SCHEMATIC PCELLS, FUTURE OF DEEP SUBMICRON CUSTOM IC DESIGN

PRANAV BHUSHAN
CADENCE DESIGN SYSTEMS INC.
+91-120-2562842
pbhushan@cadence.com

RAJ ARUMUGAM
QUALCOMM INC.
+1-858-845-7877
rarumuga@qualcomm.com

CDN LIVE! SILICON VALLEY
September 12-15, 2005
Santa Clara, CA

ABSTRACT

As we make inroads into Deep Sub Micron (DSM) processes, digital and analog designers alike see a clear need for accurate pre-layout modeling of second order effects for front-end simulations.

In this paper, we explore and improve upon the capabilities offered by schematic pcells to achieve more accurate representation of the physical effects of primitive devices in the netlist without the need for cumbersome netlist post-processing. In DSM, the physical effects like Shallow Trench Isolation (STI) and Nwell proximity effects play a significant role and needs to be accounted for during front-end simulations to closely match post-layout simulation results. Due to the limitations of the spice simulators, it becomes necessary that multi-fingered mos devices are represented as individual discrete transistors in the netlist to accurately estimate some of these effects.

In this paper, we demonstrate how one could take advantage of the pcell concept and virtual schematics to build custom logic cells and control how they are represented in the netlist. For a logic gate like a nand gate, one could also have the primitive transistors in a folded or stacked configuration yielding different parasitics. In our proof of concept library, based on user input for fingers, shared configuration and other parameters of interest, and with the help of a intelligent pcell code, we create schematics in virtual memory on the fly with the required connectivity information. Each finger is represented as a discrete transistor in the netlist with the associated parasitics. The current approach can also be enhanced to build complex schematics to illustrate the default structure of the gate with explicit wires. The OSS (Open Simulation System) would reference the corresponding schematic in virtual memory when it hits an instance of a gate built as a schematic pcell during netlisting. There are also advantages to building top level transistors (tnmos or tpmos) using schematic pcells which in turn reference the primitive nmos or pmos devices built based on foundry specifications. When foundry changes their design specification which affects the way parasitics are calculated, legacy designs using nmos or pmos would have to go through a translation to update any inter-dependent instance parameters. If one uses tnmos or tpmos instead, since the underlying schematic is created on the fly every time a netlist is generated, the legacy designs are now immune to any changes that affect primitive nmos or pmos. We believe that the schematic pcell approach is superior to the conventional wisdom to write netlist post-processors to address such issues since this solution is very clean and requires very little maintenance. Also, one could extend this approach to add inter-node capacitances and resistances at shared nodes in multi-fingered devices.

Using schematic pcells, one could come up with highly customizable cells that offer tremendous flexibility to analog designers with accurate modeling of layout parasitics at the front-end. Some future work related to this field is also described in the paper.

Keywords: SKILL, analogLib, pcell, CDF, virtual schematics

1. INTRODUCTION

In deep sub-micron design processes, there is a need to accurately model second order pre-layout effects during front-end simulations. The physical effects like Shallow Trench Isolation (STI) and Nwell proximity [5] play a significant role and needs to be accounted for during front-end simulations to closely match post- layout simulation results.

Firstly, to extract maximum accuracy out of the spice models, it becomes necessary that multi-fingered mos devices are represented as individual discrete transistors in the netlist to accurately estimate some of these effects. Secondly, the layout extraction tools extract multi-finger devices as individual fingers and to get a one-to-one correspondence during layout vs. schematic it becomes necessary to represent these devices as discrete transistors in the schematic netlist. A solution to these problems lie in building a highly flexible schematic architecture that ties to the OSS (Open Simulation System) netlisting flow. Schematic parameterized components (pcells) widely supported in front-end simulation flow offers the best of both worlds.

In this paper, we explore and improve upon the capabilities offered by schematic pcells to achieve more accurate representation of the physical effects of primitive devices in the netlist without the need for cumbersome netlist post-processing.

