Generic Standard Cell Library

Revision 1.0

Date:November 18, 2002
Author: Gilles Lamant
Contributors: Brian Campbell, David Neiman, Jessica Felippe, Justin Nielsen
Project ID: GSCLib

Revision Chart:

Revision	Date	Name	Comments
0.0	06/14/02	Gilles S.C. Lamant	Document set-up.
0.2	06/17/02	Gilles S.C. Lamant	Start being serious about it...
0.3	06/20/02	Gilles S.C. Lamant	Put in a lot of the meat...
0.35	06/21/02	Gilles S.C. Lamant	list of cells defined, and more...
0.4	06/28/02	Gilles S.C. Lamant	First release to broader audience to solicite additional input and comments
0.5	08/23/2002	Gilles S.C. Lamant	Added data from Brian for size selection. Several updates. Characterization flow is redefined to use SIMPLEX signalstorm. Added abstract generation flow.
0.8		Gilles S.C. Lamant	Doc version now in sync with library version.
0.9		Gilles S.C. Lamant	Recommended user set-up
1.0		Gilles S.C. Lamant	Added info for correct scan cell generation in .lib/.tlf

Reviewers:

Date	Name	Comments
	John Goelz PDK factory	
	Fred Sendig Fellow, R&D, IC	
	John Gianni Flow Engineering	
	Johnatan David	
	Jon Sanders	
	Vasily Syngaevsky Motorola, Russia	
	Paul Koch Texas Instruments	
	Jens Werner Spectrum Services	
	???? MIET	

1. TABLE OF CONTENTS

2. INTRODUCTION

This document describes the content planned for creating a Generic Standard Cell Library, compatible with the Cadence PDK factory, GPDK.

2.1 Problem Statement

Today, there is not a single standard cell library available in Cadence that can be freely used for educational purpose, test or demo purpose. For specific purposes, like DAC, it is possible to obtain temporary or limited authorization from library vendors to use one of their libraries. However, this usage is always restricted. In addition, existing US regulations prevent the use of technologies smaller than 0.2 µm in several countries where Cadence has educational programs.

Finally, the libraries that are available are usually incomplete, as they are targeted for digital applications only. With the increasing importance of mix-signal designs, it is a requirement for us to not only have the usual abstracted information for a digital cell library (abstract and TLF), but also the fully detailed layout and transistor level schematics. Of course, to be of use, these detailed representations need to be using an available library of primitive elements (usually called a PDK).

We have come to the conclusion that using a generic PDK, such as the GPDK produced by the Cadence PDK factory was the only solution that was allowing to freely distribute around the world this work, and make it usable to all. However, no industrial partners have yet shown interest in developing a digital library for that process, since it does not have any direct financial return for them.

Our plan is to use internal Cadence resources, interns from the Cadence University Programs, as well as guidance and help from industrial partners (such as Motorola or Texas Instruments), who are interested in seeing us realize this project.

2.2 Identification

This library is not in theory dependant on a release of any single Cadence products, since by definition, it is intended to be used across a large number of Cadence streams. At this time, we are planning to work with the following products for developing and validating the library:

- *IC 5.0 (required for support of inherited connections in VerilogA)*
- *AMS 3.5 (if in time AMS 4.0)*
- *SP&R 5.0, QSR1*
- *DSM 5.4, QSR1*
- *CADMOS 4.0, QSR2*
- *SignalStorm 1.3.0*

2.3 Scope

Because the realization of this project is based upon the voluntary contribution of several Cadence folks, the use of intern students, as well as the good will of industry partners like Motorola and Texas Instruments, the scope of this project is limited in depth (the level of detail, or number of tools supported) as well as in breath (the total number of cells in the library).

In this first version of the library, we will not cover power consumption data, or Signal

Integrity/noise data required to use the tools from the cadMos family. This will be the subject of a later project.

We do not plan to incorporate, in this library, larger IP blocks, like a processor or memory. These could be add-on, in different IP libraries, provided they are compatible with the GPDK technology.

Although important to us, we will not work in this project to improve the GPDK itself. However, we will work with the PDK factory, to insure that all problems are reported, and fixed in a timely and appropriate fashion.

Finally, we will use currently existing Cadence tools. This project does not intend to be redefining new tools, or new flows.

3. DESCRIPTION

3.1 Overall description

3.1.1 Cells in the library

The cell naming convention (for cell name and pins) is directly derived from the Artisan conventions. This will insure that a design created using this library should be easily migratable to a real Artisan library if needed.

We built the list for the basic cells required in the library, based upon the paper from J.L. Noulet and A. Ferreira-Noullet “Do we need so many cells for digital ASIC synthesis”, as well as statistical survey of designs available inhouse. Clock buffers have been added with the idea that we also need to support the clock tree synthesis programs.

Cell type (boolean function)	Cell Names (using Artisan naming conventions).
Inverters	INVX1, INVX2, INVX4, INVX8
And	AND2X1
Nand	NAND2X1,NAND2X2, NAND3X1,NAND4X1
Nor	NOR2X1, NOR3X1, NOR4X1
Or	OR4X1, OR2X1
2 level logic	AOI21X1, AOI22X1 OAI21X1, OAI22X1, OAI33X1
Xor	XOR2X1
Multiplexer	MX2X1
Flip-Flops	DFFX1, DFFSRX1, SDFFSRX1
Clock Buffers	CLKBUFX1, CLKBUFX3, CLKBUFX2
Tri-state buffers (GSCLib v 2.0)	TBUFX1, TBUFX2, TBUFX4, TBUFX8
Adders (GSCLib v2.0)	ADDHX1, ADDFX1
Buffers	BUFX1, BUFX3
Filler cell	FILL1, FILL2, FILL4

We therefore have a basic library of 32 cells (including the filler cells) in the target first release of the library (GSCLib v1.0).

3.1.2 Power supply strategy

We are planning to support the inherited connection features of dfII to support the complex multiple power supply distribution schemes found in large mixed-signal designs today.

The power terminal will be named "**POWR**" and will default to the "**vdd!**" global net.

The ground terminal will be named "**GRND**" and will default to the "**gnd!**" global net.

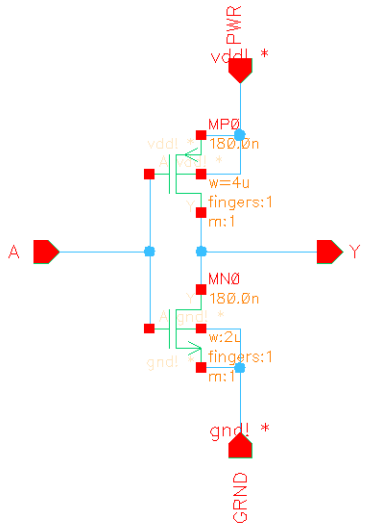
In addition, experience has showed that in addition to the ability to use pure netSet properties on instances to specify an inherited connection overwrite, in many cases, designers like to have physical pins to do this job. As a result, the library will provide, for each symbol, a duplicated set: one symbol without the power supply pins (**symbol**), and one with the power supply pins (**symbolPT**).

3.2 What representations are needed

3.2.1 Schematic

For each cell in the library, a transistor level schematic will be produced, using devices from the GPDK library.

All power supplies in the schematic will be explicitly connected, and will end-up in a terminal with an attached net expression, resulting in the definition of an inherited connection. As a result, a schematic for a cell in the GSCLib will always have the maximum number of terminals possible for that cell.



The net expressions used are:

- For the power pin: **[@POWR:%:vdd!]**
- For the ground pin: **[@GRND:%:gnd!]**

3.2.1.1 Device sizes.

Using the GPDK nominal models for the devices, we have measured the delays for a inverter driven by a pulse with rise & fall times of 50ps, while loaded with a load equivalent to that of 4 inverters (40fF).

In addition, we have set the requirement that the output voltages reaches 10% (or 90%) of the voltage supply, in less than 250ps with the specified load condition.

Based upon the data gathered, we have selected the following sizes for the basic device size (minimum process length is used: 0.18 µm):

NMOS transistor : 0.85 µm

PMOS transistor : 2.02 µm

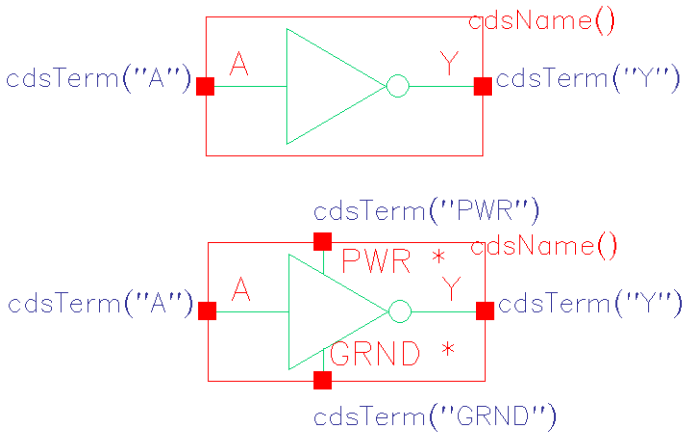
These values provide a nearly symetric rise/fall delay, as well as a good overall speed performance (72.7 ps + 72.8 ps) .This correspond to the X1 notation in the standard cell naming convention. Larger drive strength X2, X4 are obtained by increasing the output transistor size accordingly.

