LSXLIB

A Lambda Based Standard Cell Library

May 6, 2023 - v 1.0

Principles

LSXLIB is a library of characterized CMOS cells. The cells describe elementary Boolean expressions used by logic synthesis tools to create complex blocks. Each cell contains a small number of transistors (from 1 to 50) implementing a complete and independent expression, i.e. a cell does not have to be combined with another one to implement a Boolean expression. The cells are named in such a way that it is possible to recognize the implemented expression. Each cell is correct from the point of view of the drawing rules, but the rules are those of a virtual, so-called symbolic technology, for which the length of the channel is worth one lambda unit. The cells have a common physical template that allows them to be butted together to create the complex blocks. The abutment rules ensure that there are no constraints on the choice of neighboring cells for each cell. The symbolic technology uses a two-dimensional grid with a grid pitch of 1 lambda on which 2 types of symbols are placed: segments and contacts. The transformation of symbolic drawings into real masks is done symbol by symbol thanks to explicit transformation rules depending on the target technology. The lambda value and the transformation rules are determined by a semi-automatic process configured for each target technology. LSXLIB cells are created in symbolic technology and transformed into the target technology to be used by frontend and backend CAD tools (synthesis, placement, routing, DRC, etc.).

In this document you will find the details of the principles mentioned above. First the list of cells and their naming algorithm, then the definition of the symbolic rules and the template rules, the principle of transformation of symbols into real masks, the procedure of lamba calculation and finally the rules of symbolic transformation to sky130 technology.

LSXLIB cells

It is possible to deduce the behavior of a LSXLIB cell by knowing its name. This is very intuitive for inversors, and a little more complex for cells combining AND-ORs. For the latter, the name gives the structure of the tree and/or.

Name of the cells

inv	**inv**ersor buffer
buf	**buf**fer
[**n**]**ts**	[not] **t**ree-**s**tate
[**n**]**xr**<i>	[not] **x**o**r** <i> inputs
[**n**]**mx**<i>	[not] **m**ultiple**x**or <i> inputs with coded command
[**n**]**[sd]ff**<i>	[not] **s**tatic or **d**ynamic **f**lip-**f**lop <i> inputs
[**n**]**oa**[...]	[not] **o**r/**a**nd function (see below)

and_or cell (BNF type):

```
• NAME       ::= nOA_CELL                   → not OA_CELL
               |  OA_CELL                   → OA_CELL
• OA_CELL    ::= OPERATOR INPUTS            → function with INPUTS inputs
               |  OPERATOR OA_CELLS INPUTS  → function with INPUTS inputs
                                                where some inputs are OA_CELL
• OPERATOR   ::= a                          → and
               |  o                         → or
               |  n                         → not
• OA_CELLS   ::= OA_CELLS OA_CELL           → list of OA_CELL
               |  OA_CELL                   → last OA_CELL of the list
• INPUTS     ::= integer                    → number of inputs
```

Remarks:

- The input names are implicit and formed as follows: `i<number>`.
  They are assigned in order starting with `i0`.
- `nx` where x is a number means that there are x inversors in parallel.
- For example, `an23` is an and with 3 entries of which two are inverted,
  i.e. : `and( not(i0)`, `not(i1), i2).`

Examples of cell names

Some names correspond to cells absent from LSXLIB

```
• na2      not( and(i0,i1))
• an12     and( not(i0), i1)
```

- `noa2a22     not( or( and(i0,i1), and(i2,i3))`
- `noa23       not( or( and(i0,i1), i3))`
- `noao22a34   not( or( and( or(i0,i1), i2), and(i3,i4,i5), i6, i7))`

Note that `xr2` cannot be represented with a formula or/and even if `xr2 = or( and( not(i0), i1), and( not(i1), i0))`, it is different of `oan12an122`, because the input names are not in the right order.

List of cells

The cells are sorted by type. For each, the first number is equal to the width in lambdas, followed by the name of the cell, and the boolean expression of its output depending on the inputs. The output of the cells is represented by the suffix _x<*value*>. The suffix `_x1` means that the cell can command from 5 to 10 `inv_x1` cells. Cells with the suffix _x2, _x4, `_x8` are respectively 2, 4, and 8 times more powerful. These are indicative values, the real values obviously depend on the technology.

The transistor diagram of some cells is given at the end of the documentation.
In Boolean expressions:`/ = NOT ; . = AND ; + = OR ; ^ = XOR.`

INVERSOR / BUFFER

```
15   inv_x1           nq <= /i
15   inv_x2           nq <= /i
20   inv_x4           nq <= /i
35   inv_x8           nq <= /i
20   buf_x2            q <= i
25   buf_x4            q <= i
40   buf_x8            q <= i
```

THREE STATE

```
30   nts_x4           IF (cmd) nq <= /i
40   nts_x8           IF (cmd) nq <= /i
50   ts_x4            IF (cmd) q <= i
65   ts_x8            IF (cmd) q <= i
```