2. HISTORY AND BACKGROUND

With shrinking device sizes, second order effects like STI and Nwell proximity [5] play a significant role. Shallow trench isolation models the mechanical stress experienced by a CMOS device due to shallow trenches of oxide used to isolate neighboring devices. Depending on the distance from the trench to the poly gate the mobility of NMOS increases and PMOS decreases. STI also has an effect on the threshold voltage, with V_t of NMOS increasing with trench distance while that of PMOS decreases. The most significant impact is on the drain current with degradation for NMOS and enhancement for PMOS [5]. The BSIM4 and to some extent BSIM3V3 versions model these layout physical effects for spice circuit simulation. These are usually instance parameters of the subcircuit. For a multi-finger device, representing these parameters in the netlist is not straight forward and leads to grossly inaccurate estimates if all the fingers and associated parasitics are lumped as a single device. Each finger in a multi-finger device will experience different effects due to STI because the gate to oxide trench distance is different for each finger based on how much neighboring diffusion is available. To represent this accurately, one would have to break these fingers to discrete devices in the netlist. Alternately, one could pass on the finger information to the simulator and BSIM4 models would average out the STI effect of different fingers and use the effective SA and SB to calculate the current. But the accurate method would be to represent each finger with its associated STI parameters (SA and SB) as discrete devices in the netlist. For Nwell proximity, the BSIM4 models use SC effective parameter to model the effect. Since SC effective is a function of the device gate to the well edges, it is a function of the STI parameters SA and SB. Again, this would require the fingers to be broken into discrete devices for these effects to be modeled accurately.

To meet traditional front end designs involve creation of schematics for a given design specification and constraints. Typically a custom core would have hundreds of logic gates, also known as, nand, nor, inverters, xor etc. Each of these gates could potentially be instantiated with different configuration of inputs, number of fingers of primitive transistors per input, and the devices in a pull down or pull up chain either stacked or folded. This would mean that the schematic designer would be building a hierarchical schematic for each one of these gates and their variants. Using schematic pcells one could build these gates such that they are programmable and thus provide great flexibility to the schematic designer. It also improves the productivity of the designer because one need not create a schematic corresponding to each variation of the schematic. They also offer other advantages and flexibility in terms of on the fly customization, compact storage, efficient disk utilization and creation of schematic designs due to variants of a single cell in memory rather than on disk. Some of the examples and advantages of using schematic pcell design are discussed and described in [4].

Application wise, pcells can be used for multi-conductor components for example, transmission lines with a varying number of terminals (Virtuoso® analogLib/mtline pcell) or a component where the set of inputs and outputs can vary based on the type, for example IBIS component (Virtuoso® analogLib/ibis_buffer pcell). In deep submicron realm, the design scenario can be more challenging as second order physical effects come into picture which dictate the design style and layout architecture and hence pose difficulties in modeling these for pre-layout simulations. One popular approach among circuit designers is to finger devices as much as possible to share diffusions, not only to reduce diffusion capacitors, but also to reduce the effects of STI by taking advantage of neighboring diffusions. If the designer has to observe the benefits attained by breaking a large device into smaller fingers, the pre-layout simulation environment should be capable of reflecting the actual layout style. This pretty much boils down to how such a device is represented in the actual spice netlist. To achieve this parity between the layout and schematic design we use schematic pcell flow to our advantage. We illustrate how a single pcell component instantiated in a schematic can be represented with different subcircuit definitions in the spice netlist based on the component parameters using an intelligent pcell code. We term the underlying schematic created by such an intelligent pcell code which eventually gets translated to the guts of the subcircuit in terms of textual connectivity (netlist) as a '*virtual schematic'*.

The current approach delves further into the schematic pcell design and describes another category of pcells where the shape, size, orientation, number of pins etc. of the schematic component remains same but there is a variation in the generated netlist. We describe the creation and use of such a new component as a solution where the connectivity or the ASCII netlist information of the design is a function of the pcell formal parameters. Such a solution can easily be extended to cater similar design needs in deep sub-micron Custom IC design.

3. PROBLEM DESCRIPTION

In DSM, it has become increasingly important to model the physical layout effects ahead of time for the circuit designers to get a better understanding of how their circuit will perform once implemented in the actual silicon. We address the key issue of multi-fingered devices and how one could achieve some consistency in the way they are represented in the schematic and layout domain. In a simplistic sense, our goal is to have an inverter sub-circuit instance with user inputs for number of fingers such that the designer could control how the underlying NMOS and PMOS devices would be fingered. Each finger would then be represented as a discrete transistor in the netlist with associated parasitics. For a NAND or NOR like gates, one could get more fancy and give the user control to specify whether the different fingers of each input would be stacked or folded in the lower level subcircuit.