3.2.2 Symbol (with and without explicit power supply pins)

We propose to build two symbols for each cell. One will not represent the power/ground pins, while the other will do. This enables designers to physically overwrite inherited connections, instead of using netSet expressions on the instance. We recommend that no netSet expressions be set on the instance with the physical terminals, as this makes for a very confusing precendence order...

3.2.2.1 Symbol examples

- ***Inverter (INVX1)***



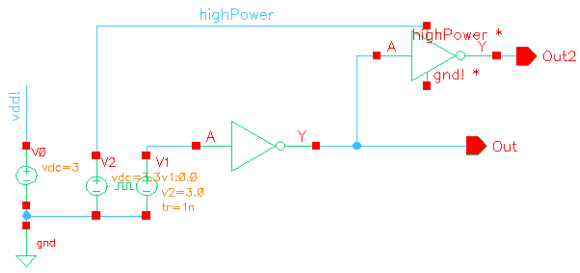
The net expressions used for the **`symbolPT`** view are the same as the ones used in the schematic:

- For the power pin: **`[@POWR:%:vdd!]`**
- For the ground pin: **`[@GRND:%:gnd!]`**

3.2.2.2 Schematic & netlist (spectre) example.

The following schematic shows an example of the same inverter instantiated once with its regular symbol, and once with its symbol with the explicit terminals for the power pins. In addition, note that one of the explicit pin from the symbol has been left un-connected, and therefore defaulted **`gnd!`**, while the other was physically connected (with a named wire) to a

different power supply.



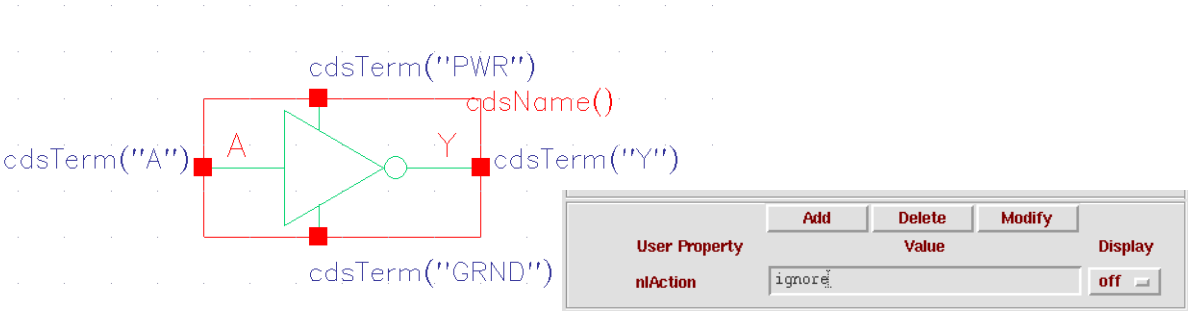
The resulting netlist, produced using Analog Artist Spectre direct netlister, is shown below:

```
// Library name: GSCLib
// Cell name: INVX1
// View name: schematic
subckt INVX1 A Y inh_GRND inh_POWR
MP0 (Y A inh_POWR inh_POWR) pmos1 w=4u l=180.0n as=2.4e-12 ad=2.4e-12 \
ps=5.2u pd=5.2u m=(1)*(1)
MN0 (Y A inh_GRND inh_GRND) nmos1 w=2u l=180.0n as=1.2e-12 ad=1.2e-12 \
ps=3.2u pd=3.2u m=(1)*(1)
ends INVX1
// End of subcircuit definition.

// Library name: test
// Cell name: INVX1Test
// View name: schematic
I3 (Out Out2 0 highPower) INVX1
V1 (net2 0) vsource type=pulse val0=0.0 val1=3.0 period=25n rise=1n \
fall=1n width=10n
V2 (highPower 0) vsource dc=3.3 type=dc
V0 (vdd! 0) vsource dc=3 type=dc
I0 (net2 Out 0 vdd!) INVX1
```

3.2.3 Verilog

The verilog view is a “black box” type view. It is used to create netlists for the pure digital verilog XL applications. This does not support the use of inherited connection. As a result, the pins corresponding to the power supplies in the verilog representation will be physically represented, as they are required for all switched master. However, they will be dropped when netlisting to Verilog-XL, using the **`nlAction = ignore`** attribute. In addition, no net expression should be attached to these pins.

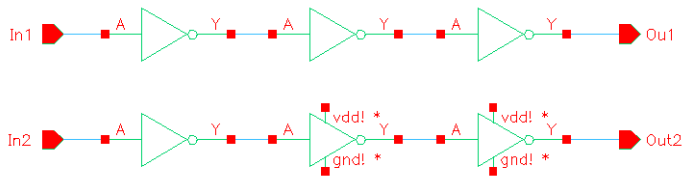


A netlist created with such a view will only contain instantiations of the verilog gates. Such

netlist can be used for creation of blocks with Silicon Ensemble.

3.2.3.1 Example: from Composer to Verilog-XL (using verilog Integ)

The following composer schematic:



will result in the verilog netlisting shown below:

```
// Library - test, Cell - INVX1Test2, View - schematic
// LAST TIME SAVED: Jun 17 17:10:16 2002
// NETLIST TIME: Jun 17 17:17:21 2002
`timescale 1ns / 1ns

module INVX1Test2 ( Ou1, Out2, In1, In2 );
       output  Ou1, Out2;
       input  In1, In2;

       specify
              specparam CDS_LIBNAME  = "test";
              specparam CDS_CELLNAME = "INVX1Test2";
              specparam CDS_VIEWNAME = "schematic";
       endspecify

       INVX1  I5 ( .Y(Out2), .A(net7));
       INVX1  I4 ( .Y(net7), .A(net9));
       INVX1  I3 ( .Y(net9), .A(In2));
       INVX1  I2 ( .Y(Ou1), .A(net13));
       INVX1  I1 ( .Y(net13), .A(net15));
       INVX1  I0 ( .Y(net15), .A(In1));

endmodule
```

For simulation, an additional directory containing the behavioral description of each gate will be required.

The verilog behavioral models for each individual gates will be included in a single file (GSCLib.v) representing the complete library. Cells will be described in a style similar to that used by Artisan in their libraries, and will include a default unit delay.

```
// Library: GSCLib, cell INVX1
`timescale 1ns / 1ps
`celldefine
module INVX1 (Y A);
       output Y;
       input A;

       not I0(Y, A);
```

```
        specify
               // parameters for the intrisic delay(
               specparam tplh$A$Y = 1.0 ;
               specparam tphl$Y$A = 1.0 ;
               // path delay through the cell
               (A *> Y) = (tplh$A$Y,tphl$A$Y) ;
        endspecify
endmodule
`endcelldefine
```

- ***Specify section: path delays***

The verilog model specify section will be created to match the timing arcs and checks from the TLF, in order to allow for smooth SDF back-annotation. Special care needs to be taken to have the same definition of the conditional paths described in the Verilog model, and in the TLF. This is normally automatically taken care of by using the Simplex `alf2ver` converter.

Initial default delay values will all be initialized to a unit delay.

- ***Specify section: timing checks.***

We plan to support the following timing checks:

MUX : $setuphold, $width

DFF: $recovery, $width, $setuphold

The verilog file, including the specify section, will be generated automatically using the alf2ver utility from the signalstorm tool suite. It takes as its input the alf file created during the cell characterization process. Using this method, we can guaranty that by construction, our verilog and the TLF (or .lib) files are compatible.

3.2.3.2 Verilog In

The methodology proposed here support the use of Verilog-In, as a way to create a schematic (Composer) database for a block associated with a single power supply pair. This is because Verilog In is usually used after a synthesis phase, where the created verilog netlist does not contain any description of the power supply connections. Here, the symbol without explicit power pins will be used, and the user can overwrite the inherited connections using a netSet expression placed on the block symbol.

The following verilog netlist

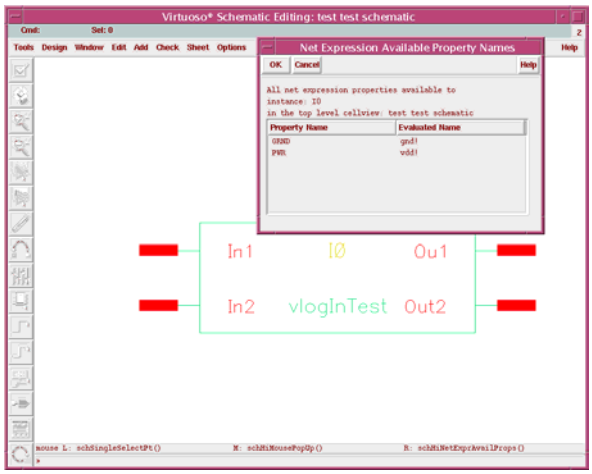
```
module vlogInTest ( Ou1, Out2, In1, In2 );

       output  Ou1, Out2;
       input  In1, In2;

INVX1  I5 ( .Y(Out2), .A(net7));
INVX1  I4 ( .Y(net7), .A(net9));
INVX1  I3 ( .Y(net9), .A(In2));
INVX1  I2 ( .Y(Ou1), .A(net13))
INVX1  I1 ( .Y(net13), .A(net15));
INVX1  I0 ( .Y(net15), .A(In1));

endmodule
```

Will result after the **File -> Import -> Verilog** command is run in a schematic and a symbol. After the user creates an instantiation of the block, he/she can use the **Edit -> Net Expression -> Available properties** to review and set overwrite for the power supplies for the block.