AND

```
20   na2_x1           nq <= /(i0.i1)
35   na2_x4           nq <= /(i0.i1)
25   na3_x1           nq <= /(i0.i1.i2)
40   na3_x4           nq <= /(i0.i1.i2)
30   na4_x1           nq <= /(i0.i1.i2.i3)
50   na4_x4           nq <= /(i0.i1.i2.i3)
25   a2_x2             q <= (i0.i1)
30   a2_x4             q <= (i0.i1)
30   a3_x2             q <= (i0.i1.i2)
35   a3_x4             q <= (i0.i1.i2)
```

35	a4_x2	q <= (i0.i1.i2.i3)
40	a4_x4	q <= (i0.i1.i2.i3)
25	an12_x1	q <= (/i0).i1
40	an12_x4	q <= (/i0).i1

OR

20	no2_x1	nq <= /(i0+i1)
40	no2_x4	nq <= /(i0+i1)
25	no3_x1	nq <= /(i0+i1+i2)
40	no3_x4	nq <= /(i0+i1+i2)
30	no4_x1	nq <= /(i0+i1+i2+i3)
50	no4_x4	nq <= /(i0+i1+i2+i3)
25	o2_x2	q <= (i0+i1)
30	o2_x4	q <= (i0+i1)
30	o3_x2	q <= (i0+i1+i2)
50	o3_x4	q <= (i0+i1+i2)
35	o4_x2	q <= (i0+i1+i2+i3)
40	o4_x4	q <= (i0+i1+i2+i3)
25	on12_x1	q <= (/i0)+i1
40	on12_x4	q <= (/i0)+i1

AND/OR 3 inputs

30	nao22_x1	nq <= /((i0+i1).i2)
50	nao22_x4	nq <= /((i0+i1).i2)
30	noa22_x1	nq <= /((i0.i1)+i2)
50	noa22_x4	nq <= /((i0.i1)+i2)
30	ao22_x2	q <=(i0+i1).i2)
40	ao22_x4	q <=(i0+i1).i2)
30	oa22_x2	q <=(i0.i1)+i2)
40	oa22_x4	q <=(i0.i1)+i2)

AND/OR 4 inputs

35	nao2o22_x1	nq <= /((i0+i1).(i2+i3))
55	nao2o22_x4	nq <= /((i0+i1).(i2+i3))
35	noa2a22_x1	nq <= /((i0.i1)+(i2.i3))
55	noa2a22_x4	nq <= /((i0.i1)+(i2.i3))
45	ao2o22_x2	q <=(i0+i1).(i2+i3))
50	ao2o22_x4	q <=(i0+i1).(i2+i3))
45	oa2a22_x2	q <=(i0.i1)+(i2.i3))
50	oa2a22_x4	q <=(i0.i1)+(i2.i3))

AND/OR 5 inputs

35	noa2ao222_x1	nq <= /((i0.i1)+((i2+i3).i4))
55	noa2ao222_x4	nq <= /((i0.i1)+((i2+i3).i4))
50	oa2ao222_x2	q <=(i0.i1)+((i2+i3).i4))
55	oa2ao222_x4	q <=(i0.i1)+((i2+i3).i4))

AND/OR 6 entries

50	noa2a2a23_x1	nq <= /((i0.i1)+(i2.i3)+(i4.i5))
65	noa2a2a23_x4	nq <= /((i0.i1)+(i2.i3)+(i4.i5))
60	oa2a2a23_x2	q <=(i0.i1)+(i2.i3)+(i4.i5))
65	oa2a2a23_x4	q <=(i0.i1)+(i2.i3)+(i4.i5))

AND/OR 7 entries

45	noa3ao322_x1	nq <= /((i0.i1.i2)+((i3+i4+i5).i6))
55	noa3ao322_x4	nq <= /((i0.i1.i2)+((i3+i4+i5).i6))
50	oa3ao322_x2	q <=(i0.i1.i2)+((i3+i4+i5).i6))
55	oa3ao322_x4	q <=(i0.i1.i2)+((i3+i4+i5).i6))

AND/OR 8 entries

70	noa2a2a2a24_x1	nq <= /((i0.i1)+(i2.i3)+(i4.i5)+(i6.i7))
85	noa2a2a2a24_x4	nq <= /((i0.i1)+(i2.i3)+(i4.i5)+(i6.i7))
75	oa2a2a2a24_x1	q <= ((i0.i1)+(i2.i3)+(i4.i5)+(i6.i7))
80	oa2a2a2a24_x4	q <= ((i0.i1)+(i2.i3)+(i4.i5)+(i6.i7))

MULTIPLEXER

35	nmx2_x1	nq <= /((i0./cmd)+(i1.cmd))
60	nmx2_x4	nq <= /((i0./cmd)+(i1.cmd))
45	mx2_x2	q <=(i0./cmd)+(i1.cmd))
50	mx2_x4	q <=(i0./cmd)+(i1.cmd))
60	nmx3_x1	nq <= /((i0./cmd0)+((i1.cmd1)+(i2./cmd1)).cmd0)
75	nmx3_x4	nq <= /((i0./cmd0)+((i1.cmd1)+(i2./cmd1)).cmd0)
65	mx3_x2	q <=(i0./cmd0)+((i1.cmd1)+(i2./cmd1)).cmd0)
70	mx3_x4	q <=(i0./cmd0)+(((i1.cmd1)+(i2./cmd1)).cmd0))