4. DESIGN DETAILS

In our proof of concept library we designed an intelligent schematic pcell subcircuit component where based on the number of fingers, shared configuration and other parameters of interest, we create schematics in virtual memory on the fly with the required connectivity information. Schematic pcell implementation to meet such a design requirement was the best choice. We designed our pcell to work in a way that the subcircuit instantiation behaved like a black box where the main pcell code was embedded.

We term such schematic designs as '*virtual schematic'* because there are no visible variations shown in the schematic on the pcell characteristics but the variation is in the netlist or connectivity of such a special subcircuit pcell based on the number of fingers. Each finger is represented as a discrete transistor in the netlist with the associated parasitics. For example, incase the number of fingers on the subcircuit pcell are two, then the corresponding netlist would contain a subcircuit instance with the description of two virtual transistors describing the actual design. In reality such a design would contain a pcell super-master (of the parent subcircuit cell) on the disk, its sub-master in the virtual memory and the variant of instance's sub-master. Similarly more such subcircuits can be customized and our cell be used in a similar manner. The implicit connections of the individual mos components in our design were created through the database calls. One could also build pretty schematics to illustrate the default structure of the gate with explicit wires.

In the subsequent sections we describe some details on the features of our special subcircuit pcell component.

4.1 Pcell Structure

Our pcell solution was similar to any other traditional schematic pcell but the noticeable difference here was that our pcell did not vary in shape, size (shrinks or grows), changes in number of pins or orientation etc. based on its formal parameters. This is the biggest difference in our pcell as compared to any traditional schematic pcell symbol. The only varying part was the variation in the generated netlist or connectivity information of the subcircuit component which was controlled based on formal parameter of the pcell.

The main point to note here is that our pcell code is more intelligent as compared to traditional schematic pcells because here we are, instantiating the standard nmos or pmos components inside our sub-circuit via explicit calls to database APIs, and later based on the formal parameter – NF (Number of Fingers) variations we were creating and adding internal connectivity information to the pcell structure. There by adding internal connectivity to the block terminals or pins not visible at top level, hence creating virtual or pseudo schematic designs.

4.1.1 Subcircuit Symbol

The actual pcell component is generated by the SKILL code which is described in [Sec 4.1.3]. The shape of the top level block can be user defined but we preferred it to be a simple rectangle which is also the shape of a symbol automatically generated in Virtuoso® composer environment.

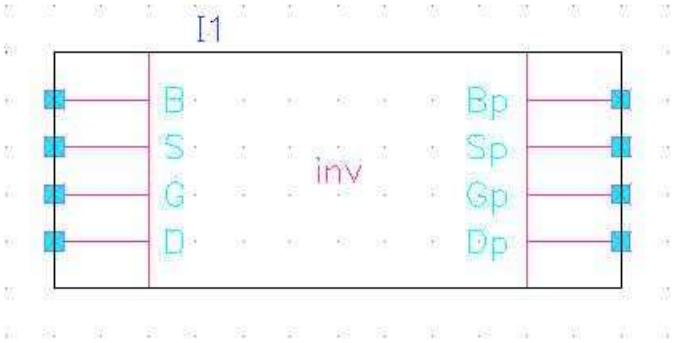

Figure 1 (a): Typical rectangle shape of subcircuit pcell (‘inv’) inverter component

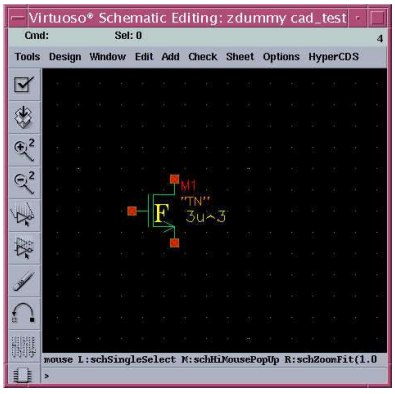

Figure 1 (b): Sample transistor like shape of the subcircuit pcell component

The above subcircuit component can be instantiated in the schematic and used to generate netlist for this special pcell. This is shown above in Figure [1a], an instance I1 of our special pcell ‘inv’, used to generate virtual schematics and user defined netlist variations. As already indicated the shape can be user

defined so for the same example in typical scenario designer would prefer a transistor like or a inverter shape for such a component as shown in Figure [1b].