3.2.4 VHDL

VHDL representation will be generated using the alf2vhdl utility from the signalstorm tool suite. We will however not verify the data generated.

3.2.5 VerilogA (GSCLib v2.x)

In IC 5.0, Verilog-A has been enhanced to support inherited connections. This make it the ideal language for running simulations with small digital content (large mixed-mode simulation are to be targeted to the AMS (NCSim) simulator). In such simulations, the digital content is actually handled by the analog simulator.

Several level of accuracy of modeling in verilog-A can be used. In the first release of the library, we propose to deliver 2. A zero order model (veriloga) that simply represents the behavioral description of the cell, and a more detailed model (first order), that would have 3 stages: an input stage, which models the input capacitance of each pin, a behavioral stage, including the propagation delay, and an output stage, modeling the drive strength of the gate. In a later version of the library, and if the need is confirmed, we might develop a third order model including the coupling thru the Miller capacitance of the output to the input.

In all models, the output voltages will be derived from the actual power supplies connected to the cell.

Important: Do not use td, tf, tr as verilog-a parameter names. These are recognized by the digital Verilog Integration, and cause trouble in the netlisting process. For example, when a default value is specified withing the verilog-a model, it does not have to be on the instance. However, when this instance get netlisted through verilogInteg, a line like the following (illegal) is ssued, since the parameters values for (tr,tf) are not defined on the the instance.

```
INVX1 #(, ) I23 (.Y(net123), .A(net78));
```

Important: The verilogA constructs need to be limited to the constructs also supported by the AMS (Verilog-AMS) simulator. This enable the use of an analog model for the digital cell in the AMS environment if required by the designer.

3.2.5.1 Zero order model (`veriloga`)

This model is intended for transient simulation only. It is a simple behavioral description of the gate, and has its intrinsic delay(s) adjusted based upon the characterization data. The designer should be aware that:

- *The output is an "ideal" voltage source, hence it is not limited in its ability to drive other elements as it will give infinite current if needed!*
- *The input does not include the effective load that the real device will put on the circuit.*

```
// VerilogA for GSCLib, INVX1, veriloga

`include "constants.h"
`include "discipline.h"

module INVX1(Y, A, GRND, PWR);
       output Y;
       electrical Y;
       input A;
       electrical A;
       input (* integer inh_conn_prop_name="GRND";
                integer inh_conn_def_value="\\gnd! "; *) GRND;
       electrical GRND;
       input (* integer inh_conn_prop_name="PWR" ;
                integer inh_conn_def_value="\\vdd! "; *) PWR;
       electrical PWR;

       parameter real   tdelay =1.0n from [0:inf);
       parameter real   tfall = 1.0n from [0:inf);
       parameter real   trise = 1.0n from [0:inf);

       real vout;
       real vth ;
       integer outState ;

       analog
       begin
              vth = (V(PWR) - V(GRND)) / 2.0 ;
              outState = V(A) >= vth ? 0:1 ;
              vout = outState *V(PWR) ;
              V(Y) <+ transition(vout, tdelay, trise, tfall) ;
       end
endmodule
```

3.2.5.2 First order model (`veriloga1`)

Although the user will always have the ability to overwrite them, we will use the results of the timing characterization (TLF) to set the default values for the parameters of these models, resulting in the best possible fit of the characteristics.

3.2.5.3 Second order model (`veriloga2`)

3.2.6 Verilog-AMS. (GSCLib v 2.x)

3.2.6.1 Element library

It is important here to note that the AMS library itself does not intend to represent the elements with an analog behavioral. For this purpose, the designer should choose the verilogA representation (which is compatible with the AMS simulator). The idea is to have a verilog (digital) representation that includes inherited connection constructs (pure verilog-D does not support such constructs). This in turn, will enable the designer to switch to transistor level below the logical level, and keep the right power supply connections.

3.2.6.2 Connection element library (interface elements).

3.2.6.3 Installation

AMS designer is based upon the NCSim technology, and requires the library to be compiled in a version dependant .pak file. This file is created at the root level of the 5.x library structure, and therefore can only be created by someone with write priviledge for the GSCLib. (It is possible to use the ASSIGN tmp statment in the cds.lib, but we will not recommend this methodology here). A script will be provided to “install” the GSCLib. It will invoke NCSim and will compile the GSCLib verilogAMS views into the appropriate .pak file. It will use the first available version of the NCSim in the binary search path, at the time of installation.

If the software is updated to a new release, it is the responsibility of the person in charge of the libraries to “recompile” the GSCLib AMS views, to create a new .pak file in sync with the software version. Note that several .pak file can coexist inside a single 5.x library hierarchy.

3.2.7 Layout

3.2.7.1 Layout drawing guidelines: Available layers.

Metal2 and above is **not** permitted inside the standard cell layout.

3.2.7.2 Layout drawing guidelines: Preferred routing directions.

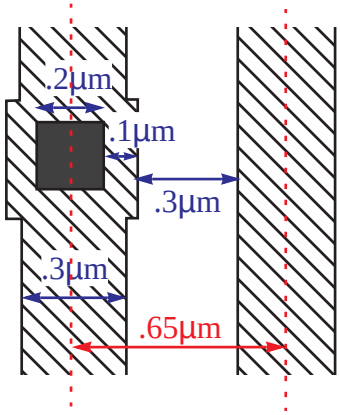
By convention, Metal1, Metal3 and Metal5 will be chosen to be of preferred horizontal routing direction, while Metal2, Metal4 and Metal6 will be vertical.

3.2.7.3 Layout drawing guidelines: Routing Pitch

As this standard cell library is targeted to be used with Silicon Ensemble, a gridded router, it is important to achieve optimum performance that for each signal getting in or out of the cell, connection points (pin) exists on the routing grid. Silicon Ensemble can route to off-grid pins, but this result in longer run times.

The routing grid definition will be based on the via-to-line spacing. This is the most commonly used grid definition in the industry. In the case of the GPDK process, all routing layers have

similar width and spacing rules. This will result in a simpler grid definition for our purpose.



o Minimum and maximum width of CONT 0.2µm
o Minimum METAL1 width 0.3µm
o Minimum METAL1 space 0.3µm
o Minimum METAL1 enclosure of CONT 0.1µm

=> theoretical pitch = 0.15 + 0.3 + 0.1 + 0.1 = 0.65µm

The resulting theoretical pitch is not divided in half without introducing an additional decimal place (thousands of a micron). This would cause us trouble as we need to place pins at 1/2 the routing pitch inside the individual cells (Note: the database and applications would support it, it is just very inconvenient for a layout designer to work with a snap grid set to 0.001 micron). We will therefore increase it to the next unit.

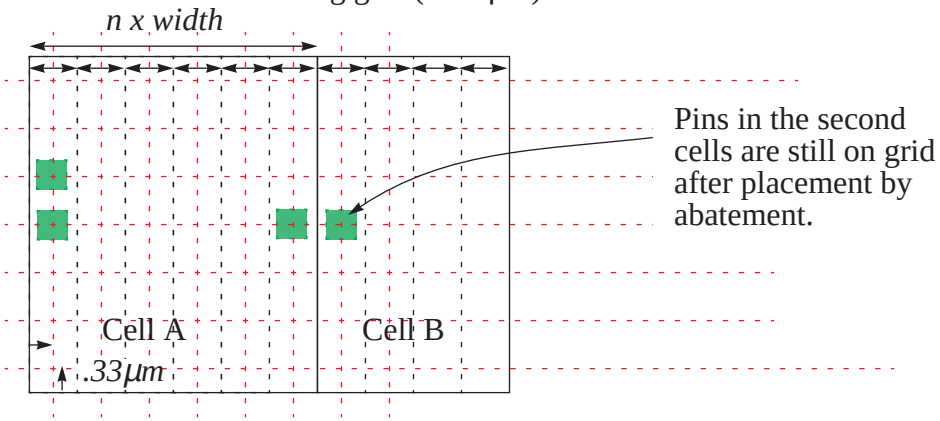
The routing pitch, for all layers, and all directions will be set to **0.66 µm**.

3.2.7.4 Layout drawing guidelines: Placement Grid (site width)

In Silicon Ensemble, standard cells are placed abutting each others within rows. In order to maintain the pins on grid when individual standard cells are abutted to each other, it is necessary that the width of the standard cell be set as a multiple of the vertical routing pitch.

The placement grid is set to 0.66 µm for our library. This will also be used as the site width, A site is the basic element used to build a row for Silicon Ensemble.

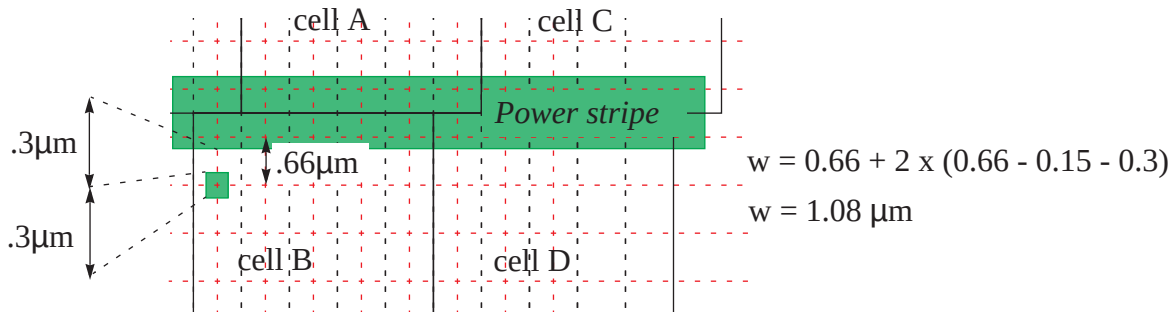
Within a specific cell, and to avoid having elements drawn directly on the prBoundary shape, we will enforce a 1/2 offset of the routing grid (0.33 µm).



