SCHEMATIC PCELL IMPLEMENTATION IN VIRTUOSO® PLATFORM

PRANAV BHUSHAN

CADENCE DESIGN SYSTEMS

+91-120-2562842

pbhushan@cadence.com

RAJA MITRA

rmitra@cadence.com

INTERNATIONAL CADENCE USERS GROUP CONFERENCE

September 13-15, 2004

Santa Clara, CA

ABSTRACT

Analog designers need compact and efficient libraries in custom IC design. This paper describes an approach to reduce the number of physical variants of a library component using schematic parameterized cells (pcells) in Virtuoso® Custom Design environment. Pcell technology makes the underlying library more compact, versatile and dynamic with less number of variants for a single component. We describe pcell design using Cadence proprietary language (SKILL) and discuss some of the challenges faced by CAD engineers in its implementation along with solutions.

To elucidate more on the challenges some well known pcell implementations like mtline, nport, scasubckt, ibis_buffer components etc. in analog library (analogLib) are considered. The repetitive challenges in pcell implementation along with future enhancements to make it more robust are discussed.

Keywords: SKILL, analogLib, CDF, pcell, simInfo

1 INTRODUCTION

Virtuoso® Platform contains an analog design components library known as analogLib, which contains different components that are widely used by analog designers in a variety of full custom analog and RF IC design. Analog library (analogLib) contains analog primitives, subcircuits of a wide variety ranging from - active devices, analysis devices, passive elements, parasitics and sources (dependent, independent etc.).

A parameterized cell, or pcell, is a graphic, programmable cell that allows to create a customized instance each time it is placed or used in design. The term pcell can mostly be related in physical design space where it is used to create layout views. A schematic pcell is a variant of a layout pcell, which can be used in schematic designs. Schematic pcells are used in front-end arena where they can simply instantiated in Virtuoso® Composer (Schematic Design Entry Tool) window and easily netlisted and simulated

In the following section, motivation for schematic pcell creation is discussed in detail. This will be followed by extensive design level details for implementing pcell. Lastly the challenges and future work related with pcell is discussed and improvements suggested.

2 MOTIVATION AND HISTORY

Traditional analog library components do not support variants of a single component. In order to support all variants of a component for example for a multi-port device (nport) device, different variants for each port should be provided. These variants can have different types, starting from single port (n1port) device, to two ports (n2port) device, three ports (n3port) device, four ports (n4port) etc. and so on. As a consequence, this makes the underlying library to contain four components for a nport (assuming we are providing support for 4 variants) device namely n1port, n2port, n3port and n4port. This makes the library more bulky in terms of managing more number of components, more disk space utilization in terms of storing different variants and more maintenance overhead in terms of revision control of all variants.

With the growing demand of multi port/conductor/terminal components and wide variety of design pins, terminals and shape requirements there was always a need for a single customizable component that could easily be modified and used per designer’s need. The component should be versatile, working seamless in a design having a wide variety of parameters and shape and should be easy to implement and use. With such a request pcell came out to be the best choice. Other well known requirements for pcell components are: multi-conductor transmission line (mtline) having a large number of terminals, often required and used in RF and mixed-signal designs and the IBIS (I/O Buffer Information Specification) buffer component having variation in its shape and pins based on the type and mode of the buffer selected. Such components have a common requirement i.e. a change in pins, shape, structure etc., and pcell seems to be an appropriate choice because it provides advantages in terms of making a library more compact, less bulky, more efficient and free from the overheads of maintaining and storing different symbols for a single component.

Pcells date back to years when layout designers used them to adjust chip geometries and combinations in chip design. Schematic pcells recently came into picture when people felt the need to extend the same functionality from layout to schematic domain. Having a potential solution in terms of a schematic pcell for variants, the next challenge in its design is the appropriate netlisting support. In early 2001 schematic pcells came to be supported by Virtuoso® OSS (Open Simulation System/netlisters) and this opened the doors for current work and schematic pcells widely came into use.

More details on pcells and their implementation work which has already been done is discussed in next section.

3 DESIGN DETAILS

To create a new component in a library you need to create its symbol, its CDF (Component Description Format) parameters, simulation information and a netlisting procedure (if the component does not has a default netlisting support). Details on pcell design are mentioned in [2] and [3]. Here we will briefly touch upon basic requirements for a schematic pcell design and its characteristics.

