Creating a PDK that is Fully-Interoperable with Multiple EDA Tools from an Open Access PDK with Proprietary Code

Bill Ellersick
Analog Circuit Works
Microprocessor Project Summary

- Project: power efficient microprocessor
  - Integrated power management with 6 zones
  - Asynchronous logic to maximize efficiency
  - Two primary applications: data center servers and IoT
Process Choice and EDA Tool Flow

• Goal: single-vendor tool flow that provides best value
• Semiconductor process: 28nm CMOS
  – Foundry Open Access oaPDK used proprietary code
  – oaPDK had everything necessary but need more for full productivity
    • Slight modifications to be fully interoperable
  – Needed to create an Interoperable iPDK (www.iplnow.com)
EDA Challenges/Goals

• 2 Challenges/Goals to achieve full productivity with oaPDK
  – Enable source/drain area/perimeter calculations on schematics
    • Provides accurate pre-layout SPICE simulation results
    • Not necessary: post-layout sims use actual source/drain area/perimeter
  – Modify PyCells to automatically generate device layouts
    • Expedites transistor-level layout
    • Not necessary: design rule checking (DRC), layout vs. schem. (LVS)

• Why surmount this dual challenge?
  – To be able to choose the EDA flow that provides the best value
  – Because it is there (and in the way)!
Mapping Proprietary PDK Code to TCL

Combined Foundry PDK + Reference40nm iPDK

Proprietary Code in Foundry “Open Access” PDK
- Proprietary Callback Code
- Library and Cellview Parameters
- Proprietary PCELL Code

TCL Code in iPDK (Reference40nm)
- TCL Callback Code
- Library and Cellview Properties
- Python PyCELL Code

Foundry PDK
- Techfile (process layers), HSPICE models, Layout and Schematic Database Files, LVS, DRC, PE

μProcessor IP
- Standard Cell IP
oaPDK+iPDK Development

• Copied reference40nm iPDK files into oaPDK
  – Based on TCL and Python: supported by many EDA tools
• Modified reference40nm iPDK properties
• Minor PyCell and TCL code changes for 28nm process
• Key directories and files in combined iPDK+oaPDK:

  ipdk_oapdk_28nm/
  reference40nm_modified
  pdk-28-v1.1
  cdslib/5U1x_2U2x_1T8x_LB/oa
  autoLoad.file
  cmos32lp/
  techfiles
  scripts
    callbacks
      mos_reference40nm_modified_cbk.tcl
    common
  pdk-28-v1.1_SPICE_Model
NFET Symbol and Callback

**oaPDK symbol, params**
- Default W/L is OK, with dependent parameters

**iPDK symbol, params**
- Dependent params match oaPDK for default W/L

(the 28nm process has a 30nm drawn gate-length)
oaPDK Callback Fails, iPDK works

- oaPDK: Change in W causes error, no change in dependent params
- iPDK: Change in W updates dependent parameters correctly
The Netlist is What Matters

• Schematics are only a means to a netlist (productivity tool)
  – iPDK and oaPDK netlists match for default parameters
    • Drain/source area/perimeter are estimates to guide design process
  – Parasitic extracted (PEX) netlist true test of transistor-level design
    • PEX is fully supported by oaPDK, can use to verify callback as, ad, etc.
• Modify iPDK symbol parameters: HSPICE model, defaults

• oaPDK netlist (only correct for 80/30n default W/L):
  \[
  \text{xm123 s g d b nfet w=80n l=30n nf=1 m=1 as=6.08f ad=6.08f ps=312n pd=312n}
  \]

• iPDK netlist (parameters scale properly with W, L):
  \[
  \text{xn121 s g d b nfet w=80n l=30n m=1 nf=1 as=6.08f ad=6.08f ps=312n pd=312n}
  \]
  \[
  \text{xm114 s g d b nfet w=0.2u l=30n nf=1 m=1 as=15.2f ad=15.2f ps=552n pd=552n}
  \]
Modifying iPDK Cellview Properties

• **Modify propBagCreateIPL.tcl to conform to 28nm rules:**
  
  ```tcl
  set mos_props [model minLength maxLength minWidth maxWidth polyDiffEndcap mosType]
  set mos_values [list [list nfet 0.03 20 0.08 50 0.10 nmos]
  oaStringPropCreateWrapper $dmd contMinWidth 0.04
  ```

• Or, use script (propBag.tcl) to modify properties with GUI
  – Creates new menu choice when right-click on cellview

  ![Property Bag for reference40nm nfet_b](image)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Value</th>
<th>AppProp Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>minWidth</td>
<td>StringProp</td>
<td>0.08</td>
<td>nil</td>
</tr>
<tr>
<td>maxLength</td>
<td>StringProp</td>
<td>20</td>
<td>nil</td>
</tr>
<tr>
<td>minLength</td>
<td>StringProp</td>
<td>0.03</td>
<td>nil</td>
</tr>
</tbody>
</table>

