ALLIANCE & CORIOLIS Checker Toolkit

Contents

Toolkit Purpose

This toolkit has been created to allow developpers to share through `git` a set of benchmarks to validate their changes in ALLIANCE & CORIOLIS before commiting and pushing them in their central repositories. A change will be considered as validated when all the developpers can run successfully all the benchs in their respective environments.

As a consequence, this repository is likely to be *very* unstable and the commits not well documenteds as they will be quick corrections made by the developpers.

Toolkit Contents

The toolkit provides:

- Eight benchmark designs (unchecked are not counted and multiple target technologies count for one).

Design	**Technology**	**Cell Libraries**	**Status**
`adder`	MOSIS	`nsxlib`, `mpxlib`, `msplib`	Unchecked
AM2901 (standard cells)	ALLIANCE symb. cmos	`sxlib`, `pxlib`	OK
AM2901 (datapath)	ALLIANCE symb. cmos	`sxlib`, `dp_sxlib`, `pxlib`	OK
`alliance-run` (AM2901)	ALLIANCE symb. cmos	`sxlib`, `dp_sxlib`, `padlib`	Unchecked
`RingOscillator`	ALLIANCE symb. cmos	`sxlib`	OK
CPU	MOSIS	`nsxlib`, `mpxlib`, `msplib`	OK
SNX			
SNX / Alliance	ALLIANCE symb. cmos	`sclib`	Unchecked
SNX / sxlib2M	ALLIANCE symb. cmos 2M	`sxlib`	OK
SNX / cmos	ALLIANCE symb. cmos	`sxlib`, `pxlib`	OK
SNX / cmos45	ALLIANCE symb. cmos 45	`nsxlib`, `mpxlib`	OK
SNX / FreePDK_45	FreePDK 45	`gscl45`	OK
SNX / c35b4	AMS 350nm c35b4	`corelib`	KO
6502	ALLIANCE symb. cmos 45	`nsxlib`	OK
MIPS (microprogrammed)	ALLIANCE symb. cmos	`sxlib`, `dp_sxlib`, `rf2lib`	OK
MIPS (pipeline)	ALLIANCE symb. cmos	`sxlib`, `dp_sxlib`, `rf2lib`	OK
MIPS (pipeline+chip)	ALLIANCE symb. cmos	`sxlib`, `dp_sxlib`, `rf2lib`, `pxlib`	Unchecked
FPGA (`Moc4x4_L4C12`)	ALLIANCE symb. cmos	`sxlib`	KO
ISPD05 (`bigblue1`)	None	Generated on the fly	Unchecked
ARMV2A	ALLIANCE symb. cmos	`sxlib`, `pxlib`	OK
Vex RISC-V			
VEXRISCV / cmos	ALLIANCE symb. cmos	`sxlib`, `pxlib`	OK
VEXRISCV / cmos45	ALLIANCE symb. cmos 45	`nsxlib`, `mpxlib`	OK
VEXRISCV / FreePDK_45	FreePDK 45	`gscl45`	KO
VEXRISCV / c35b4	AMS 350nm c35b4	`corelib`	KO

- Three cell libraries.

  All thoses libraries are for use with MOSIS and FREEPDK45 technologies. We provides them as part of the toolkit as we are still in the process of validating that technology, and we may have to perform quick fixes on them. The design are configured to use them instead of those supplied by the ALLIANCE installation.

  - `nsxlib` : Standard Cell library.
  - `mpxlib` : Pad library, compliant with CORIOLIS.
  - `msplib` : Pad library, compliant with ALLIANCE / `ring`. Cells in this library are *wrappers* around their counterpart in `mpxlib`, they provides an outer layout shell that is usable by `ring`.

- The RDS files for MOSIS (`scn6m_deep_09.rds`) and FREEPDK45 technologies, for the same reason as the cell libraries.

- Miscellenous helper scripts.