XOR

45	nxr2_x1	nq <= /(i0^i1)
55	nxr2_x4	nq <= /(i0^i1)
45	xr2_x1	q <= (i0^i1)
60	xr2_x4	q <= (i0^i1)

FLIP-FLOP

90	sff1_x4	IF RISE(ck) q <= i
120	sff2_x4	IF RISE(ck) q <= ((i0./cmd)+(i1.cmd))
140	sff3_x4	IF RISE(ck) q <= (i0./cmd0)+(((i1.cmd1)+(i2./cmd1)).cmd0)

SPECIAL

15	zero_x0	nq <= '0'
15	one_x0	q <= '1'
5	rowend_x0	Empty cell

Symbolic design rules

The symbolic technology on lambda grid defines 2 main types of symbols: segments and contacts. Segments are used to represent transistors and wires, but also boxes and connectors. The contacts are used to connect segments of different types. In addition to segments and contacts, there is also the abutment box, a symbol defining the border of each cell.

A segment symbol is defined by its type, the coordinates of its ends and its width. Each segment type has a minimum length and width. A contact symbol is defined by its type and the coordinates of its center.

For LSXLIB, only the types of symbols needed for cell design are defined. These are segments for transistors, segments for interconnecting transistors, segments for biasing cells and contacts between all these segments. There are no symbols for the types of metal interconnecting the cells, these types are managed by the routing tools.

The symbolic design rules define how to physically realize the cells. The rules can be broken down into 2 subsets. There are, on the one hand, the **LSXLIB template rules** which define the shape of the cells chosen for LSXLIB and the constraints on the location of the internal symbols and, on the other hand, the **symbolic drawing rules** which define the shape and the relative position of the symbols realizing the netlist of the transistors

In the first part of this section, we first introduce the role and geometry of symbols, then we give the drawing rules, and finally the template rules.

Role of the segment symbols

The segments are classified in 3 categories: first the segments used for the signals; then the segments used for the transistors and finally the segments used for the well and their polarization, boxes in which the transistors are drawn.

Signal segments

POLY	Polysilicon for the interconnection of transistor gates
NDIF	N-doped active area for interconnection of drains and N-transistor sources.
PDIF	P-doped active area for interconnection of P-transistor drains and sources.
ALU1	First level of metal for local interconnection of transistors
	The name ALU is historical, but it is usually not aluminum
CALU1	Normal ALU1 segment, but used as a connector.
	CALU1 segments must follow the same rules as ALU1.

Transistor segments

It is not allowed to create a transistor by crossing polysilicon wire with a diffusion wire.

NTRANS	Transistor N, only the polysilicon segment realizing the gate of the transistor is drawn. For a transistor, this single segment represents not only the gate, but also the drain and the source which are placed on both sides of this gate.
PTRANS	Same as NTRANS, but for P transistors.

WELL segments

The use of these segments is imposed by the template rules.

NTIE	Zone active N-doped for NWELL polarization.
PTIE	Zone active P-doped for PWELL or P-substrate.
NWELL	containing the P transistors.
PWELL	containing the N transistors. The PWELL must be drawn even if the target technology does not contain oneThe reason for this constraint is explained in the section on transforming symbolic drawings into real masks.

Role of the contact symbols

Each type of contact makes it possible to connect an ALU1 segment with a segment of a different type (POLY, NDIF, etc.). There are as many types of contact as there are signal segments, i.e. 4, plus 2 others for the polarization of the box, whose position is imposed by the cell template. There are no specific contacts for the transistor segments. In fact, the transistors are composed of a combination of polysilicon (POLY) and diffusion (NDIF or PDIF) segments and, consequently, it is the contacts used for the POLY and the DIF that are used.

CNDIF	Contact ALU1 - NDIF
CPDIF	Contact ALU1 - PDIF
CPOLY	Contact ALU1 - POLY
CNTIE	Contact ALU1 - NTIE whose use is imposed by the template rules.
CPTIE	Contact ALU1 - PTIE whose use is imposed by the template rules.

Geometry of segment symbols

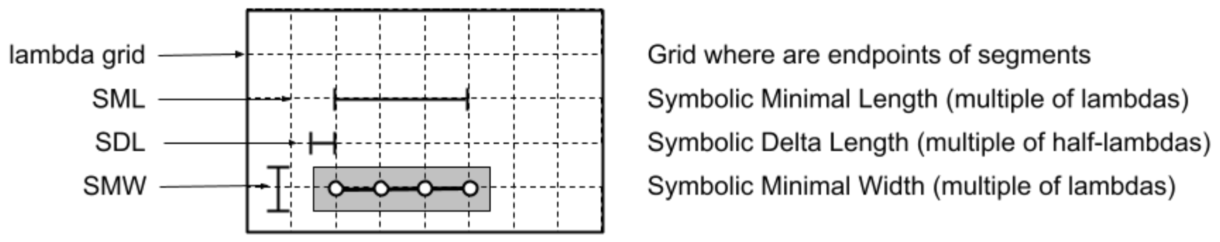
Signal segments

We define 3 quantities to qualify a signal segment: length, extension and width.