3.1 Basic Pcell Structure and its implementation

A parameterized cell or pcell exists at three different levels:

- superMaster (which consists of SKILL code, parameter definitions and default values; it resides on disk);
- subMaster (contains unique geometries produced by unique set of parameter values; resides in virtual memory) and
- instance (derived from sub-master, inherits parameters from subMaster; resides in virtual memory).

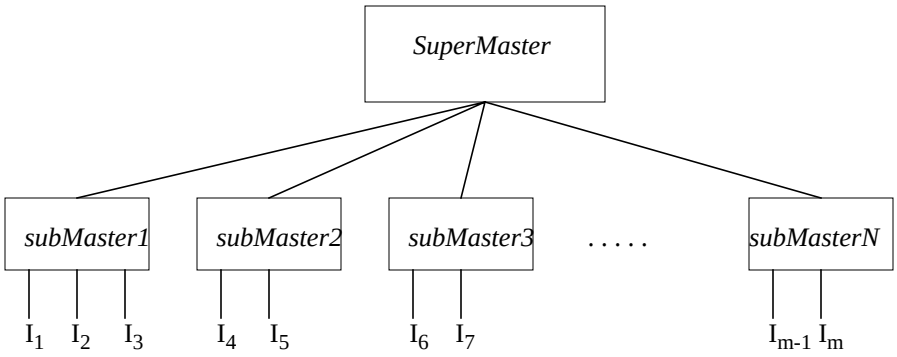


Figure 1: A Pcell structure consisting of superMaster, subMaster and instance

Figure[1], illustrates the super-master of a pcell component at the top and its different sub-masters, with each sub-master representing different instances of the pcell. e.g I_1, I_2... I_m used by a designer. More details of these can be found in [2].

3.1.1 Writing SKILL code to create a symbol

Ordinary schematic symbols are fixed and static but pcells are dynamic and more versatile as compared to schematic symbols. Special, accurate SKILL code is required to create and compile them [2]. Several database calls mentioned in [4] are used along with the pcell code. Pcell code is independent of the underlying databases i.e. C-level database access (CDBA) and OpenAccess (OA). Once written the same pcell code should work fine in both CDBA and OA.

The basic structure of a pcell SKILL code consists of:

- Cell Identifier section: Consists of a list of library identifier(e.g. analogLib), cell name (i.e., name of component e.g. mtline), view name (symbol) and view type (type of view consists of maskLayout, schematic and schematicSymbol, here schematicSymbol). Here the type of pcell is defined - for a schematic pcell viewType must be equal to schematicSymbol.

- Parameter declaration section: This consists of a list of all formal parameters and their defaults. It is important to always define all defaults used in the pcell code, otherwise this may lead to malfunctions.
- Actual body of pcell code: This is the core pcell skill code which is required for building geometries (shapes) and pins, labels etc. for the pcell. Each and every entity of the component is described in terms of the database creation functions. For example a simple line is created via *dbCreateLine*, rectangle is created via *dbCreateRect*, nets on the symbol are created via *dbCreateNet*, terminals via *dbCreateTerm*, labels via *dbCreateLabel*, pins via *dbCreatePin* etc. Various database functions [4] are contained in this portion of code. Special checks like the pcell shape, its limit, variant size etc. are programmed in this section only.

A sample code snippet for mtline pcell component existing in analogLib is shown in Figure[2] as a sample.

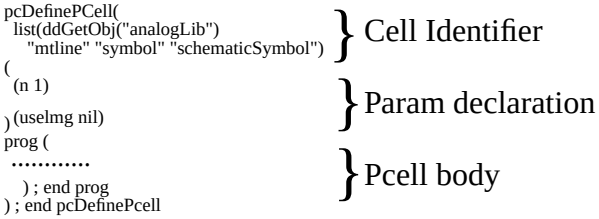


Figure 2: Structure of a sample pcell code

3.1.2 Pseudo code for mtline pcell

The programming of the pcell, i.e., in case of mtline the change in number of pins is described via CDF parameters, ‘n’ and ‘uselmg’. Here ‘n’ describes the number of pins excluding the reference pins, and ’uselmg’ changes the mtline reference pins with respect to the use subcircuit or non-subcircuit model. This controls the overall mtline structure to have an odd or even number of pins.