Toolkit Layout

The files are organized as follow :

Directory	Contents
`./etc/`	Configuration files
`./etc/mk/`	Makefiles rules to build benchmarks. This directory must be symbolic linked into each benchmark directory
`./etc/mk/users.d/`	Directory holding the configuration for each user
`./bin/`	Additionnal scripts
`./cells/<LIBDIR>`	Standard cells libraries.
`./benchs/<BENCH>/<techno>/`	Benchmark directories
`./doc/`	This documentation directory

Benchmark Makefiles

A benchmark `Makefile` is build by setting up variables `USE_<FEATURE>=Yes/No` then including the set of rules `./mk/design-flow.mk`. The directory `alliance-check-toolkit/etc/mk/` must be symlinked in the directory where the `Makefile` resides.

The `Makefile` provides some or all of the following targets. If the place and route stage of a benchmark has multiple target technology, one directory is created for each.

CORIOLIS	`blif`	Synthetize the netlist with `Yosys`.
	`layout`	The complete symbolic layout of the design (P&R).
	`gds`	Generate the real layout (GDSII)
	`druc`	Symbolic layout checking
	`lvx`	Perform LVS.
	`graal`	Launch `graal` in the `Makefile` 's environement
	`dreal`	Launch `dreal` in the `Makefile` 's environement, and load the `gds` file of the design.
	`view`	Launch `cgt` and load the design (chip)
	`cgt`	Launch `cgt` in the `Makefile` 's environement

A top `Makefile` in a bench directory must looks like:

```
         LOGICAL_SYNTHESIS = Yosys
        PHYSICAL_SYNTHESIS = Coriolis
                DESIGN_KIT = nsxlib45

             USE_CLOCKTREE = No
                 USE_DEBUG = No
                USE_KATANA = Yes

                  NETLISTS = VexRiscv

 include ./mk/design-flow.mk

blif:   VexRiscv.blif
layout: vexriscv_r.ap
gds:    vexriscv_r.gds

lvx:    lvx-vst-vexriscv
drc:    druc-vexriscv_r
```

Where variables have the following meaning:

Variable	Usage
`LOGICAL_SYNTHESIS`	Tells what synthesis tool to use between `Alliance` or `Yosys`. Netlists must be pre-generated if this variable is empty or not present.
`PHYSICAL_SYNTHESIS`	Tells what place & route tools to use between `Alliance` (i.e. `ocp`, `nero` & `ring`) and `Coriolis`.
`DESIGN_KIT`	The target technology and the standard cell libraries to use, for the supported values see below.
`NETLISTS`	The list of *netlists* that are requireds to perform the place and route stage. The files must we given *without* extension. According to the value of `LOGICAL_SYNTHESIS` they are user supplied or generated. In the later case, be aware that calling the `clean` target will remove the generated files. In certain contexts, the first item of `NETLISTS` will be considered as the chip's core. Note that the clean will remove all generated files.
`USE_CLOCKTREE`	Adds a clock-tree to the design (CORIOLIS).
`USE_DEBUG`	Use the debugger enabled version of `cgt`.
`USE_KATANA`	Use the new router `katana` (mixed signal) instead of `kite` (digital only). `katana` do not manage I/O pads yet.

Availables design kits (to set `DESIGN_KIT`):

Value	Design kit
`sxlib`	The default ALLIANCE symbolic technology. Use the `sxlib` and `pxlib` libraries.
`nsxlib45`	The symbolic technology fitted for 180nm and below. Used for FREEPDK45 in symbolic mode.
`FreePDK_45`	Direct use of the real technology FREEPDK45.
`c35b4`	AMS 350nm c35b4 real technology.