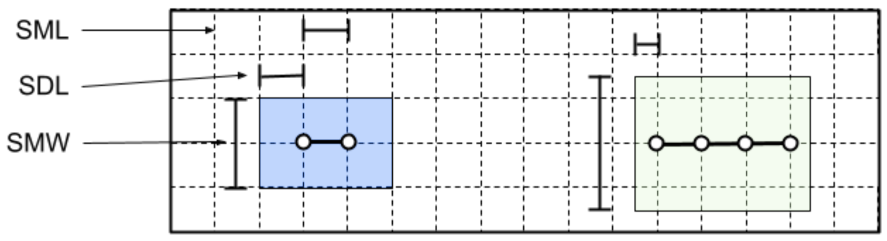
- **SML** (Symbolic Minimal Length) means that it is forbidden to draw smaller segments, but it is possible to draw larger segments in steps of 1 lambda.
  SL is the actual symbolic length of the drawn segment.

- **SDL** (Symbolic Delta Length): it is an extension of constant length which depends only on the type of segment, this extension is a multiple of the half-lambda.
- **SMW** (Symbolic Minimal Width) means that it is forbidden to draw smaller segments, but it is possible to draw larger segments in steps of 1 lambda.
  SW is the effective symbolic width of the segment.

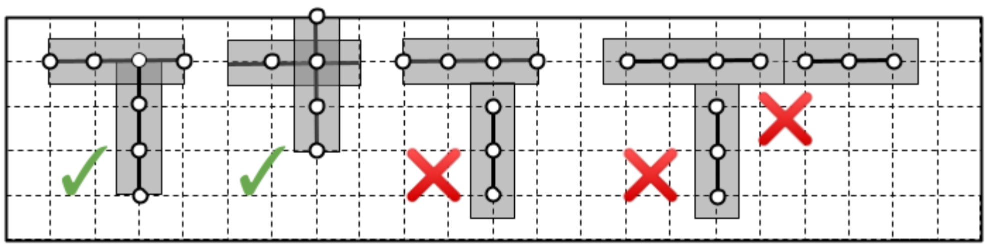
In the drawing below, we represent an area of the lambda grid on which a segment is drawn at its minimum size. The circles at the intersections of the grid and the segment are the junction points of the segment.


lambda grid
SML
SDL
SMW
Grid where are endpoints of segments
Symbolic Minimal Length (multiple of lambdas)
Symbolic Delta Length (multiple of half-lambdas)
Symbolic Minimal Width (multiple of lambdas)

These 3 quantities (SML, SDL and SMW) are independent, here 2 examples: ALU1 and NDIF.



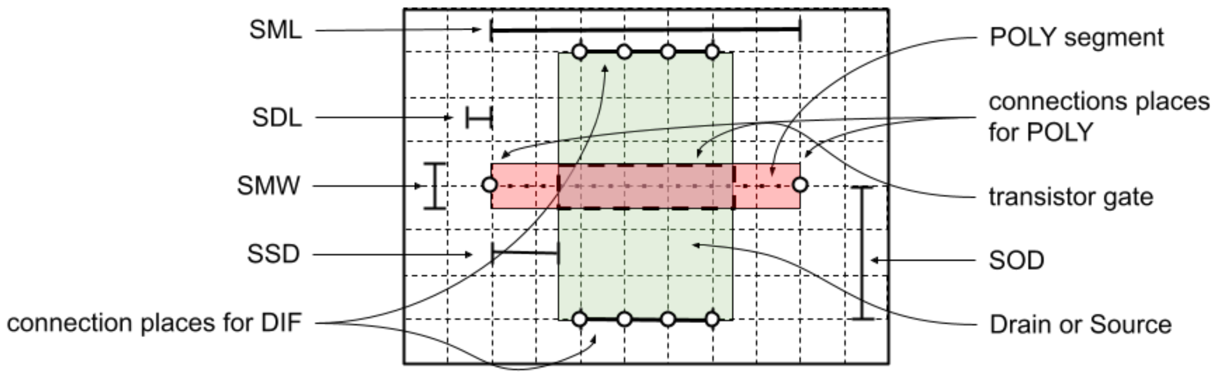
Two segments are in contact if their core (center) touches at the junction points. In the drawing, the junction points are at the ends of the segments or at the crossings of the grid, the first two patterns are correct, the next two are not because the centers of the segments are not touching.



NTRANS and PTRANS transistor segments

We draw only the gate of the transistors and not their drain and source. The gate is in polysilicon and the drains and sources are in NDIF for N transistors or in PDIF for P transistors.

For the connection to the transistors, the polysilicon (POLY) segments can connect the gate of the transistor only at its ends (where there are **o** on the schematic below), while the diffusion segments can connect the drains or sources on both sides of the gate (where there are **o** on the schematic below).