Once a block is specified for the inverter shape all the details inside the subcircuit component are hidden from user. It should be noted that the subcircuit component 'inv' cell is itself a symbol of a schematic which consists of transistors (nmos4 and pmos4 of Virtuoso® analogLib). If you descend into the inv cell you can see the virtual circuitry of the inverter created by the special pcell code but, note that variations in terms of connectivity or more number of devices will not be visible because in reality this represents a '*virtual schematic*' of the subcircuit.

4.1.2 Database APIs

The pcell writing guidelines recommend a set of APIs which can safely be used in any pcell code; these are described in more detail in Virtuoso® pcell reference [3]. Similar to any other schematic pcell our pcell code also consists of a set of db APIs but, some of them which made our schematic pcell solution distinct from other pcells are shown and marked in Table [1].

S.No	API Name
1	*dbOpenCellViewByType*
2	*dbCreateNet*
3	*dbCreateTerm*
4	*dbCreatePin*
5	*dbCreateInst*
6	*dbCreateParamInst*
7	*dbCreateConnByName*
8	*dbAddFigToNet*
9	*dbDisablePropTimeStamp*
10	*dbCreateProp*
11	*dbReplaceProp*

Table1: List of Database APIs used in our schematic pcell design

We would explicitly like to mention different CDBA APIs [1] used in our pcell code like; dbCreateInst, dbCreateParamInst, dbCreateConnByName, dbAddFigToNet which were used in our pcell code to add virtual devices and establish connectivity and link between the virtual transistors added to our pcell component on the fly. These database APIs were present in our pcell body and based on any change in the formal parameter NF [Sec 4.2] our pcell would intelligently establish connections and take care of connectivity. It should be important to note that these highlighted APIs are typically not used in traditional schematic pcells.

4.1.3 Pseudo Code

Programming of a pcell is the most important and critical part in its design. The pcell SKILL code is the IP of the design which plays a critical role in its behavior and implementation. It should be noted that for the initial generation of symbol's pins, terminals, nets etc. the algorithm is similar to any traditional pcell design but the only difference here is the automatic connectivity generation and auto-addition of transistor instances creating virtual schematics.

The pseudo code for our special subcircuit pcell is described below in terms of the three sections of a pcell code.

1. *Cell Identifier Section:*
   Set pcDefinePcell and define all cell identification parameters.
2. *Param Declaration Section:*
   Set defaults for formal parameter, NF (Number of fingers) = 1

3. *Pcell Body*
    - *Store instances of standard nmos4 or pmos4 components from Virtuoso® analogLib in memory. This is done so as to avoid time in creating individual mos transistors geometry because these are readily available in standard libraries. We will add subsequent transistors based on the value of NF.*
    - *Add instances of 'iopin' from Virtuoso® basic library.*
    - *Specify each of these iopins as drain(D), gate(G), bulk(B) and source(S) for the nmos and pmos transistors. This is done because these virtual pins will be used to establish connectivity information between transistors.*
    - *For each I = 1 to NF (where, NF represents the number of virtual transistors in the inverter subcircuit)*
        i. *Specify values of the CDF parameters of nmos4 and pmos4 devices e.g. length, width, substrate area etc.*
        ii. *Add and create parameterized instances of nmos4 and pmos4 transistors.*
        iii. *Specify the connectivity of nmos4 and pmos4 transistors with their pins and terminals with the virtual pins (D,G,B,S) specified and created above.*
        iv. *Repeat the same for other values of formal parameter - NF.*
    - *Add other required cellview properties to the subcircuit pcell component e.g.lastSchematicExtraction etc.*

It should be noted that pcell super-master is created using the pseudo code for the subcircuit instance. Based on the values of NF for each sub-master its variants are created in virtual memory. The connectivity of these transistors is specified by the special pcell code.

4.2 CDF Parameter support

The special pcell component had to provide support for a CDF component [2] which was the pcell formal parameter. This parameter was called as, NF – Number of Fingers and it was attached to the symbol of the subcircuit's CDF. This is shown in Figure [2].