Setting Up the User's Environement

Before running the benchmarks, you must create a configuration file to tell where all the softwares are installeds. The file is to be created in the directory:

```
alliance-check-toolkit/etc/mk/users.d/
```

The file itself must be named from your username, if mine is `jpc`:

```
alliance-check-toolkit/etc/mk/users.d/user-jpc.mk
```

Example of file contents:

```
# Where Jean-Paul Chaput gets his tools installeds.

export NDA_TOP       = ${HOME}/crypted/soc/techno
export AMS_C35B4     = ${NDA_TOP}/AMS/035hv-4.10
export FreePDK_45    = ${HOME}/coriolis-2.x/work/DKs/FreePDK45
export CORIOLIS_TOP  = $(HOME)/coriolis-2.x/$(BUILD_VARIANT)$(LIB_SUFFIX_)/$(BUIL
export ALLIANCE_TOP  = $(HOME)/alliance/$(BUILD_VARIANT)$(LIB_SUFFIX_)/install
export CHECK_TOOLKIT = $(HOME)/coriolis-2.x/src/alliance-check-toolkit
export AVERTEC_TOP   = /dsk/l1/tasyag/Linux.el7_64/install
export YOSYS_TOP     = /usr
```

All the variable names and values are more or less self explanatory...

CORIOLIS Configuration Files

Unlike ALLIANCE which is entirely configured through environement variables or system-wide configuration file, CORIOLIS uses configuration files in the current directory. They are present for each bench:

- `<cwd>/.coriolis2/techno.py` : Select which symbolic and real technology to use.
- `<cwd>/.coriolis2/settings.py` : Override for any system configuration, except for the technology.

CORIOLIS and Clock Tree Generation

When CORIOLIS is used, it create a clock tree which modificate the original netlist. The new netlist, with a clock tree, has a postfix of `_clocked`.

> **Note**
>
> **Trans-hierarchical Clock-Tree.** As CORIOLIS do not flatten the designs it creates, not only the top-level netlist is modificated. All the sub-blocks connected to the master clock are also duplicateds, whith the relevant part of the clock-tree included.

RHEL6 and Clones

Under RHEL6 the developpement version of CORIOLIS needs the `devtoolset-2`. `os.mk` tries, based on `uname` to switch it on or off.

Yosys Auxiliary Script

As far as I understand, `yosys` do not allow it's scripts to be parametriseds. So, for each VERILOG file that has to be synthetized, a simple script must be provided. Here is a basic example: `VexRiscv.ys`:

```
set verilog_file VexRiscv.v
set verilog_top  VexRiscv
set liberty_file .../alliance-check-toolkit/cells/nsxlib/nsxlib.lib
yosys read_verilog          $verilog_file
yosys hierarchy -check -top $verilog_top
yosys synth            -top $verilog_top
yosys dfflibmap -liberty    $liberty_file
yosys abc       -liberty    $liberty_file
yosys clean
yosys write_blif VexRiscv.blif
```

Benchmarks Special Notes

alliance-run

This benchmark comes mostly with it's own rules and do not uses the ones supplieds by `rules.mk`. It uses only the top-level configuration variables.

It a sligtly modified copy of the `alliance-run` found in the ALLIANCE package (modification are all in the `Makefile`). It build an AM2901, but it is splitted in a control and an operative part (data-path). This is to also check the data-path features of ALLIANCE.

And lastly, it provides a check for the CORIOLIS encapsulation of ALLIANCE through PYTHON wrappers. The support is still incomplete and should be used only by very experienced users. See the `demo*` rules.

Libraries Makefiles

Note

For those part to work, you need to get `hitas` & `yagle`:

HiTas -- Static Timing Analyser

The `bench/etc/mk/check-library.mk` provides rules to perform the check of a library as a whole or cell by cell. To avoid too much clutter in the library directory, all the intermediate files generated by the verification tools are kept in a `./check/` subdirectory. Once a cell has been validated, a `./check/<cell>.ok` is generated too prevent it to be checked again in subsequent run. If you want to force the recheck of the cell, do not forget to remove this file.

Checking Procedure

- DRC with `druc`.
- Formal proof between the layout and the behavioral description. This is a somewhat long chain of tools:
  1. `cougar`, extract the spice netlist (`.spi`).
  2. `yagle`, rebuild a behavioral description (`.vhd`) from the spice netlist.
  3. `vasy`, convert the `.vhd` into a `.vbe` (Alliance VHDL subset for behavioral descriptions).
  4. `proof`, perform the formal proof between the refence `.vbe` and the extracted one.