We define 5 quantities to qualify a type of transistor segment.

- **SML** (Symbolic Minimal Length of the polysilicon segment) meaning that it is forbidden to draw a smaller transistor, but it is possible to increase the length of the transistor segment by steps of 1 lambda.
  SL is the effective symbolic length of the polysilicon segment.
- **SDL** (Symbolic Delta Length of the polysilicon segment) : it is a constant length not depending on the type of transistor.
- **SMW** (Symbolic Minimal Width of the polysilicon segment) means that it is forbidden to draw smaller gates, but for LSXLIB, there is a template rule that also forbids drawing larger transistors.
  SW is the effective symbolic width of the polysilicon segment.
- **SSD** (Symbolic Shrinkage of the diffusion). It is a constant and negative length identical for all types of transistors. It allows to have a positive extension of the polysilicon segment over the diffusion.
- **SOD** (Symbolic Offset of the Diffusion). It is the offset between the center of the gate and the junction points with the drain and source diffusion. For LSXLIB, this offset is constant since the width of the polysilicon segment is constant.

Beware!

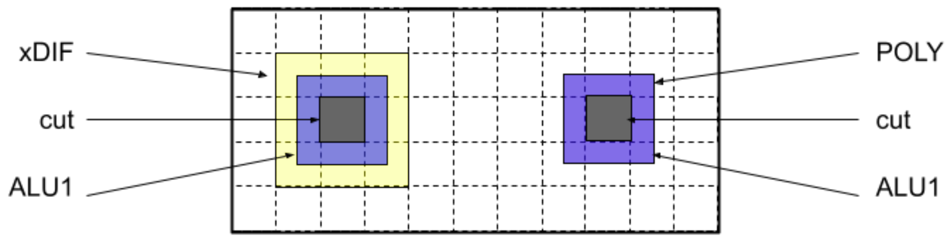
The channel of the transistor is defined by the intersection of the polysilicon and the diffusion. The width of the channel is therefore equal to the length of the polysilicon segment plus 2 times the narrowing (negative value) of the diffusion: **SL** + 2 * **SSD**.
The width of the channel grows with the length of the polysilicon segment!

Well segments

The well segments NWELL and PWELL define the place for the P-transistors (PTRANS) and N-transistors (NTRANS) respectively. The NTE and PTIE segments define the location of the bias contacts. These segments have the same geometry as the signal segments with the 3 sizes **SML**, **SDL** and **SMW**, however the position and size of these segments are imposed by the cell size rules. For more details, please read the "Template" section of this document.

Geometry of contact symbols

The contact symbols have a fixed square size, they are composed of a fixed size ALU1 pad (**SWA**), a fixed size cut pad (hole in the insulator between the levels) (**SWC**) and a fixed size pad of one of the xDIF, xTIE or POLY types (**SWO** with O for Other). The ALU1 and POLY / xDIF / xTIE pads do not have to be the same size.



Geometry of the Special Symbol

- Abutment Box — Cell border, where neighboring cells should abut. For more details, read the "Template" section of this document

Symbolic drawing rules

The segments and contacts are drawn on the lambda gate, but we have just seen that each has its own geometry. For each symbol, we give the minimum width and length characteristics and the length extension(s), as well as the minimum offset for the drain and source of the transistors. We also give the minimum distances between segments of the same and different types. The distances between signals are measured from edge to edge. In these rules, we only show the symbols needed to create the netlist of transistors: POLY, NDIF, PDIF, ALU1, NTRANS and PTRANS. For the others, their position is governed by the template rules that are presented next.

Minimum geometry and edge-to-edge distance of signal segments

The table on the left contains the minimum geometry of the symbols and the table on the right contains their minimum distances measured edge to edge. "none" means that there is no rule, "T" means that there is a template rule, thus it is respected by construction if the cell respects the template rules presented later.

Geometry of signal segments

Name	**SML**	**SDL**	**SMW**
POLY	1	0.5	1
NDIF	2	0.5	3
PDIF	2	0.5	3
ALU1	1	1	2

Distance of the signal segments

SMD	**POLY**	**NDIF**	**PDIF**	**N/A**
POLY	2,5	1	1	none
NDIF	1	3	**T**	none
PDIF	1	**T**	3	none
ALU1	none	none	none	3

Contact geometry

The table below contains only the geometry of the contacts. Indeed, the contacts used for the transistor netlists are of the types ALU1, POLY, NDIF and PDIF. The minimum edge to edge contact-contact or contact-segment distances are therefore the same as those given in the table "Signal segment distances".

Contact geometry

Name	SWA	SWC	SWO
CPOLY	2	1	2
CNDIF	2	1	3
CPDIF	2	1	3
CNTIE	2	1	3
CPTIE	2	1	3

Geometry of the transistor segments

The table below contains only the geometry of the transistors. Indeed, the transistors are made of POLY, NDIF and PDIF types. The minimum edge-to-edge distances between the transistors and the other symbols are therefore the same as those given in the table "Signal segment distances".

