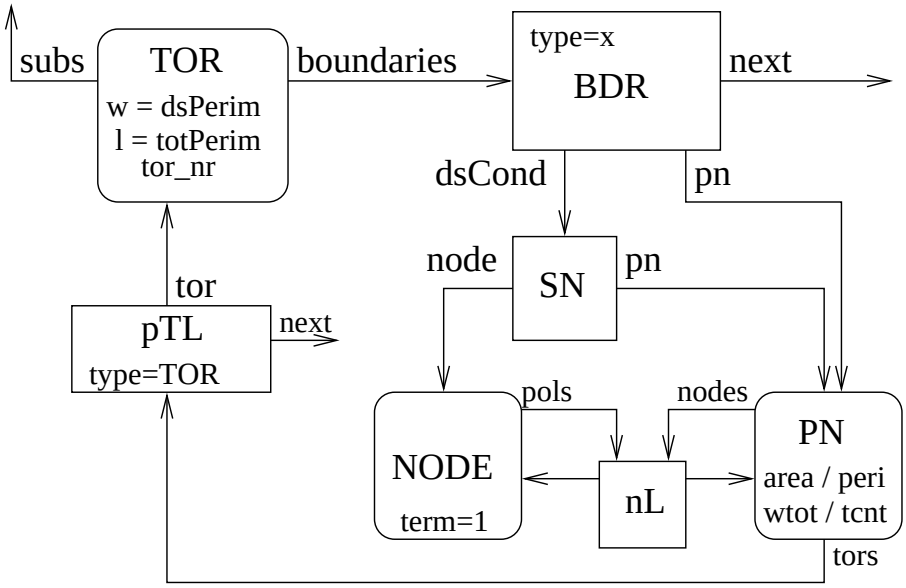
For a drain/source pin, however, the node term flag should be cleared if the transistor is skipped. For example, because the transistor has only one drain/source pin. This is however not possible, because we can not figure out if other transistors are connected to the same node. We can only find all other transistors via the node "ports" member, if also for drain/source nodes nodeTorLinks are made. See for example function termNodeClear which is used to clear the node term flag for BJTs. The node term flag can only be cleared if (1) term < 2 and (2) the node has no "names" and (3) the node has no other devices (no other BJTs). Note that the other devices can be found via polnode-tor-links. However otherDevices does not look for normal transistors via node-tor-links!

Thus we have a problem to clear the node term flag correctly. Maybe we must use another bit in the term flag for transistors as well. Then we only need to look for other transistors. We can also only set the node term flag for drain/source nodes in outTransistor when we know that the transistor is not skipped. Note that a drain/source node can not be ready, because we use a special boundary subnode. Only if this boundary subnode is deleted (subnodeDel) the node can become ready.

6. USING POLNODES FOR DRAIN/SOURCE AREAS

We can use polnodes for the drain/source areas and store the area and perimeter of the drain/source in the polnode data structure. We need only do this when a dscap is specified in the technology file for the transistor. The area of the dscap can be smaller than the area of the polnode. Therefor we must use the surface condition of the dscap to set the drain/source area and we must use the edge condition to set the drain/source perimeter. When we use polnodes, we don't need to generate ground capacitances for the dscaps. With these polnodes *space* can calculate the d/s parameters itself.

Function torBoundary assigns the polnode to the new boundary data structure if the tor element data structure tells us that the transistor has dsCap. The given drain/source subnode must in that case contain a polnode. Also a polnode-Tor-Link (pTL) is added (with function pnTorLinkAdd). Only a polnode can point to this pTL and must have only pTL for each transistor. The pTL must be of type TORELEM, because then we know that it points to a transistor. Other types points to BJT transistors.



You see that the boundary itself also points to the polnode. Note that the BJT data structure does this with a polnode array for its pins. When the subnode is deleted, the subnode cannot be used for this purpose. Note that the node and polnode are also connected with each other by the nodeLink (nL) data structure. A polnode has always a connection with a node and both belong always to the same node group. When no resistors are extracted then one node can have more than one polnode, because different conductors can be joined together by vias. When resistors are extracted then one polnode normally has more than one node, because all nodes of a conductor in a tile points to the same polnode. Note that a pnTorLink is not a locker (like the nodeTorLink), however, the nodeLink is a locker (and increments group->notReady).

7. TESTING WITH FREEPDK45 I8051_ALL

The FreePDK45 technology file contains a *fets* section with drain/source-cap definitions to get the drain/source parameters in the *spice* netlist (see detail below).

```
fets:
# name:condition_list:mask_g mask_ds [ds_cap] [:mask_b]
  nenh:(!nwell n_active poly):poly active (!nwell n_active !poly):@sub
  penh:( nwell p_active poly):poly active ( nwell p_active !poly):nwell
```

With the new *space* extractor, which is not using subtorJoin and extracts the drain/source parameters itself with polnodes, the I8051_ALL cell can be extracted with the following statistics:

```
% space -Fv I8051_ALL
    nodes              : 54016
    mos transistors    : 126069
    memory allocation  : 9.198 Mbyte
    user time          : 21.2 second
```

With the *space* -i option we can find out that 165555 polnodes are used.
The old method can still be used by specifying the "old_ds_caps" parameter:

```
% space -Fv -Sold_ds_caps I8051_ALL
    nodes              : 54017
    mos transistors    : 126069
    d/s capacitances   : 83077
    memory allocation  : 9.533 Mbyte
    user time          : 20.8 second
```