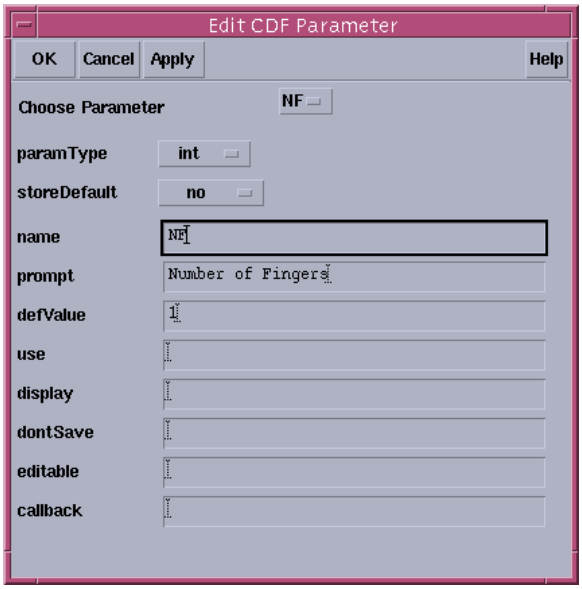

Figure 2: Formal parameter NF defined in CDF of the 'inv' pcell component

Figure 3: NF parameter shown in the Edit Object Properties Form

The variations in parameter, NF were recognized by the pcell code which causes the generation or addition of virtual transistors to the schematic. Based on variations in number of fingers on the subcircuit instance transistors vary which is reflected in the netlist as a virtual schematic. The NF parameter on the Edit Object Properties Form in Virtuoso® Composer is shown in Figure [3].

4.3 Netlisting support

No special netlisting support was required in our case because the generated pcell component was a subcircuit and it was netlisted using the default OSS flow. Being a subcircuit, OSS traversed through the schematic and netlisted all the components added inside the virtual pcell block. The instance was automatically interpreted by OSS and hence the final netlist was generated. This was a major advantage in our design as the default netlisting by OSS yielded desired results for different formatters. For example, the same instance would netlist for Spectre direct, hspice direct etc. simulation interfaces per our design requirements. Netlist examples Spectre direct (spectre) and Hspice direct (hspiceD) are shown below.

```
// Generated for: spectre
// Generated on: Aug 12 14:17:59 2005
// Design library name: test_lib
// Design cell name: top
// Design view name: schematic
simulator lang=spectre
global 0

// Library name: test_lib
// Cell name: inv
// View name: schematic
subckt inv_pcell2460 B Bp D Dp G Gp S Sp
    MN4 (D G S B) nmos4 w=2u l=10u sa=0.13u sb=0.13u
    MN3 (D G S B) nmos4 w=2u l=10u sa=0.13u sb=0.13u
    MN2 (D G S B) nmos4 w=2u l=10u sa=0.13u sb=0.13u
    MN1 (D G S B) nmos4 w=2u l=10u sa=0.13u sb=0.13u
    MP4 (Dp Gp Sp Bp) pmos4 w=5u l=10u sa=0.13u sb=0.13u
    MP3 (Dp Gp Sp Bp) pmos4 w=5u l=10u sa=0.13u sb=0.13u
    MP2 (Dp Gp Sp Bp) pmos4 w=5u l=10u sa=0.13u sb=0.13u
    MP1 (Dp Gp Sp Bp) pmos4 w=5u l=10u sa=0.13u sb=0.13u
ends inv_pcell2460
// End of subcircuit definition.

// Library name: test_lib
// Cell name: top
// View name: schematic
I1 (net4 net8 net1 net5 net2 net6 net3 net7) inv_pcell2460
```

Figure 4: Spectre Direct Netlist of 'inv' subcircuit pcell, with NF=4

```
** Generated for: hspiceD
** Generated on: Aug 12 14:19:38 2005
** Design library name: test_lib
** Design cell name: top
** Design view name: schematic

.TEMP 25
.OPTION
+    ARTIST=2
+    INGOLD=2
+    PARHIER=LOCAL
+    PSF=2

** Library name: test_lib
** Cell name: inv
** View name: schematic
.subckt inv_pcell2460 b bp d dp g gp s sp
mn3 d g s b nmos L=10e-6 W=2e-6
mn2 d g s b nmos L=10e-6 W=2e-6
mn1 d g s b nmos L=10e-6 W=2e-6
mp3 dp gp sp bp nmos L=10e-6 W=5e-6
mp2 dp gp sp bp nmos L=10e-6 W=5e-6
mp1 dp gp sp bp nmos L=10e-6 W=5e-6
.ends inv_pcell2460
** End of subcircuit definition.

** Library name: test_lib
** Cell name: top
** View name: schematic
xi1 net4 net8 net1 net5 net2 net6 net3 net7 inv_pcell2460
.END
```

Figure 5: Hspice Direct Netlist of 'inv' subcircuit pcell, with NF=3

Figure [4] and [5] show the generated netlist for the inverter subcircuit (inv_pcell2460) in our case. Here, inv_pcell2460 is a random subcircuit name attached by OSS during netlisting. This can be customized per the requirements please refer to [Sec 6] for more details and future work on this.