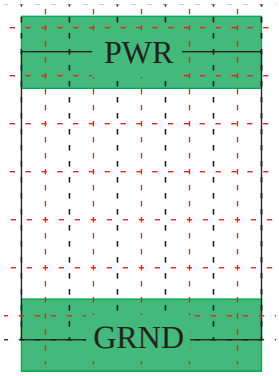
3.2.7.5 Layout drawing guidelines: Power pin width

We will assume here that rows for this process will be created abutting with each other, and flipped every other rows. This means that the power stripe, going between the cells, can be shared, and that the width of the line can be spread half on each cell. We will also try to keep

the second horizontal routing grid available, while creating the widest possible power wire.



This calculation shows that a total width of 1.08µm can be obtained. Such width is actually shared between the two abutting cells. However, because on the first row and last row in a block no cell are abutted to complete the power rail, we will be drawing the power strip, extending past the cell boundary in the vertical direction, to cover the full defined width. On the left and right sides, the power pins will be flushed to the side of cell boundary.



3.2.7.6 Layout drawing guidelines: Placement Grid (site height)

In process with several metal available for routing, as it is here the case with the GPDK, rows are created abutting with each other in the vertical direction as well. This means that, in order to obtain on-grid pins on the horizontal routing grid, the height of the cells will also need to be a multiple of routing grid.

However, additional considerations need to be reviewed in order to decide the exact multiple, as well as review of a few layout, to study the resulting porosity versus the desired horizontal porosity on Metal1. Based upon existing libraries of similar process (0.18 µm) we are expecting to use a height of about 7.92µm. This leaves 10 horizontal routing tracks potentially available through the cell.

This number might be reviewed after we have completed a few cell layout.

3.2.7.7 Layout drawing guidelines: Minimum spacing from boundary

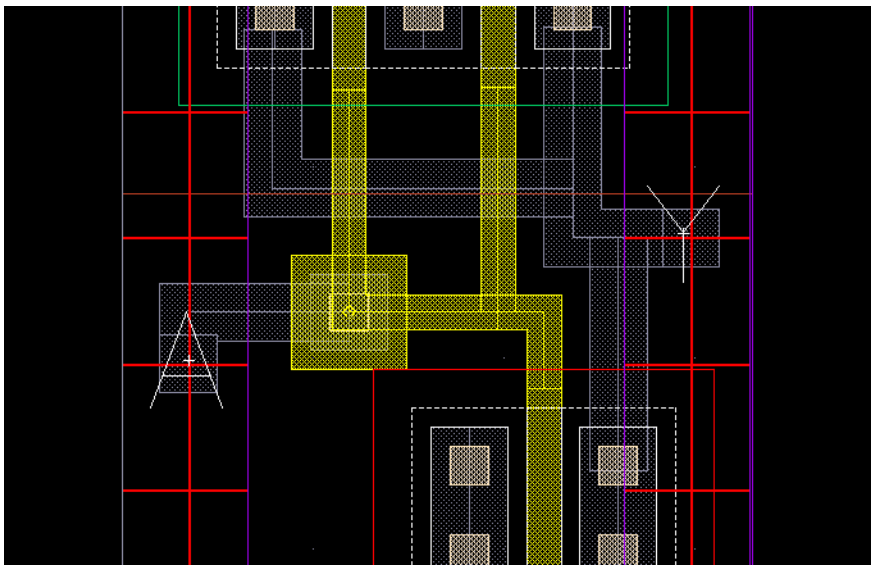
As it was already mentioned several times, during the placement phase, the individual cells will be tiled, abutting each other in all four direction. The tiling is based upon the cell boundary, which is represented in the layout cellview with the **`prBoundary/drawing`** layer. In order

for the layout to be DRC clean after the tiling, it is necessary to respect 1/2 minimum spacing rule for all rule, between the **`prBoundary`** and any other layers.

We might write an additional set of Diva rule to assist the designer in enforcing this rule while designing the cell library layout.

3.2.7.8 Layout drawing guidelines: on-grid pins.

At the minimum it is required that there be at least one access point to a signal pin on grid, on Metal1 (note: pins are to be drawn with Metal1/drawing lpp).



In this picture, the red lines represent the routing grid. The pins “A” and “Y” are located at the intersection of the routing tracks and are therefore good targets for the routers.

Ideally, the layout should provide for 2 on-grid access point for each signal.

3.2.7.9 Layout drawing guidelines: purpose for pins.

Pin in the layout are to be drawn as geometric shapes (no symbolic pins), using the **`drawing`** purpose.

3.2.7.10 Layout drawing guidelines: label for pins.

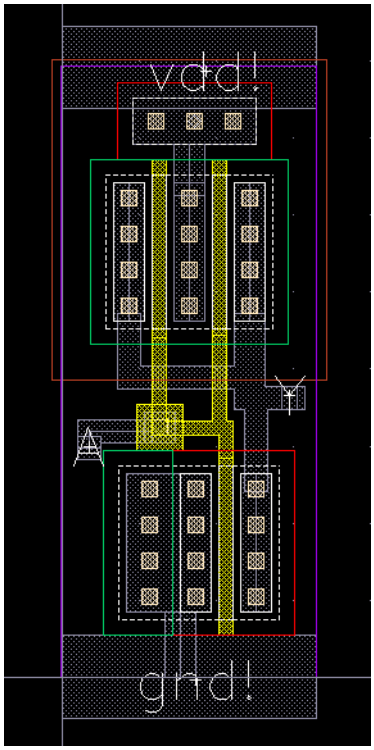
Pin must be labeled using an associated text display (created automatically if using VXL). The label must be on the **`text/drawing`** LPP. The origin of the label **must** be within the shape representing the pin.

3.2.7.11 Layout drawing guidelines: pin accessability from Metal 2

In addition to being on grid, if a pin is totally surrounded by Metal 1, the layout designer must be carefull to insure that there is room to drop a Metal1 -> Metal2 via on top of the pin without creating DRC violations, in order to enable the router to access the pin.

In general, it is recommended that the layout designer verifies that MOST pins in the cell are accessible both directly from Metal1 and from Metal2. The cell layout should be such that no NOTCH error be generated in the final layout, when such a via is placed ontop of a pin.

3.2.7.12 Layout drawing guidelines: Latch up protection (well contacts)



The GPDK rule manual specifies that the maximum spacing between well taps is 10 µm. Since there is no way to predict how cells are going to be combined in a row, we will require that each individual cell contained its own well contacts. In the case the cell width is larger than 10 µm, the well contacts needs to be distributed such that abutting combination of such cells results in no latch-up problem. This means that the well contact will not be put in the middle, but that rather well contacts will be placed on the sides of the cells.

No well tap should be further than **5 µm** away from the boundary of a cell.

For example a 12 µm wide cell could be covered by a contact placed in the center. However, when two such cells are placed next to each other, this would result in a spacing of 12 µm between the contacts.

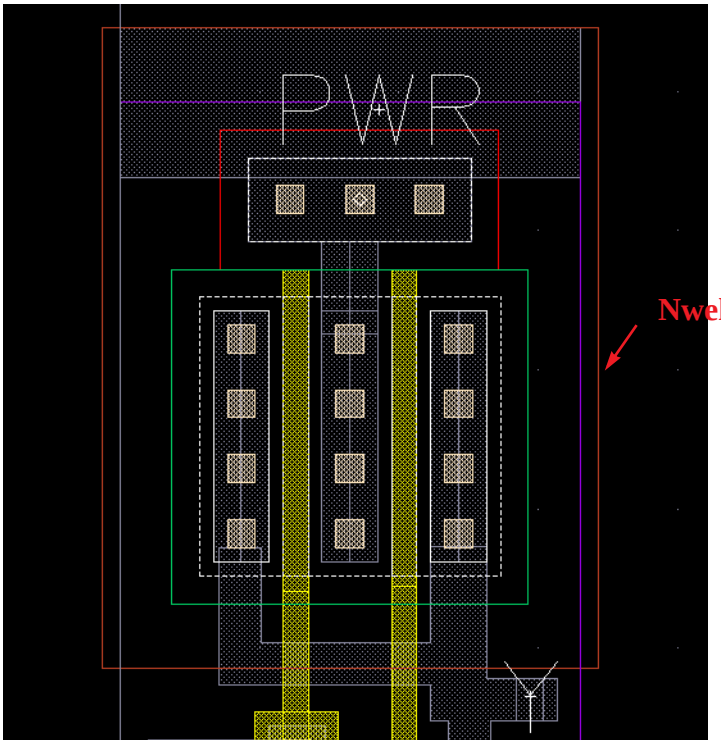
The example shows a possible legal configuration for an inverter.

Note also that the ties for the PWell around the NMOS transistor have been butted to the source of the transistor, allowing some space saving.