Rule or File	Action
`check-lib`	Validate every cell of the library
`clean-lib-tmp`	Remove all intermediate files in the `./check` subdirectory **except** for the `*.ok` ones. That is, cells validated will not be rechecked.
`clean-lib`	Remove all files in `./check`, including `*.ok`
`./check/<cell>.ok`	Use this rule to perform the individual check of `<cell>`. If the cell is validated, a file of the same name will be created, preventing the cell to be checked again.

Synopsys Liberty .lib Generation

The generation of the liberty file is only half-automated. `hitas` / `yagle` build the base file, then we manually perform the two modifications (see below).

The rule to call to generate the liberty file is: `<libname>-dot-lib` where `<libname>` is the name of the library. To avoid erasing the previous one (and presumably hand patched), this rule create a `<libname>.lib.new`.

1. Run the `./bin/cellsArea.py` script which will setup the areas of the cells (in square um). Work on `<libname>.lib.new`.
2. For the synchronous flip-flop, add the functional description to their timing descriptions:

```
cell (sff1_x4) {
  pin (ck) {
    direction : input ;
    clock : true ;
    /* Timing informations ... */
  }
```

```
    pin (q) {
      direction : output ;
      function : "IQ" ;
      /* Timing informations ... */
    }
    ff(IQ,IQN) {
      next_state : "i" ;
      clocked_on : "ck" ;
    }
  }

  cell (sff2_x4) {
    pin (ck) {
      direction : input ;
      clock : true ;
      /* Timing informations ... */
    }
    pin (q) {
      direction : output ;
      function : "IQ" ;
      /* Timing informations ... */
    }
    ff(IQ,IQN) {
      next_state : "(cmd * i1) + (cmd' * i0)" ;
      clocked_on : "ck" ;
    }
  }
```

Note
The tristate cells **ts_** and **nts_** are not included in the `.lib`.

Helpers Scripts

TCL scripts for `avt_shell` related to cell validation and characterization, in `./benchs/bin`, are:

- `extractCell.tcl`, read a spice file and generate a VHDL behavioral description (using `yagle`). This file needs to be processed further by `vasy` to become an Alliance behavioral file (`vbe`). It takes two arguments: the technology file and the cell spice file. Cell which name starts by `sff` will be treated as D flip-flop.
- `buildLib.tcl`, process all cells in a directory to buil a liberty file. Takes two arguments, the technology file and the name of the liberty file to generate. The collection of characterized cells will be determined by the `.spi` files found in the current directory.

Macro-Blocks Makefiles

The `bench/etc/mk/check-generator.mk` provides rules to perform the check of a macro block generator. As one library cell may be used to build multiple macro-blocks, one `Makefile` per macro must be provided. The *dot* extension of a `Makefile` is expected to be the name of the macro-block. Here is a small example for the register file generator, `Makefile.block_rf2`:

```
                  TK_RTOP = ../..
export       MBK_CATA_LIB = $(TOOLKIT_CELLS_TOP)/nrf2lib

include $(TK_RTOP)/etc/mk/alliance.mk
```

```
include $(TK_RTOP)/etc/mk/mosis.mk
include $(TK_RTOP)/etc/mk/check-generator.mk

check-gen: ./check/block_rf2_p_b_4_p_w_6.ok   \
           ./check/block_rf2_p_b_2_p_w_32.ok  \
           ./check/block_rf2_p_b_64_p_w_6.ok  \
           ./check/block_rf2_p_b_16_p_w_32.ok \
           ./check/block_rf2_p_b_32_p_w_32.ok
```

Note

In the `check-gen` rule, the name of the block **must** match the *dot* extension of the `Makefile`, here: `block_rf2`.

Macro-block generators are parametrized. We uses a special naming convention to pass parameters names and values trough the rule name. To declare a parameter, add `_p_`, then the name of the parameter and it's value separated by a `_`.

String in Rule Name	Call to the generator
`_p_b_16_p_w_32`	`-b 16 -w 32`

When multiple flavor of a generator could be built upon the same cell library, one `Makefile` per flavor is provided. To run them all at once, a `makeAll.sh` script is also available.