Note that Figure [4] shows a spectre direct netlist with NF=4 where, I1 is the instance of the pcell subcircuit and the definition contains four nmos4 and pmos4 transistors. Similarly Figure [5] shows the hspiceD netlist with NF=3, note that this netlist contains three nmos and pmos components for three fingers.

5. RESULTS

The motivation for undertaking this effort was to understand how best one could estimate and model the second order effects in deep sub micron processes. It is very important for the circuit designers, particularly analog designers, that the pre-layout simulations closely match that of the post-layout simulation results after extraction. The first step towards achieving this was to break the multi-finger devices to discrete fingers in the netlist.

Via our approach we have illustrated the difference between the netlist of a conventional lumped MOS device with the one which takes advantage of the schematic pcells. For example lets consider a mosfet device with three fingers (NF=3), as shown in Figure [6]. There could be two cases, one in which we interpret the transistor as a conventional lumped MOS device represented as Case I and, second in which we interpret and realize the device using our schematic pcell approach, Case II.

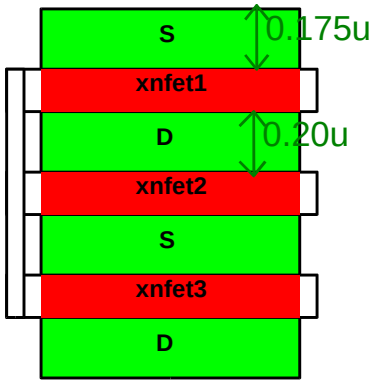

Figure 6: MOSFET Transistor shown with 3 fingers (NF=3)

Case I: The netlist of a NMOS transistor with 3 fingers when lumped

```
xm0 net6 net4 net5 0 nfet l=60e-9 w=3e-6 ad=375e-15 as=375e-15
pd=4.75e-6 ps=4.75e-6 nrd=33.3333e-3 nrs=33.3333e-3 nf=3 sa=175e-9
sb=175e-9 sd=200e-9
```

Case II: The netlist of the same NMOS transistor with 3 fingers created using schematic subcircuit pcells:

```
.subckt inv_pcell2460 d g s inh_vss_sub_tie

xmn_3 d g s inh_vss_sub_tie nfet l=60e-9 w=1e-6 ad=175e-15 as=100e-15
pd=2.35e-6 ps=1.2e-6 nrd=100e-3 nrs=100e-3 nf=1 sa=695e-9 sb=175e-9

xmn_2 d g s inh_vss_sub_tie nfet l=60e-9 w=1e-6 ad=100e-15 as=100e-15
pd=1.2e-6 ps=1.2e-6 nrd=100e-3 nrs=100e-3 nf=1 sa=435e-9 sb=435e-9

xmn_1 d g s inh_vss_sub_tie nfet l=60e-9 w=1e-6 ad=100e-15 as=175e-15
pd=1.2e-6 ps=2.35e-6 nrd=100e-3 nrs=100e-3 nf=1 sa=175e-9 sb=695e-9

.ends inv_pcell2460
```

With BSIM4 models, one could pass the distance between the fingers as one of the parameters (SD) to the simulator. The Hspice simulator would then estimate the total LOD (Length of Oxide Definition) effect on the device based on the average LOD per finger. This is illustrated in Case I. The end devices do not have the advantage of neighboring diffusions like the inner fingers do and averaging out the LOD does not truly reflect the actual layout of the device. Also, in a real design, one could have the different fingers separated by different distances. That is the reason Case II, is a more accurate representation of the device shown above and has a one to one correspondence to how the extraction tools will treat this device. It should be noted that the netlist in Case II, xmn_1, xmn_2 and xmn_3 represent the individual devices in terms of number of fingers representing a more accurate design.

Similar arguments hold true when one attempts to model the Nwell proximity effect for front end spice simulations.

6. LIMITATIONS, CHALLENGES AND FUTURE WORK

Schematic pcell design requires a set of safety rules to be followed to write pcell code, mentioned in [3]. The user needs to follow these precautions and use the recommended list of functions to be used in a pcell code. In the current solution the major challenge was the intelligent design of pcell code which required several iterations of re-writing pcell code to establish connectivity and addition of transistors to the schematic pcell subcircuit.