3.2.7.13 Layout drawing guidelines: Well continuity

In order to be design clean, including when located on a last row, or as the last cell of the row, it is important that the NWell shape be drawn correctly. Continuity of the Nwell within the rows is insured by drawing **Nwell to go at minimum, until the left and right side of the prBoundary**. In addition, because the cell can be the last of a row, or on a last row, it also must preserve the minimum overalp of N/P Imp (0.5 µm). It is therefore allowed to drawn the NWell shape extending outside of the prBoundary shape. The example below shows the NWell shape

for the inverter cell:

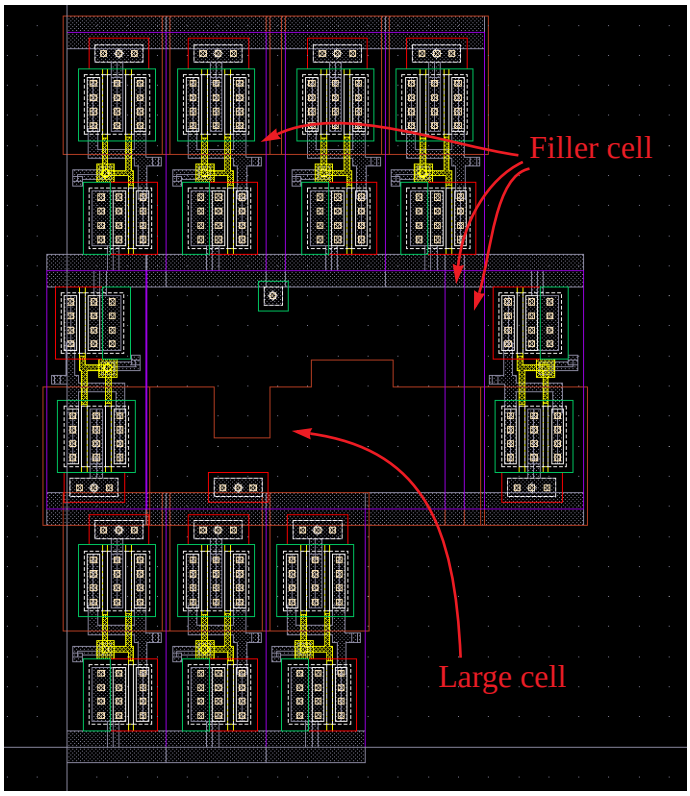


If placed in butting, flipped rows, the resulting NWell areas, with the help of a filler cell, are continuous (for large empy areas, filler cells with a well tie are needed).

By convention, the Nwell shape will cover in the vertical direction the full power rail, but never extend any further.

It will start from coordinate 3.66µm. This correspond to the area of the Nwell required to cover in the configuration of the minimum inverter, the width of 2.02 µm. Within the cell, it is possible to extend or retract the Nwell shape, but these values need to be maintained at the edges of the prBoundary. The following pictures shows a combination of the inverter cell, a large standard cell with a varying NWell shape, and some filler cells (FILL1) used to insure the

continuation of the well. Such a placement can be obtained with Silicon Ensemble QPlace.



3.2.7.14 Layout drawing guidelines: Antenna protection

We do not plan in this release to include diodes for antenna protection inside the cell layout. The cell however will have to pass the Diva antenna check provided with the GPDK to insure that the cell itself does not contain an antenna violation (Poly and Metal1).

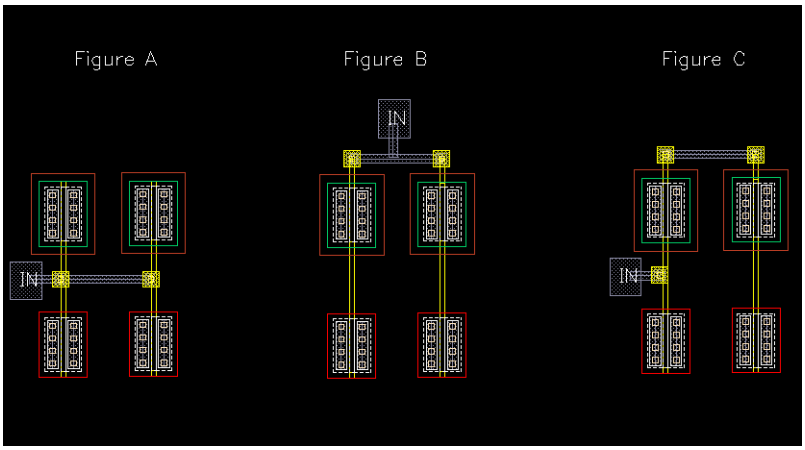
3.2.7.15 Layout drawing guidelines: backgate contacts

Although not currently available from the GPDK PCells, it is highly desirable that the layout be drawn using butting back-gate contacts, rather than standalone ones. This makes for a more real/professional library. We will work in the future with the PDK factory to see that this feature is added to the standard MOS PCells. The price for this is that for this iteration of the layout library, some PCells will have to be flattened, as butting the backgate contact will require modifying the N/PImp shape currently drawn by the PCell to allow DRC clean design.

3.2.7.16 Layout drawing guidelines: Poly gates used as conductor.

In as much as possible, the layout designer will avoid using the POLY gates (high resistance)

as conductor for signals.



In the illustration, Figure A shows a good configuration. B and C are not so good (with C being worse than B) as the gates are being used as conductors for the signals.

3.2.7.17 Filler cells

Filler cells are used to fill in the spaces left by the placer after all the regular cells that constitute the design have been placed. In addition, they are allowing to insure continuity of the power supply rails, the NWELL area and the Pimp/Nimp rails.

Filler cells are created for multiple width of the placement grid (1, 2 and 4).

When possible (in FILL4) additional substrate contacts are also added.

3.2.8 Abstract & LEF

3.2.8.1 Technology section of LEF

We expect that we will have to create the technology section of the LEF by hand, based upon the technology information available from the GPDK, and the routing grid definition presented earlier.

We will target the LEF 5.4 format, although it is not clear at this time that we will be able to fill in all the technology sections, based upon the currently available information for the GPDK. We might have to use some creativeness for some of the sections and extrapolate generic (but meaningful and consistent with the GPDK values) values for some parameters, based upon the review of other similar process accessible to us.

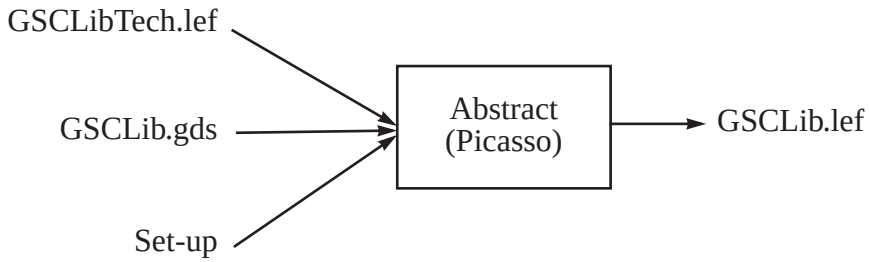
We plan to work with the PDK factory to insure that once this work is done, the technology file from the GPDK is updated to contain this information, hence enabling the attachment of the GSCLib to the GPDK.

This section will also be provided as a standalone file, as it is required when users are going to want to create their own abstracts using Picasso for blocks in their designs. Here again, we will work with the PDK factory, to define a standard location for this file, in the PDK distribution directory.

3.2.8.2 MACRO section of LEF (standard cells & Filler cells)

The MACRO section of the LEF describes the abstracted version of the layout cellview used by the P&R tools. We will use Picasso (aka “abstract”) to generate the LEF information for our

standard cell library. The technology information will be obtained from the technology LEF file (see section 3.2.8.1 on page 18). The data transfer between Picasso and dfII is not transparent today, and requires the use of an intermediate GDSII file.



• ***Stream-out set-up***

A SKILL procedure is available to allow dumping a single GDSII file, for all the valid cells in the library that have a layout. A valid cell is defined as a cell with a name ending in Xn, where n is taken from 1,2,3,4,5,6,7,8,16. The SKILL procedure is automatically available, and is loaded from the GSCLib libInit.il file.

```
GSCLibPipo(libName)
```

The output GDSII is placed in the run directory, and is called <libName>.gds.

• ***Abstract set-up***

1 - The first step is to create a technology. This can be done by combining the technology LEF file provide in the GSCLib/lef directory as `GSCLibTech.lef` and the GDSII layer mapping file provided in GSCLib/stream as `stream.map`.

2 - Then one needs to load the GDSII file containing the library element layouts. If created with the provided SKILL utility, it will be called GSCLib.gds.

3 - Then one needs to load the logical description of the cells. This can be achieved by either loading the TLF file, or the verilog file. We will be using the GSCLIB.v file provided in the GSCLib/verilog directory.

4 - Set-up a different bin (“THIN”) for the cell FILLER1. All the other cells can go in the default “CORE” bin.

5 - Pin step

5.1 - It is not needed to specify any special TEXT layer for the pins.

5.2 - Please specify “POWR” for the power pin

5.3 - Please specify “GRND” for the ground pin

5.4 - Please specify “Y Q QN” for the output pin list

5.5 - Run...

6 - Extraction steps

6.1 - The only special set-up there is that we want to limit the extraction of power nets to at least elements as wide as the power pin. Set under the “Power” tab, the minimum

width to 1.08 µm.

Note: Note: the smaller FILLER cell has a width inferior to that minimum width, and therefore needs to be taken care of in a separated bin (“THIN”), otherwise it will be completely missing the POWR/GRND pins.

6.2 - Run...

7 - Abstract step

7.1 - For some reason, Picasso does not pick the grid from the technology. Please make sure to specify 0.66/0.33 & 0.66/0.33 for Metal1/Metal2 pitch and offset.