Geometry of the transistor segments

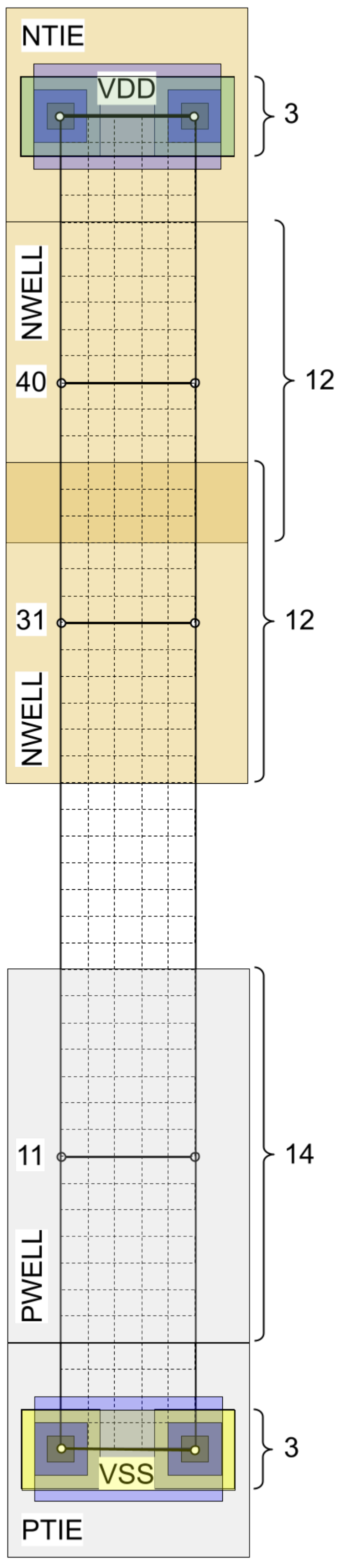
Name	SML	SDL	SMW	SSD	SOD
NTRANS	6	0	1	1.5	3
PTRANS	6	0	1	1.5	3

Typical patterns using segments and contacts for transistor netlists

In the drawing below are represented patterns of segments and contacts of POLY, ALU1, NDIF, PDIF, NTRANS, PTRANS, CNDIF, CPDIF and CPOLY. The patterns are placed on a lambda grid. The segments and contacts are at least drawing rules.

Cell template rules

The purpose of the template rules is to easily allow the abutment of cells to make complex blocks, they also integrate the constraints, imposed by the placement and routing tools, such as the nature and position of connectors. For LSXLIB, there are also rules to simplify the transformation of segment and contact symbols into real masks. The quantities in the rules are given in lambdas.



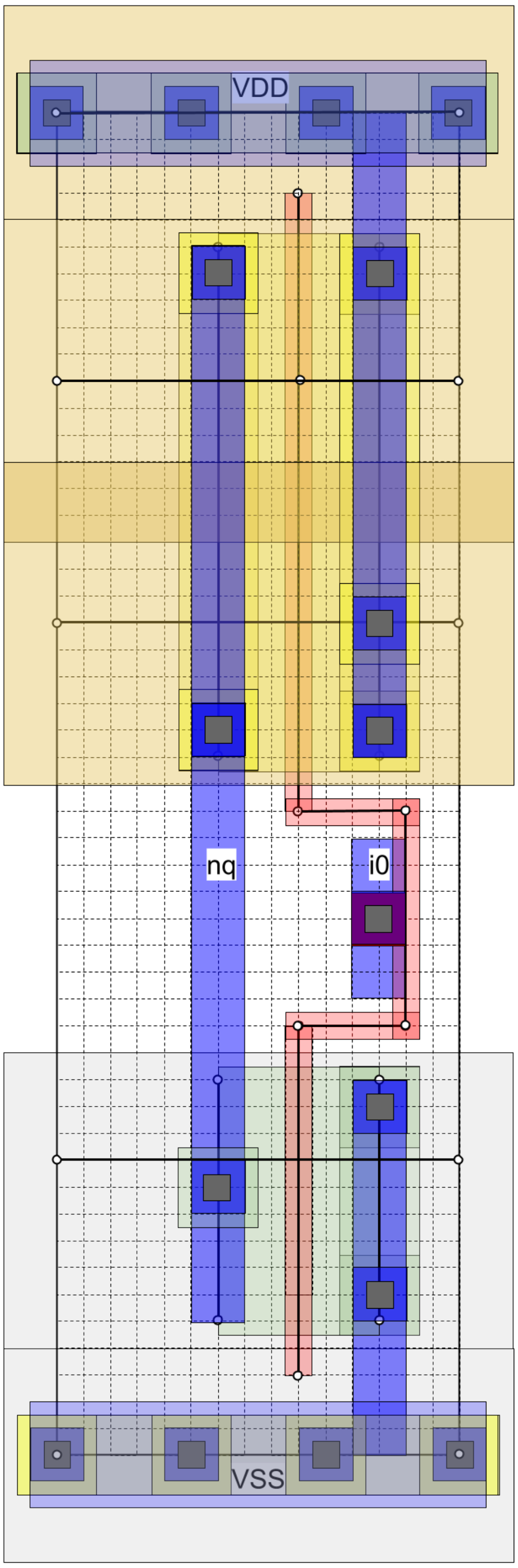
The drawing shows an empty cell, without transistors, with only the NWELL, PWELL, NTIE and PTIE segments, as well as the contacts touching the neighboring cells when they are abutted (CNTIE and CPTIE).