One possible drawback of this pcell approach was in the inability of ADE (Virtuoso® Analog Design Environment) front-end netlister to create (write) unique data mapping information for each subcircuit variant, each time a fresh netlist-create (note, netlist-recreate does not causes this) was done. This was because each time user made an explicit call to netlist-create option in ADE, OSS assigned a new random pcell instance name and based on the new subcircuit name netlister created additional mapping data. This caused an increase in netlisting time and also writing of extra mapping information files for the virtual instance name in netlist. Since the pcell was evaluated every time the device was netlisted, one had to pay for the one time latency during simulation. It should be noted that the above issue is not reported incase user clicks on netlist re-create option from ADE because OSS only assigned a new subcircuit instance name to for pcell instance when a user wanted to generate a fresh netlist every time.

Another limitation in this flow was because OSS assigned a random suffix to the subcircuit name in the netlist. A solution was devised by OSS to customize the subcircuit name by specifying a global prefix via a OSS variable *'simPcellPrefix'* in the *.simrc* file. But, this assignment was global as all the subcircuits got the same prefix. Ideally, the designers would like to customize the subcircuit names such that one could look at a NAND gate in the netlist and figure out if it is part of the PLL block or the DSP. At this time, this was not possible due to the lack of a library/cell/instance based *simPcellPrefix.*
Product change requests have been filed to track these two issues.

Apart from the limitations, current schematic subcircuit pcell are very useful when building logic gates like NAND and NOR. The designer would be able to specify underlying structure of the primitive devices that make up the gate for example, like the number of fingers of primitive NMOS and PMOS devices, whether the fingers are stacked or folded and other options based on how fancy one is willing to add with in the pcell code. The schematic pcell flow in conjunction with OSS could be a powerful tool in the hands of the CAD developer who understands the circuit design needs better. This approach gives one complete control over how an object is represented in the netlist.

In future, this approach can be used to add device specific interconnect parasitic information to the netlist. For example, the pull down of a NAND gate with 3 inputs has two intermediate nodes. Each of these nodes has a interconnect capacitance associated with it based on the number of connections at a given node. Since one has complete access to the device and all its terminals being netlisted when built as a schematic pcell, one could introduce the capacitance with appropriate connectivity information in the netlist. This could be extended to add input and output capacitance associated with the device. This allows one to model most of the interconnect parasitics of the device in a self contained manner rather than using explicit wire models. Even if one uses sophisticated wire models with the conventional approach, it becomes almost impossible to account for the intermediate nodes in a gate, which is described as a biggest advantage of current approach.

7. CONCLUSION

We have described a novel way to create a schematic subcircuit pcell which actually brings in a new concept of *'virtual schematics'* where the schematic pcell does not changes in size, shape or dimension but, varies the connectivity information (netlist) based on pcell formal parameter. We believe this approach can further be extended to include inter node resistances and capacitances at shared nodes in multi-fingered devices and get rid of extra netlist post-processing done in many cases. Some of the challenges and future work based on this approach have also been discussed.

IMP: The core pcell code was developed at Cadence Design Systems, Inc. and was further modified and customized per requirements at Qualcomm Inc. Currently this solution is a part of Qualcomm's CIC flow.

8. REFERENCES

[1]. Integrator s Toolkit: Database Reference Manual, Product Version 5.0
http://voyager.cadence.com/techpubs/ic50/itkdbase/itkdbaseTOC.html

[2]. Component Description Format User Guide, Product Version 5.0
http://voyager.cadence.com/techpubs/ic50/cdfuser/cdfuserTOC.html

[3]. Virtuoso® Parameterized Cell Reference, Product Version 5.0
http://voyager.cadence.com/techpubs/ic50/pcellref/pcellrefTOC.html

[4]. Schematic pcell implementation in Virtuoso® platform, Pranav Bhushan, Raja Mitra
International Cadence User Group Conference (ICUG - 2004), Santa Clara, CA.
http://www.cadenceusers.org/cgi-bin/viewAbstract.cgi?abstractName=PranavBhushan1.html

[5]. BSIM4 Modeling and Parameter Extraction, Joachim Assenmacher, Infineon Technologies AG, 2003
http://mikro.ee.tu-berlin.de/ifm/AW/HandOuts/assenmacher.pdf