The `check-gen` rule only perform a DRC and a LVS to check that their router as correctly connected the cells of a macro-block. It doesn't perform any functional verification.

To perform a functional abstraction with `yagle` you may use the following command:

```
ego@home:nrf2lib> make -f Makefile.block_rf2 block_rf2_b_4_p_w_6_kite.vhd
```

Even if the resulting VHDL cannot be used it is always good to look in the report file `block_rf2_b_4_p_w_6_kit` for any error or warning, particularly any disconnected transistor.

Calling the Generator

A script `./check/generator.py` must be written in order to call the generator in standalone mode. This script is quite straigthforward, what changes between generators is the command line options and the `stratus.buildModel()` call.

After the generator call, we get a netlist and placement, but it is not finished until it is routed with the CORIOLIS router.

Note

Currently all macro-block generators are part of the STRATUS netlist capture language tool from CORIOLIS.

Scaling the Cell Library

This operation has to be done once, when the cell library is initially ported. The result is put in the `git` repository, so there's no need to run it again on a provided library.

The script is `./check/scaleCell.py`. It is very sensitive on the way the library pathes are set in `.coriolis2/settings.py`. It must have the target cell library setup as the `WORKING_LIBRARY` and the source cell library in the `SYSTEM_LIBRARY`. The technology must be set to the target one. And, of course, the script must be run the directory where `.coriolis2/` is located.

The heart of the script is the `scaleCell()` function, which work on the original cell in variable `sourceCell` (argument) and `scaledCell`, the converted one. Although the script is configured to use the *scaled* technology, this do not affect the values of the coordinates of the cells we read, whatever their origin. This means that when we read the `sourceCell`, the coordinates of it's components keeps the value they have under `SxLib`. It is *when* we duplicate them into the `scaledCell` that we perform the scaling (i.e. multiply by two) and do whatever adjustments we need. So when we have an adjustment to do on a specific segment, say slihgtly shift a `NDIF`, the coordinates must be expressed as in `SxLib` (once more: *before* scaling).

Note
There is a safety in `./check/scaleCell.py`, it will not run until the target library has not been emptied of it's cells.

The script contains a `getDeltas()` function which provide a table on how to resize some layers (width and extension).

As the scaling operations is very specific to each macro-block, this script is *not* shared, but customized for each one.

Tools & Scripts

One script to run them all: `go.sh`

To call all the bench's `Makefile` sequentially and execute one or more rules on each, the small script utility `go.sh` is available. Here are some examples:

```
dummy@lepka:bench$ ./bin/go.sh clean
dummy@lepka:bench$ ./bin/go.sh lvx
```

Command Line `cgt`: `doChip.py`

As a alternative to `cgt`, the small helper script `doChip.py` allows to perform all the P&R tasks, on an stand-alone block or a whole chip.

Blif Netlist Converter

The `blif2vst.py` script convert a `.blif` netlist into an ALLIANCE one (`vst`). This is a very straightforward encapsulation of CORIOLIS. It could have been included in `doChip.py`, but then the `make` rules would have been much more complicateds.

Pad Layout Converter `px2mpx.py`

The `px2mpx.py` script convert pad layout from the `pxlib` (ALLIANCE dummy technology) into `mpxlib` (MOSIS compliant symbolic technology).

Basically it multiplies all the coordinate by two as the source technology is 1µ type and the target one a 2µ. In addition it performs some adjustement on the wire extension and minimal width and the blockage sizes.

As it is a one time script, it is heavily hardwired, so before using it do not forget to edit it to suit your needs.

The whole conversion process is quite tricky as we are cheating with the normal use of the software. The steps are as follow:

1. Using the ALLIANCE dummy technology and in an empty directory, run the script. The layouts of the converted pads (`*_mpx.ap`) will be created.

2. In a second directory, this time configured for the MOSIS technology (see `.coriolis2_techno.conf`) copy the converted layouts. In addition to the layouts, this directory **must also contain** the behavioral description of the pads (`.vbe`). Otherwise, you will not be able to see the proper layout.