The grid is the lambda grid on which the end of all segments and the center of the contacts are placed.

- Height of a cell : 50 lambdas.
- Width of a cell : multiple of 5 lambdas.
- Coordinates of abutment box lower left corner are 0,0
- On the north face of the abutment box
    - CALU1 VDD segment of 4 lambdas width
    - NTIE segment of 3 lambdas wide. This segment is used for the polarization of the NWELL in which are the P transistors (PTRANS). The NWELL under the NTIE is automatically generated with NTIE.
    - CNTIE contacts placed every 5 lambdas
- Two 12 lambdas wide NWELL segments from west to east, at ordinate 31 and 40
    - The segments have an intersection of 3 lambdas. The goal is to obtain a segment of 21 lambdas wide and since the center of the segments must be on the lambda grid, 2 segments must be used
- PWELL segment from west to east of 14 lambdas wide at ordinate 11
- On the south face of the abutment box
    - CALU1 VSS segment of 4 lambdas width
    - PTIE segment of 3 lambdas wide with a PWELL segment. This segment is used to bias the PWELL in which the N transistors are (NTRANS).
    - PTIE contacts placed every 5 lambdas

The north and south abutting cells must be rotated about the axis so that the VDD and VSS connectors overlap.

The VDD and VSS connectors in CALUL1 must be covered with a routing metal to supply the cells. The ALU1 is only used for the local supply of the transistors, even if it is connected to the neighboring cells. This choice explains the reason why ALU1 is only 4 lambdas wide.



In the figure opposite, we have represented an inversor with a transistor P of channel width 20 lambdas and a transistor N of channel width 10 lambdas.

The transistors have been placed as far to the right as possible to illustrate the boundary rule that prohibits segment and contact symbols (other than CALU1 VDD, VSS and box segments) from approaching the boundary, defined by the abutment box, by less than the minimum half distance so as not to fear a minimum distance violation with respect to the symbols of neighboring cells

In the drawing, the polysilicon segments are placed to illustrate this border rule, in the LSXLIB cell, the inversor (inv_x2) is more "pretty".

- The NTRANS and PTRANS segments are always vertical
- The ends of the PTRANS are in the ordinate range between 25 and 37
- The ends of the NTRANS are in the ordinate range between 9 and 19
- The segments of NDIF, resp. PDIF, are included in the PWELL, resp. NWELL with a margin of 0.5
- Segments and contacts are placed at a distance from the abutment box depending on their type.
  The measurement is made from the edge of the segment.

ALU1	
POLY	1.5
NDIF	1.5
PDIF	1.5

Symbolic transformation to real masks

The symbolic to real mask transformation consists in transforming a symbolic cell into a real cell for a target technology. It consists in transforming the segment and contact symbols used by the symbolic cell drawing into a set of real rectangular masks with the right dimensions for a target technology. If the symbolic drawing rules and the template rules are well respected, then the real drawing obtained by the transformation operation respects the rules of the target technology.

To enable the symbolic transformation to real masks, the lambda value must be known. In general, the lambda value is a little less than the minimum channel length for the target technology. For example, for the sky130 technology, the drawn channel length is 150 nm and the lambda value is 140 nm (for example, the effective channel length for sky130 is 130 nm). We will see in the next section how to obtain the lambda value by knowing the symbolic and target rules. Here, we assume the lambda value in microns is known.

Principle of the transformation

- Each segment or contact symbol is individually transformed into a set of real masks. The coordinates of the center of these masks are equal to the symbolic coordinates multiplied by the lambda value.
  - For sky130, each symbol is transformed into 1 to 3 real masks

ALU segment	→	LI mask
POLY segment	→	POLY mask
Contact CPOLY	→	LI Mask
		LICON Mask
		POLY Mask

- For the segments
  - The width of each real mask RW is equal to the drawn symbolic width of the segment SW multiplied by the value of the lambda to which we add a delta (increase or decrease) RDW different for each mask. These RDW are calculated so that a minimal drawn symbol is transformed into a real mask of minimal width.
    - For each mask: = SW * lambda + RDW
  - The symbolic length extension SDL is replaced by a different real length extension RDL for each mask.
- For contacts
  - The actual size of each real mask is fixed
- For the abutment box
  - Its dimensions are simply multiplied by the lambda

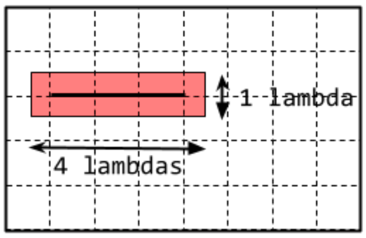
Application to sky130 technology

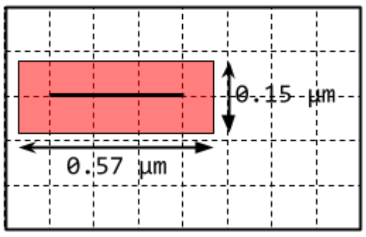
In this part, we give the transformation rules for all symbols used for LSXLIB to sky130 technology. First the segments, then the contacts.