7.2 - Run...

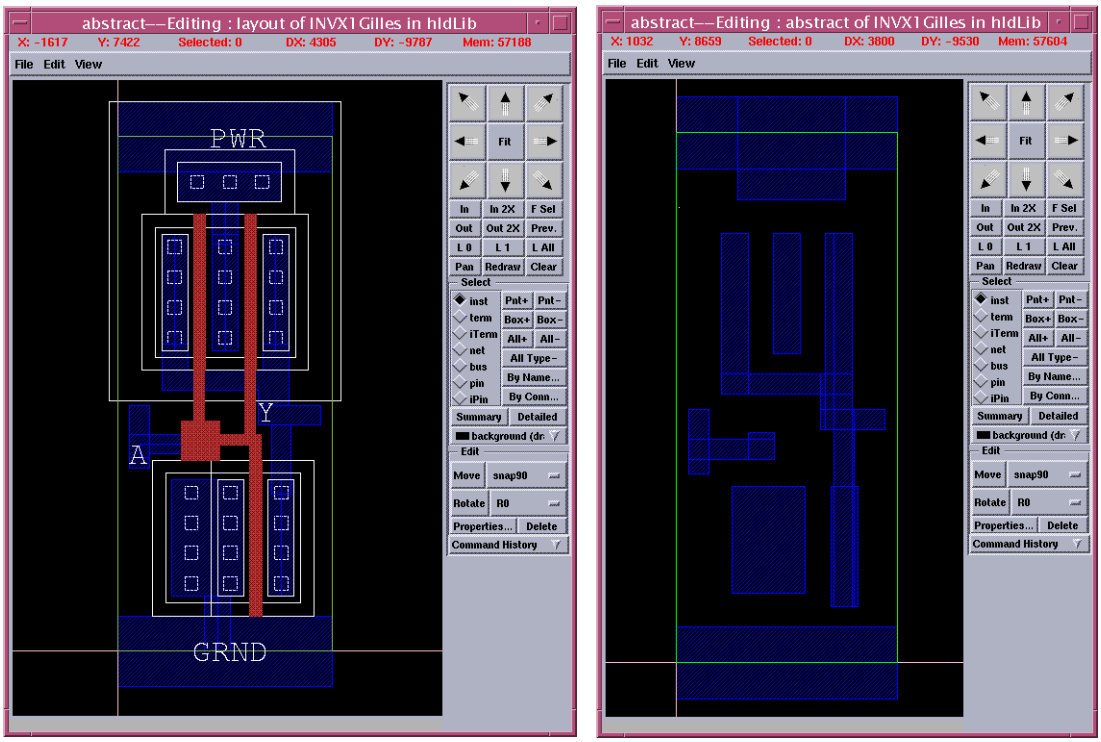
8 - Verification step

8.1 - Make sure to ask for several orientations, this allows checking that when rotating and abuting the cells, we still have DRC correct layout.

9 - Save the resulting LEF file

9.1 - You need to save all the bins (“CORE” and “THIN”) , including the technology section.

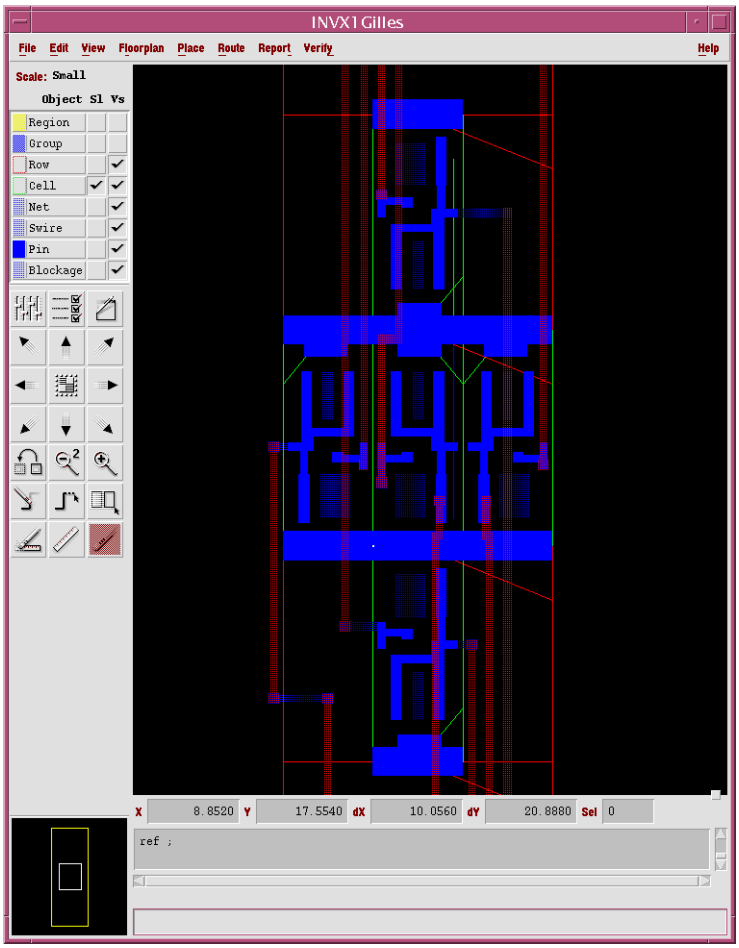
The following screen-shots show the layout cellview and the produced abstract cellview in the



“abstract” tool.

- ***Verification with Silicon Ensemble***

The last step of the abstract generation is to verify that the abstract, when placed in a design



can be routed. “Abstract” provides a nice utility that creates a simple DEF file with the cells, and allows to run placement and routing for it. The picture here shows different configuration of the inverter, with different rotations/symetries, and the routing obtained using WROUTE.

3.2.8.3 abstracts

Abstracts in the dfII library for each individual standard cell will be created using the LEF In process, from the LEF file generated out of Picasso. The regular power supply pin names created by LEFIn have to be replaced to be inherited connection net expressions, to enable full compatibility with the layout views, or prFlatten. A SKILL procedure is provided to automatically add the appropriate net expressions on the terminal.

```
GSCLibPostProcessAbstracts(s_libname)
```

Note: The `netExpression` itself is not visually displayed in the abstract. It is however visible if the pin is selected, in the “connectivity” tab.

3.2.9 TLF

With the recent acquisition of Simplex, Cadence has acquired the ability to characterize standard cell librairies with the signalstorm tool suite. The input to the characterization process is a set of subcircuits in Spice format, the Spectre models and the corner information. Signalstorm automatically perform the functionality recognition for the cells, and generate the

appropriate simulation vectors.

3.2.9.1 Generation of subcircuit file (Spectre-like format)

The following SKILL procedure can be used in the CIW to generate a single file containing all the subcircuits for the library (the function is included in the library and automatically loaded from the libInit.il file).

```
GSCLibSubcktFile(libName viewList fileName)
```

where:

- *libName* is the name of the library to scan for cellviews (“GSCLib”)
- *viewList* is a list of ordered string showing the viewname to use (“extracted” “schematic”)
- *fileName* is the name of the file to write the subcircuits into.

A set-up file reflecting the characterization conditions described below is used to drive the characterization process.

3.2.9.2 Characterization conditions.

• Voltage

The Generic PDK is the representation of a generic 3V process. The library will be characterized for at least the following working conditions, using the nominal process corner:

Voltage: 2.7 V, 3.0 V, 3.3V

• Temperature

Temperature: -30^{o}C, 27^{o}C, 125^{o}C.

If resource runs low, we will give priority to characterization for the multiple voltage points over the multiple temperatures.

• Input slew rates

Discussion with our partners (Texas Instruments) shows that the choice of the input slews used to create the characterization table is a critical part of the set-up of the characterization information. Based upon the review of other similar process, we are proposing to run characterizations, for the following input slew rates:

Input slew (ns) : 0.05, 0.15, 0.60, 1.40, 2.30, 3.30, 4.50

Maximum allowable slew (pin property) 4.50 nS

• Output loads

Output loads need to cover a range large enough for our design needs. They are also different in range for cells with larger drive strength (for example a X4 cell) than cells with smaller drive strength (X1). The basic value range is defined for the X1 type cells. A Xn type cell output load range is defined by multiplying each individual measurement point.

The basic value for the output load is based upon measuring the dynamic input capacitance of the INVX1 cell, and taking for it a reasonable average value. Our measurements lead to a value of 10fF for the basic load unit.

Output loads (X*n*) (pf): 0.005 x *n,* 0.01 x *n,* 0.015 x *n,* 0.02 x *n,* 0.03 x *n,* 0.04 x *n,* 0.1 x *n*

Built in here is the assumption that the designer will not attempt to drive a small load (0.01 pf)

for example with an X4 type cell.

3.2.9.3 Additional information (Area / Scan attributes).

The liberty (.lib) files generated out of the subcircuits by signalstorm do not include the area information, as they are not derived from the physical data. This value is used during synthesis, as a parameter for optimization. It is necessary to add this information to the .lib file as an additional step.

The area of the cell is defined as the area of the prBoundary rectangle. A SKILL utility is provided to dump (GSClib/libInit.il), a file containing the areas of each cell in the library:

```
GSCLibArea(s_libName s_fileName)
```

Once characterization is completed using signalStorm, a first .lib file is generated using the alf2lib utility. Then, this file is broken down into tables, using the tbliberty utility. Finally, using the merge.pl Perl script (from GSClib/bin), the GSCLib.area file and the liberty_cell.txt files are merged. The last step consist into reforming the .lib file with the different tables.