3. When you are satisfied with the new layout of the pads, you can copy them back in the official pad cell library.

Note

How Coriolis Load Cells. Unlike in ALLIANCE, CORIOLIS maintain a much tighter relationship between physical and logical (structural or behavioral) views. The loading process of a cell try *first* to load the logical view, and if found, keep tab of the directory it was in. *Second* it tries to load the physical view from the same directory the logical view was in. If no logical view is found, only the physical is loaded.
Conversely, when saving a cell, the directory it was loaded from is kept, so that the cell will be overwritten, and not duplicated in the working directory as it was in ALLIANCE.
This explains why the behavioral view of the pad is needed in the directory the layouts are put into. Otherwise you would only see the pads of the system library (if any).

CADENCE Support

To perform comparisons with CADENCE EDI tools (i.e. `encounter` NANOROUTE), some benchmarks have a sub-directory `encounter` holding all the necessary files. Here is an example for the design named `<fpga>`.

`encounter` directory	
File Name	**Contents**
`fpga_export.lef`	Technology & standard cells for the design
`fpga_export.def`	The design itself, flattened to the standard cells.
`fpga_nano.def`	The placed and routed result.
`fpga.tcl`	The TCL script to be run by `encounter`

The LEF/DEF file exported or imported by Coriolis are *not* true physical files. They are pseudo-real, in the sense that all the dimensions are directly taken from the symbolic with the simple rule `1 lambda = 1 micron`.

Note

LEF/DEF files: Coriolis is able to import/export in those formats only if it has been compiled against the SI2 relevant libraries that are subjects to specific license agreements. So in case we don't have access to thoses we supplies the generated LEF/DEF files.

The `encounter` directory contains the LEF/DEF files and the TCL script to be run by `encounter`:

```
ego@home:encounter> . ../../etc/EDI1324.sh
ego@home:encounter> encounter -init ./fpga.tcl
```

Example of TCL script for `encounter`:

```
set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
suppressMessage ENCEXT-2799
win
loadLefFile fpga_export.lef
loadDefFile fpga_export.def
floorPlan -site core -r 0.998676319592 0.95 0.0 0.0 0.0 0.0
getIoFlowFlag
fit
setDrawView place
```

```
setPlaceMode -fp false
placeDesign
generateTracks
generateVias
setNanoRouteMode -quiet -drouteFixAntenna 0
setNanoRouteMode -quiet -drouteStartIteration 0
setNanoRouteMode -quiet -routeTopRoutingLayer 5
setNanoRouteMode -quiet -routeBottomRoutingLayer 2
setNanoRouteMode -quiet -drouteEndIteration 0
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail
global dbgLefDefOutVersion
set dbgLefDefOutVersion 5.7
defOut -floorplan -netlist -routing fpga_nano.def
```

Technologies

We provides configuration files for the publicly available MOSIS technology `SCN6M_DEEP`.

- `./bench/etc/scn6m_deep_09.rds`, RDS rules for symbolic to real transformation.
- `./bench/etc/scn6m_deep.hsp`, transistor spice models for `yagle`.

References:

- MOSIS Scalable CMOS (SCMOS)
- MOSIS Wafer Acceptance Tests

Technical informations:

```
                               MOSIS WAFER ACCEPTANCE TESTS

          RUN: T92Y (MM_NON-EPI_THK-MTL)                      VENDOR: TSMC
   TECHNOLOGY: SCN018                                  FEATURE SIZE: 0.18 microns
                                      Run type: DED


INTRODUCTION: This report contains the lot average results obtained by MOSIS
              from measurements of MOSIS test structures on each wafer of
              this fabrication lot. SPICE parameters obtained from similar
              measurements on a selected wafer are also attached.

COMMENTS: DSCN6M018_TSMC


TRANSISTOR PARAMETERS     W/L       N-CHANNEL P-CHANNEL  UNITS

 MINIMUM                  0.27/0.18
  Vth                                    0.50      -0.49  volts

 SHORT                    20.0/0.18
  Idss                                 572      -276     uA/um
  Vth                                    0.52      -0.49  volts
  Vpt                                    4.7       -5.2   volts

 WIDE                     20.0/0.18
  Ids0                                  20.8      -15.2   pA/um

 LARGE                    50/50
  Vth                                    0.42      -0.41  volts
  Vjbkd                                  3.7       -4.4   volts
  Ijlk                                 &lt;50.0     &lt;50.0   pA

 K' (Uo*Cox/2)                         171.0     -37.0   uA/V^2
 Low-field Mobility                    406.07     87.86  cm^2/V*s

COMMENTS: Poly bias varies with design technology. To account for mask
           bias use the appropriate value for the parameters XL and XW
           in your SPICE model card.
                      Design Technology                  XL (um)  XW (um)
                      -----------------                  -------  ------
                      SCN6M_DEEP (lambda=0.09)            0.00    -0.01
                                  thick oxide             0.00    -0.01
                      SCN6M_SUBM (lambda=0.10)           -0.02     0.00
                                  thick oxide            -0.02     0.00


FOX TRANSISTORS           GATE      N+ACTIVE  P+ACTIVE  UNITS
 Vth                      Poly         &gt;6.6     &lt;-6.6   volts


PROCESS PARAMETERS      N+     P+    POLY  N+BLK  PLY+BLK     M1      M2    UNITS
 Sheet Resistance        7.0    8.1  8.3    59.5   306.6    0.08    0.08   ohms/sq
 Contact Resistance      8.3    8.8  8.1                            4.83   ohms
```

```
 Gate Oxide Thickness  41                                              angstrom

PROCESS PARAMETERS     M3   POLY_HRI     M4       M5        M6    N_W     UNITS
 Sheet Resistance     0.08              0.08    0.07     0.01    951     ohms/sq
 Contact Resistance   9.74             15.36   21.50    23.45           ohms

COMMENTS: BLK is silicide block.


CAPACITANCE PARAMETERS  N+   P+  POLY M1 M2 M3 M4 M5 M6 R_W  D_N_W  M5P N_W  UNIT
 Area (substrate)      969 1234  101  34 14  9  7  5  4        129       130 aF/u
 Area (N+active)                8517  53 20 14 11  9  8                      aF/u
 Area (P+active)                8275                                         aF/u
 Area (poly)                          64 17 10  7  5  4                      aF/u
 Area (metal1)                           35 14  9  6  5                      aF/u
 Area (metal2)                              36 14  9  6                      aF/u
 Area (metal3)                                 37 14  9                      aF/u
 Area (metal4)                                    36 14                      aF/u
 Area (metal5)                                       35            1039      aF/u
 Area (r well)         953                                                   aF/u
 Area (d well)                                          562                  aF/u
 Area (no well)        140                                                   aF/u
 Fringe (substrate)    196  229       53 36 29 24 21 19                      aF/u
 Fringe (poly)                        68 38 29 23 19 18                      aF/u
 Fringe (metal1)                         49 34    22 20                      aF/u
 Fringe (metal2)                            45 35 27 23                      aF/u
 Fringe (metal3)                               54 34 30                      aF/u
 Fringe (metal4)                                  63 43                      aF/u
 Fringe (metal5)                                     66                      aF/u

CIRCUIT PARAMETERS                            UNITS
 Inverters                     K
  Vinv                        1.0       0.74  volts
  Vinv                        1.5       0.79  volts
  Vol (100 uA)                2.0       0.08  volts
  Voh (100 uA)                2.0       1.62  volts
  Vinv                        2.0       0.83  volts
  Gain                        2.0     -24.67
 Ring Oscillator Freq.
  D1024_THK (31-stg,3.3V)             302.91  MHz
  DIV1024 (31-stg,1.8V)               377.13  MHz
 Ring Oscillator Power
  D1024_THK (31-stg,3.3V)               0.07  uW/MHz/gate
  DIV1024 (31-stg,1.8V)                 0.02  uW/MHz/gate

COMMENTS: DEEP_SUBMICRON
```