Based on selection of values of ’n’ and ’uselmg’ the component pins can grow, shrink and vary. The pseudo code for this is illustrated below.

1. Set pcDefinePcell and define all cell identification parameters.
2. Set defaults for formal parameters:
   - n=1
   - uselmg=nil
3. If n and uselmg are not defined then set their values to defaults. Otherwise take values from the component.
4. Write SKILL code to create bounding box, lines, rectangle etc.
5. Write SKILL to create lines, rectangle etc. for pins, terminals, labels etc. If uselmg=nil then create two reference lines (inref/outref) else create a single reference line (ref)
6. For each I = 1 to n (where n = number of pins other than the reference pin.
   - Create other desired pins, their terminals, nets, labels etc.
   - end for loop
7. Add cdsName

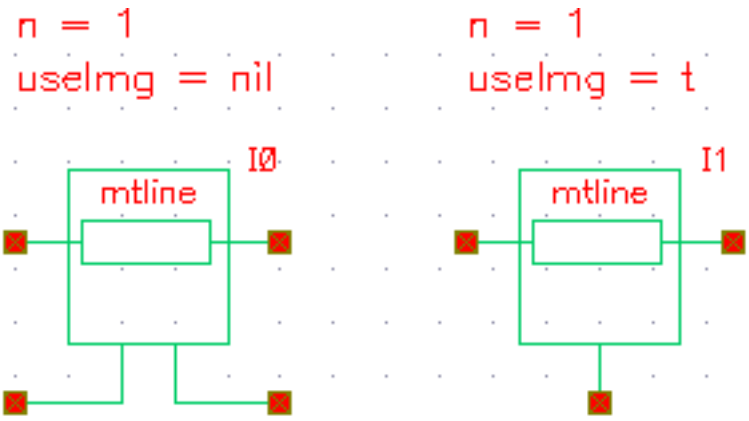
It should be interpreted from the above pseudo-code that the pcell’s lines, terminals, pins etc. are all cre-

ated using ITKDB functions. First only the superMaster is created using defaults mentioned in SKILL code and later based on the values of formal parameters (here, n and uselmg) the pcell's required pins are created and or destroyed from the subMaster variants.

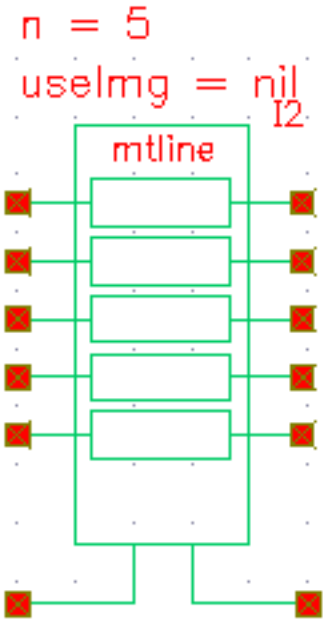
This section is most error prone and any incorrect steps here can lead to failures in pcell compilation process. So careful checking and debugging should be done while writing pcell code. More details of these can be found in [2,3]

3.1.3 Results of work done

To understand more on this lets take the example multi-conductor transmission line (mtline) [5], component where there are two formal parameters - variation in number of pins (n) and use of sub-circuit component (uselmg). It should be noted that these formal parameters are exactly mapped to the component description format (CDF) parameters[1] on the symbol. Based on the values of 'n' and 'uselmg' the component pins can grow, shrink and vary. This is illustrated in Figure[3].