In addition, the pins of the scan cell needs to be marked as such in order for the correct test_cell() block to be generated in the .lib file. This is done using a property file, used as a command line optional argument when invoking the alf2lib utility.

```
cell SDF*{
  pin SE{
      direction input;
      scan enable;
  };
  pin SI{
      direction input;
      scan in;
  }
  pin Q*{
      direction output;
      scan out;
  };
};
```

3.2.9.4 syn2tlf (.lib -> .tlf)

Although the latest versions of buildGate are able to directly load a liberty (`.lib`) file, we will provide a `.tlf` file for each characterization corner. This is done using the syn2tlf utility. It is important to make sure that the output format is set to be at least 4.3, as this is the mimimum acceptable version for buildgate. In addition, the parameters describing how the timing characterization has been performed are also specified. Default values could be automatically picked up by the latest version of the syn2tlf utility, but they do not match the ones from the characterization set-up.

```
syn2tlf GSCLib_nom_33_27.lib -format 4.3\
  -ir 50 -if 50 -dr 50 -df 50 -sr 10 -sf 10 -tr 90 -tf 90\
  -o GSCLib_nom_33_27.tlf
```

3.2.10 Wire load models

Wire load models is a statistical model representing average wire loads for different number of

wire segments. Wire load models can only be constrcuted after gathering statistical data from several P&Red design. If the user community provides us this information, we will consider including it in subsequent releases of the library.

At this time, the different flows move toward physically knowledgeable synthesis, or even, in the 900 nm areas, to full real wire models... In such environment, better and more accurate wire models are directly extracted from the design.

3.2.11 Noise characterization (CeltIC).

The noise information library for each element in the library is created using Cadmos/CeltIC `make_cdb` utility. This utility however requires SPICE models and at this time, GPDK only includes Spectre models.

3.2.12 Documentation

3.3 Unix directory structure

We propose to deliver the generic standard cell library in a Unix directory, following the structure described below. The dfII library itself is a standard 5.x library. It is attached to the GPDK process information using the standard technology file attachement mechanism.

Versioning is done at this level. When an upgrade of the library is made, it is recommended that the user replace the whole directory, as we will not test individual components of individual versions against each other.

GSCLib_x.x/GSCLib	*CDS 5.x library directory*
GSCLib_x.x/timing/low.tlf	*2.7 Volt, TLF file*
GSCLib_x.x/timing/nom.tlf	*3.0 Volt, TLF file*
GSCLib_x.x/timing/high.tlf	*3.3 Volt, TLF file*
GSCLib_x.x/timing/GSCLib.sp	*Spectre subcircuits for the elements.*
GSCLib_x.x/bin	*binary utility directory*
GSCLib_x.x/scripts	*Scripts used in different part of lib.*
GSCLib_x.x/bin/GSCLibtech.lef	*Technology section of LEF file*
GSCLib_x.x/lef/GSCLib.lef	*MACRO section of LEF file*
GSCLib_x.x/BG/GSCLib.slib	*BuildGates symbol library*
GSCLib_x.x/verilog	*Verilog directory with models (.v)*
GSCLib_x.x/doc	*Release notes & docs.*

3.4 Recommended user set-up

It is recommended that all the process information (PDK and libraries) be stored in a single location, visible from any points in the network. In the ideal case, a dedicated user should be created, and be used to manage this directory. We will assume further that this user name is “process” and that automount is set-up so that the “~process” UNIX path is resolved automatically to the physical location of the process directory in the network.

All the files in this directory should be write protected and accessible in read mode only from the normal users.

As both process and library version change on a regular basis, it is convenient to be both able to specify a *specific* version (to be insensitive to changes in a critical time of the design) or a generic version.

The following directory structure gives the ability to work in both of these modes.

```
~process/
            gpdk_MIET_1.4/
            gpdk_MIET_1.5/
            gpdk_MIET_1.9/
            gpdk -> gpdk_MIET_1.9
            GSCLib_0.7/
            GSCLib_0.8/
            GSCLib_0.9/
            GSCLib_1.0/
            GSCLib -> GSCLib
```

For the users who want to use a specific version of the library or of the process, the entries in the cds.lib would look like:

```
DEFINE gpdk ~process/gpdk_MIET_1.5/gpdk
DEFINE GSCLib ~process/GSCLib_0.9/GSCLib
```

For the users who want to use the “current” version of the library and process, the entries in the cds.lib would look like:

```
DEFINE gpdk ~process/gpdk/gpdk
DEFINE GSCLib ~process/GSCLib/GSClib
```

```
It is possible to define a 2 level structure for the cds.lib,
where a master cds.lib is defined with the libraries used by a
group, and the end user only includes that cds.lib in his/her
local cds.lib.
```

3.5 Supported flows

3.5.1 Full custom design flow

3.5.1.1 Schematic capture

The supported tool for interactive schematic capture is Cadence Composer. The user will be able to select any cell from the library, and freely place it amongst other devices, including compatible primitives from the GPDK.

The user will be able to select either the view without explicit pins for the power supplies (the default), or the view with explicit terminals. If the view with explicit terminals is used, the user can optionally draw a wire to the pin representing that terminal to complete the connection the assignment of the power supply.

We will support the use of verilogIn to create schematic, as a way to create the verilog gate representation of the inners of a digital block. In this case, the default symbols (no explicit power supplies) will be used, and the power supply overwrite can only be performed on the instance of the block being created, and not within the block itself (see section 3.2.3.2 on page 8).

3.5.1.2 Virtuoso

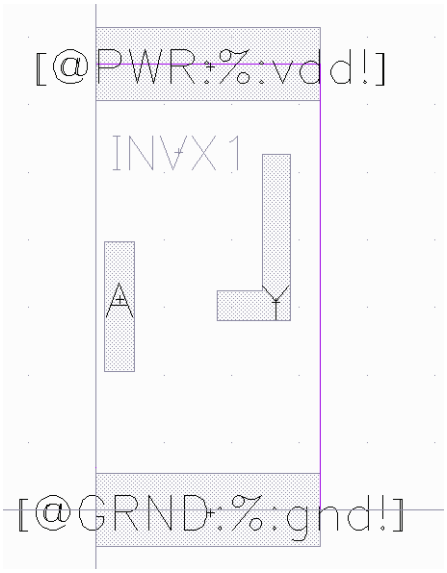
Virtuoso is the basic full custom editing tool supported. Users will have a layout view available for each individual cell, and will be able to place such cell in their layout.

3.5.1.3 Virtuoso-XL (basic)

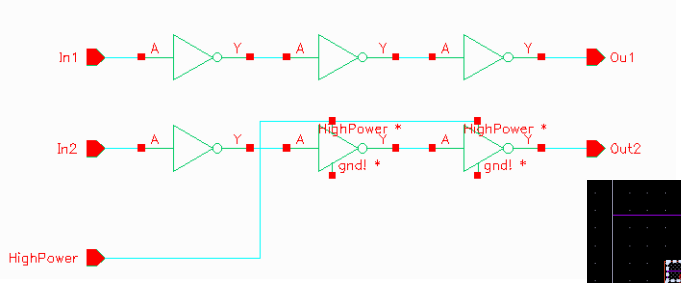
Virtuoso is the connectivity driven version of Virtuoso. All the cells will be created with pins matching their symbols, in name and number. As a result, Virtuoso-XL will be supported.

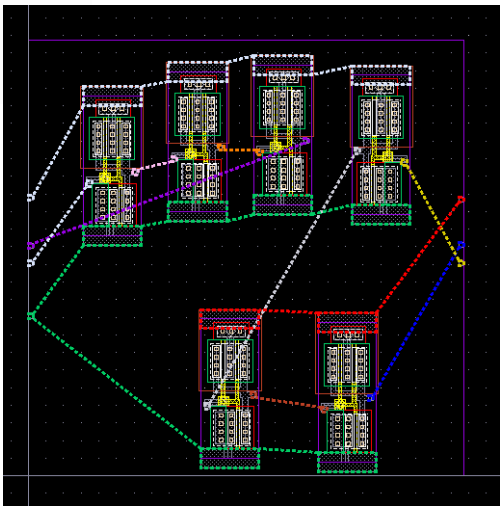
3.5.1.4 Layout drawing guidelines: Supply pin naming.

The power and ground pins in the layout must be defined like terminals with netSet expressions, to be compatible with the schematics.



The following pictures shows an example of how a basic schematic using inverters will look like in VXL. Note that the inherited connections are correctly interpreted, both for the explicit and implicit connections, including the local physical overwrite (“highPower” local pin is connecting the PWR pins of the 2 inverters located in the lower right corner of the layout, while the others are connected together to a pin with a net expression on the left).