You see that the old method is faster and uses one more node (GND). Note that the amount of d/s capacitances is too low, this because the d/s nodes are joined with vias.
When the d/s parameters are omitted the extraction is much faster:

```
% space -Fv -Somit_ds_caps I8051_ALL
    nodes              : 54016
    mos transistors    : 126069
    memory allocation  : 9.094 Mbyte
    user time          : 18.2 second
```

Note that *space* detects 2260 transistors with only one drain/source terminal. Default, *space* does not add d/s terminals (use parameter "add_ds_terms" if you want this) and does not skip these transistors. The maximum internal kept nodes and mosfets is not equal for the first two cases. The first case kepts more nodes (2628 vs. 2594) and fets (371 vs. 180), this because the fets are later ready.
Capacitance extraction (option -C) for the above 3 cases gives the following results:

```
    user time (second):  37.1   36.6   35.1
```

Resistance extraction (option -r) for the above 3 cases gives the following results:

```
    user time (second):  9600.5   9555.1   9443.8
```

8. SEPARATE D/S BOUNDARIES

Parameter "separate_ds_boundaries" can be used to set separate d/s boundaries. Default, when "off", variable optDsConJoin is TRUE. This gives the following default behaviour:

8.1 When there is NO d/s boundary

The transistor does not have any d/s terminal pins. There are no terminals added, because this is not possible. Use parameter "omit_incomplete_tors" to skip the transistor.

8.2 When there is only ONE d/s boundary

The transistor does not have two d/s terminal pins. Default, the first d/s terminal is also used for the second terminal. Thus, the d/s terminals are "joined" (are not separate). This is only possible if the transistor does not have separate drain and source conductors. The transistor can be skipped with parameter "omit_incomplete_tors".

Note that with "separate_ds_boundaries=on" no second d/s terminal is added.

8.3 When there are exact TWO d/s boundaries

Then normally everything is ok and no d/s terminals are skipped. Only exception, if there are two drain or two source terminals. Default, with optDsConJoin is TRUE, try to skip one (if unconnected).

8.4 When there are more than TWO d/s boundaries

Default, with optDsConJoin is TRUE, extra d/s terminals can be skipped. This is only possible when one of the d/s terminal nodes is unconnected. The netEquiv "termSkip" flag is used for this purpose. With "separate_ds_boundaries=on" no d/s terminals are skipped. Thus, a transistor can have too many d/s terminal pins.

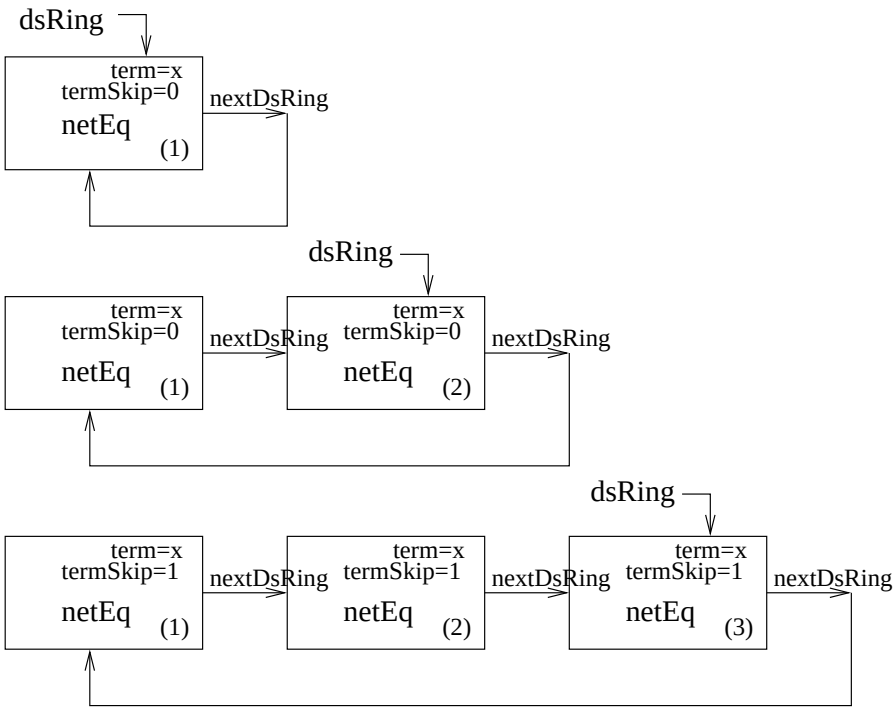
8.5 What is an unconnected node?

An unconnected node is a node which does not have resistors or capacitors and does not have label or terminal names. When this unconnected node does have a netEquiv to a transistor it is not unconnected, but if it is the only netEquiv it has, then this transistor pin is not connected to something else.

9. RING OF D/S BOUNDARIES

The netEquivalences of d/s terminal nodes are always linked together in a d/s-ring.
The d/s-ring is used for:

1. Setting the 'x' names to 'd' and 's' names. The first outputted (not skipped) terminal gets the name 'd'. All other terminals the name 's'.
2. For skipping d/s terminal nodes by optDsConJoin is TRUE, but only when there are too many and possibly one (or more) can be skipped (if unconnected).



We don't want to use the d/s-ring anymore. Because for the new implementation we don't want to set the 'x' names in outNode. We must set the names when outputting the transistor with its parameters. Also the possible d/s terminal skipping method is not important.