• **Edit parameter definitions on foundry PDK cellviews**
  – Displays parameters used by proprietary code

  ![Edit Parameter Definitions](image)
Modifying iPDK Technology Properties

• Read relevant code to determine what to change
  – FET callback code from mos_reference40nm_cbk.tcl:
    set sWidth [expr $tech(contMinWidth)+$tech(diffContEnclose)+$keys(polyContSpacing)]
    # if { ([lsearch $len_lis [expr $len*1e6]] == -1) … (commented out discrete length list)

• Modify technology file properties
  – Technology Manager -> Export, modify exported file, then Import
    cadGrid 0.001

• Edit parameter definitions on foundry PDK library
  – Displays parameters used by proprietary code
  – Translate these to corresponding technology file properties
iPDK PyCells, Modified Properties

- iPDK PyCell’s respond to W/L and other changes
  - Parameters modified to match 28nm PDK and pass LVS, DRC
PyCell Layer Modifications

- Simple modification of reference40nm mosfets.py PyCells
  - Guided by layout views of reference40nm and 28nm cells
  - Verified with DRC, LVS decks from foundry PDK

```python
layerMapping = dict(
    poly       = ( "PC", "drawing"),
    cont       = ( "CA", "drawing"),
    metal1     = ( "M1", "drawing"),
    #poly      = ( "poly", "drawing"),
    # cont     = ( "cont", "drawing"),
    # metal1   = ( "m1", "drawing"),
)
```

- Need to compile PyCells for changes to take effect
Boutique Devices

- Finger caps, MiMcaps, ESD devices, LDFETs, VPNP, etc.
- Schematic callbacks are trivial or similar to core FET’s
- Simple netlist with W, L is often all that is needed
- oaPDK provides layout for all devices with default sizing

**Finger cap**
Stretch layout to resize. Simple netlist, callback:
xc1 t b apmom l='Lc' w='Wc' botlev=1 toplev=6

**High voltage Lightly Doped Drain FET (LDFET)**
Stretch layout to resize. Netlist and callback like nfet:
xn108 s g d b ldegnfet_4p5 w=1.6u l=180n nf=2 m=1 + as=256f ad=128f ps=3.84u pd=1.92u
Quality Assurance, LVS, DRC, PEX

- Fully Supported LVS, DRC, PEX mitigate risk
- Schematic callbacks and layout Pcells are really just productivity tools, not crucial PDK features

<table>
<thead>
<tr>
<th>Risk</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>as/ad calculated incorrectly in modified callback</td>
<td>post-layout parasitic extraction (PEX) doesn’t use callback</td>
</tr>
<tr>
<td>PyCell layout has illegal contact to gate space</td>
<td>design rule checking (DRC) will flag</td>
</tr>
<tr>
<td>PyCell layout generates incorrect device width/length</td>
<td>Layout vs. Schematic checking (LVS) will flag mismatch</td>
</tr>
</tbody>
</table>
Struggles

• Struggles
  – PDK enablement 1 person-month, including learning curve
    • PDK enablement with this presentation as guide: 1 week
  – Transistor-level EDA tool learning curve: 0.5 months
    • Similar to other EDA tools, but still need to look up commands/syntax
  – EDA tool errors without indications of cause or source
    • Example: “string index out of range”
    • Would like a way to turn on source-code debugging
Tips (i.e. Struggles Overcome)

- **TCL error tracing**
  - To get more information on errors such as “Unknown Error” (open layout with a layer that is not in the technology file)
    Unknown error
    Designer> puts $errorInfo
    can't read "iPDK_isLayout": no such variable while executing "set iPDK_isLayout" ("uplevel" body line 1) invoked from within "uplevel #0 set $data"
  - Also: increase stop level, refresh Layout view to see PyCell error
  - And: look in terminal window where EDA tool was launched
I Wish EDA Tool Industry Would …

- Partner with more foundries to develop interoperable PDKs
  - A few people could do many processes/month
  - Continue to push fully interoperable standards: users demand this
- Develop EDA tools to rival e.g. C++ design environments
  - Transistor level EDA tools lag those for larger markets (e.g. C++)
  - Source code debugging, signal to driver/load click/view
- Provide more warning/error tracing information
  - File names and line numbers really help to identify issues
- Make all PDK properties/parameters editable with menus or scripts that are included with EDA tools
Successes, Conclusions

• Successes
  – Schematic simulation easy to enable
  – PyCell enablement more difficult, but much was learning curve

• Conclusions
  – Many EDA tools for mixed signal design on oaPDK processes OK
    • Some start-up effort required
    • Risks largely mitigated by fully supported LVS, DRC and PEX
    • With ~1 person-week of effort, can enable a non-interoperable oaPDK