3.5.1.5 Virtuoso-XL (VCP)

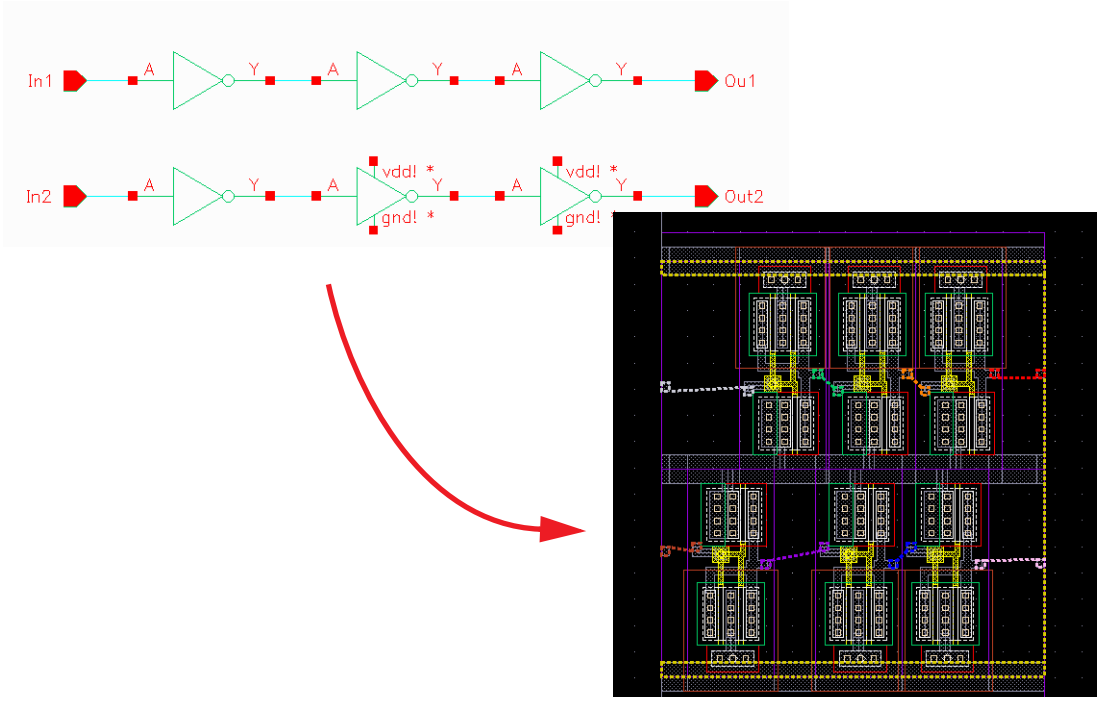
Virtuoso Custom Placer (VCP) an option to VXL, allow automatic placement of a small set of standard cells.

It is easier for the end-user if the library containing the reference standard cells is already added to the list of reference libraries recognized by VCP. This can be done by setting an environmental variable (in IC 5.0). We plan to use the Skill equivallent in the library attached file, so that this will always be set-up automatically for the user.

```
envSetVal("layoutXL" "compTypeRefLibs" 'string "GSCLib")
```

Other VCP properties are design dependant, and do not appear to be pre-settable (component type is one we are going to be looking at however).

The following pictures, shows a simple schematic, and the result of the placer invokation.

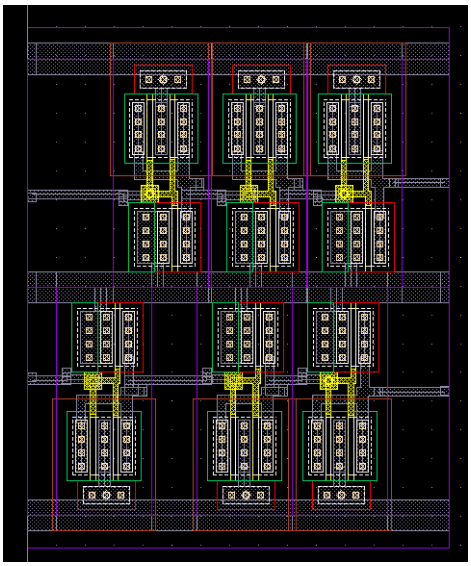


3.5.1.6 Virtuoso-XL (VCR)

Virtuoso Custom Router (VCR) an option to VXL, allow routing of a number of net corresponding to the needs of the custom designer (up to a few thousands). Using the technology information and rules provided from the GPDK, VCR is directly available to be used on the standard cell library. VCR is not a timing driven router, and will not use the TLF information from the library.

The following picture is a representation of the same layout as shown in the previous example,

after it has been auto-routed by VCR.



3.5.1.7 Preview

Preview is no longer a supported tool... However it is used. We believe that by providing the abstract view in dfII, we are enabling the usage of Preview. There might however be issues with the inherited connections resolutions. We will not focus on this tool.

3.5.1.8 Diva (DRC, Extraction, LVS)

Each individual standard cell in the library will be created with the intend of being free of DRC/ LVS error. In addition, it should be possible to abut such cells manually, in a manner similar to a Silicon Ensemble placement, and not create DRC errors.

No special rules need to be create to enable Diva Support. This is provided by the GPDK.

Although back-box extraction is supported in Diva, it is not a flow that we plan to test or provide views for at this time. The main target for this flow was the defunct analog artist msps (mix-signal back-annotation) flow.

3.5.2 “Digital on top” SoC flow

In such flows, the analog content is “packaged” and represented in the digital tools as a block box.

3.5.2.1 Packaging of analog blocks.

- ***LEF/abstracts***

The designer can use the technology LEF file provided with the library to initialize Picasso (the abstract generation tool). We do not plan to provide any additional support as part of this effort.

- ***TLF***

We have no plans to provide as part of this effort a TLF generator for user created analog blocks.

3.5.2.2 First Encounter

WHAT IS NEEDED TO SUPPORT THIS TOOL THAT WE DO NOT ALREADY HAVE ?

3.5.2.3 Silicon Ensemble (SE-PKS) 5.4

The supported flow will take as an input:

1 - a Verilog netlist representing the design,

2 - the technology GPDK.lef, as well as the GSCLib.lef containing the standard cell description,

3 - the TLF file containing the characterized timing information for the standard cell library.

Note: A bug exists in VAN when compiling the verilog file to create the binary version of the library. Please make a copy of the Verilog library file, and commend out the offending statement ($recrem and/or $remove). This will not affect the behavior of Silicon Ensemble.

3.5.2.4 BuildGates

To support BuildGates, in addition to the TLF file, that contains the description of the timing information for each cell, as well as its logical function, we also need symbols. These symbols need to be created from the dfII symbols without explicit power pin connections, as BuildGate has no understanding of the inherited connections constructs at this time.

Standard format for translating data between the synthesis world and dfII is (beside Verilog for the actual netlist) EDIF 200. We will use EDIF Out to create symbols for each cell in the digital library. These will be read into buildGates, and combined with the TLF to create the final version of the buildGates Library for the GSCLib (using the **`slibconv`** utility from the BuildGate tree).

The use of the same symbol between dfII and our synthesis library will also enable (although not directly critical to our purpose here) the import of schematics created in the Synopsys environment.

3.5.2.5 Parasitic extraction

- ***Hyper Extract***

HOW DO I GET A tech.dpux ????

- ***Simplex (Fire & Ice)***

WHAT IS NEEDED HERE ?

3.5.2.6 Power analysis

This is deferred to a second release of the library.

4. QUALITY

4.1 Performance Requirements

There are no direct performance requirement on the library itself. However, it should be a goal of the team to insure that the library shows the Cadence tools under their best light/ For that, the team will endeavour to produce:

- layouts that are reasonably sized,

- behavioral models that have no convergence problems,

- TLF that have sufficient (but not too complex) level of accuracy.

Quality Requirements

4.1.1 Quality goal

Because this library is intended to show the Cadence tools and the Cadence flows to a large group of potential users (from Universities, to customers), it should be a clear goal to build this library with high quality goals in mind. However, because of the volunteer nature of the work, this goal might be limited by the actual time available. In such case, the compromise of choice will be to reduce the number of cells in the library, rather than the actual quality or a representation.

On the positive side however, it should be noted that because of its non confidentiality and generic aspect, this library is likely to be used by a large number of people, and we expect that suggestions for improvements will be readily available (maybe too many in fact).

Finally, we hope that, again because of its generic aspect, it will be possible to place several regression tests into the Cadence main regression flows, based upon this library. This should contribute to not only the overall quality of the library itself, but to the overall quality of the Cadence Tools as well.

4.1.2 Maintainability

Maintenance of such a library is going to be an issue, as the base for its construction is the usage of interns. Efforts will be made to insure proper documentation of the flows used during its creation.

4.1.3 Installability

This library should be provided as a stand alone library, with the largest possible amount of data include in a single 5.x hierarchy structure. This will insure easy and consistent installation process for the user (simply add one line to the cds.lib file).

4.1.4 Other Quality Issues

4.1.5 Product Packaging and Licensing

No licensing of any kind required for this library.

4.1.6 Operating Environment

This library is intended to be use in the same type of environment as the Cadence Tools.

4.1.7 Development Environment

It is expected that the development work for this library will be conducted in the Cadence

facilities around the world, involving intern students in the US and in Russia.

We propose that the main integration area will be maintained in Moscow. In addition, to insure that all parties are using the same version of the software and of the GPDK, a set of tarkits will be made available from the Moscow facility, for the people involved in the developement work for the library.

Finally, the software configuration used for the development and testing environment will be made public (wich version of each stream) within the released directory of the library in a README file, to allow potential users to replicate the same environment.

5. TECHNICAL DEPENDENCIES

The development of the timing library for this library is dependant on the good will and time available to our partners, Motorola in Moscow and Cadence service folks in Germany.

In addition, several problems still exists in the IC 5.0 release to support inherited connections. These are centered around the support of verilogA test representations, and the physical overwrite of inherited connections. PCRs have been filed, but the usability of this library is dependant on the resolution of these PCRs. Such PCRs are tracked in the PCR system with the keyword “GSCLib”.