In the tables, it may seem strange that the PDIF and NDIF segments only turn into TAP masks (active area) without the PSDM (P+IMPLANT) or NSDM (N+IMPLANT) masks. This is not a mistake, the PSDM and NSDM layout areas are added with the NWELL and PWELL symbols. Indeed, the symbolic template rules impose that the PDIF segments are included in the NWELL and that the NDIF segments are included in the PWELL, so it is the NWELL and PWELL segments that generate the PSDM and NSDM masks. In this way, it is possible to implement N+ or P+ all the active zones of a cell with very few masks

Symbols Segments

lambda grid	1
SL	3
SDL	0.5
SW	1





real grid	0.14 µm
RL	0.42 µm
RDL	0.075 µm
RDW	0.01 µm

Symbolic Name	Real Name	RDL	RDW
ALU1	LI	0.165	0.05
POLY	POLY	0.075	0.01
NDIF	TAP	0.07	0
PDIF	TAP	0.07	0
NTRANS	POLY	-0.075	0.01
	TAP	-0.21	0.56
PTRANS	POLY	-0.075	0.01
	TAP	-0.21	0.56
NWELL	NWELL	0.32	0.28
	PSDM	0.085	0.17
PWELL	NSDM	0.085	0.17
NTIE	NWELL	0.32	0.51
	NSDM	0.27	0.12
	TAP	0.145	-0.13
PTIE	PSDM	0.27	0.12
	TAP	0.145	-0.13

Symbols Contacts

Symbolic Name	Realname	RW
CPOLY	LI	0.33
	LICON	0.17
	POLY	0.29
CNDIF	LI	0.33
	LICON	0.17
	TAP	0.42
CPDIF	LI	0.33
	LICON	0.17
	TAP	0.42
CNTIE	LI	0.33
	LICON	0.17
	TAP	0.29
CPTIE	LI	0.33
	LICON	0.17
	TAP	0.29

Lambda calculation

Principle

To calculate the value of the lambda, we must study the transformation of the patterns of symbols of minimal size into real masks of minimal size. A pattern is for example, two segments of minimal width placed at the minimal distance. Each pattern imposes a value for the lambda and it is then sufficient to keep the minimum value found. The found values must be aligned to an even number of minimum physical grid steps which is equal for sky130 to 0.005 µm. This must be an even number of physical grid steps because the masks are laid on the lambda grid and they spill over symmetrically on both sides.

For example: POLY segment - POLY segment

- SMW = 1 lambda and SMD = 2.5 lambda
- RMW = 0.15 µm and RMD = 0.21 µm
- (SMW + SMD) * lambda ≥ RMW + RMD
  ⇒ lambda ≥ (RMW + RMD) / (SMW + SMD)
  ⇒ lambda ≥ 0.36 / 3.5 = $\lceil 0.1028 \rceil$ = 0.11 µm

Patterns used for lambda calculation

- NWELL segment - parallel NWELL segment
- NWELL segment - perpendicular NWELL segment
- NWELL segment - NWELL segment aligned
- ALU1 segment - ALU1 parallel segment
- ALU1 segment - ALU1 segment perpendicular
- ALU1 segment - ALU1 segment aligned
- For *MASK* in { NDIF, PDIF, NTIE, PTIE, POLY }
    - *MASK* segment - parallel *MASK* segment
    - *MASK* segment - *MASK* segment perpendicular
    - *MASK* segment - *MASK* segment aligned
    - *MASK* segment - *CMASK* contact using the *MASK* level
    - ALU1 segment - *CMASK* contact using ALU1 level
    - *CMASK* contact - *CMASK* contact using the *MASK* level
    - *CMASK* contact - *CMASK* contact using ALU1 level
- For *type* in { N, P }
    - *typeTRANS* segment - *typeTRANS* segment parallel common drains
    - *typeTRANS* segment - *typeTRANS* segment parallel separate drains
    - *typeTRANS* segment - POLY parallel segment
    - *typeTRANS* segment - perpendicular POLY segment
    - *typeTRANS* segment - CPOLY parallel segment
    - *typeTRANS* segment - CPOLY perpendicular segment
    - *typeTRANS* segment - *typeDIF* parallel segment
    - *typeTRANS* segment - *typeDIF* perpendicular segment
    - *typeTRANS* segment - *CtypeDIF* parallel segment
    - *typeTRANS* segment - *CtypeDIF* perpendicular segment

Rules for symbolic transformation to real masks

For symbolic transformations to real masks, once the value of the lambda is known, the values of the variables RDL and RDW for the masks produced by the symbols and the sizes RW for the masks produced by the contacts must be calculated.

The lambda calculation imposes minimum values for RDL and RDW, but it is possible to increase them as long as it does not impact the lambda value.