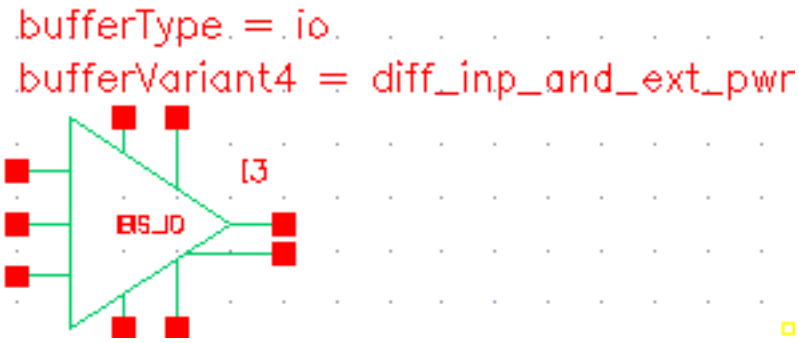
3(a). n=1, uselmg=nil **3(b). n=1, uselmg=t**



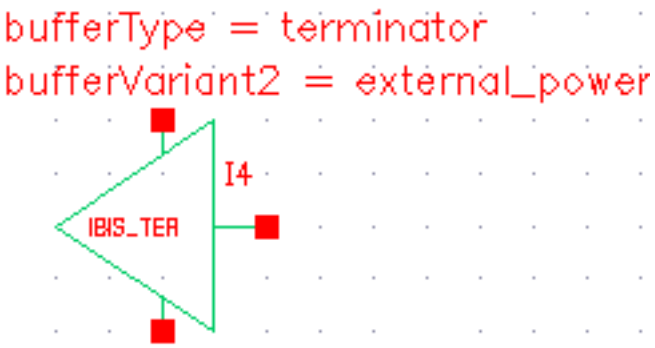
3(c). n=5, uselmg=nil

Figure 3: Different Pcell variants of mtline component

Taking another example of the ibis_buffer component, ref[6], where the shape and pins of the entire component can change from a buffer to a inverted-buffer based on CDF parameters. In this case the CDF parameters are ‘bufferType’ and ‘bufferVariant’. The value of these is specified on the CDF of the pcell instance and the shapes and pins can vary.



4 (a). BufferType=inputOutput, with differential input pins and external power



4(b). Buffer Type=terminator, with external_power

Figure 4: Pcell variants of IBIS_Buffer

Figure[4], shows an example of different ibis_buffer cells having different shapes and number of pins. This requires special programming skills as well as the synchronization between the formal parameters and the shape of the pcell component.

Most of the errors are done in this section and a careful check and debugging should be used for pcell programming.

3.1.4 Mathematical analysis

The limitation of not using a pcell based implementation for the same component is that we will have a geometric number of independent unique variants to be maintained in the library. Mathematically this can be explained as below.

Let us consider a pcell with *m* formal parameters, where *m* belongs to a set of all natural numbers.

Each of the pcell instantiations can have their own set of variations in number of components.

Let $K_1, K_2, \ldots, K_k, \ldots K_m$, be the variations of the formal parameters. Then the total number of variants for the pcell component would be defined as:

$$\prod_{i=1}^{m} K_i \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

For the case as shown in Figure[3], there are 2 formal parameters (m=2) n and uselmg; where n varies from 1 to 5 (K_1=5) and uselmg can have two values either true or false (K_2=2).

So, per the inference made from derivation done in equation (1) we conclude that the mtline pcell, shown in e.g. 3(c), will have a total of

$= K_1 * K_2 = 5 * 2 = 10$ variants.

when, n varies from 1 to 5 and uselmg from t/nil.

In contrast for a similar case in a non-pcell component this would require the creation of 10 separate library components which can be done via a single 'mtline' pcell.

This shows the advantage and importance in usage of the pcell components.

3.2 Adding CDF parameters

CDF (Component Description Format) parameters play an important role for any library component. These parameters are visible in the Edit Object Properties form in the Virtuoso® schematic composer window. Various CDF parameters belonging to the instance are added here. These parameters are of different types ranging from string, int, float, radio, cyclic, boolean, button etc. More details about the CDF parameters their usage and type is mentioned in [1].

The parameters which are visible in the netlist (connectivity information of circuit) file are commonly called as instance parameters (instParameters) and the ones which are not visible in netlist may belong to other parameters (otherParameters). These parameters have a 1-1 mapping with the simulation information of the component and these are know from the simulators help or manuals.

For a pcell care must be taken to ensure that the defaults between the CDF and the pcell must match. These CDF parameters have associated callbacks which perform the desired action on the mouse click or selection made by the user in the Edit Object Properties form.

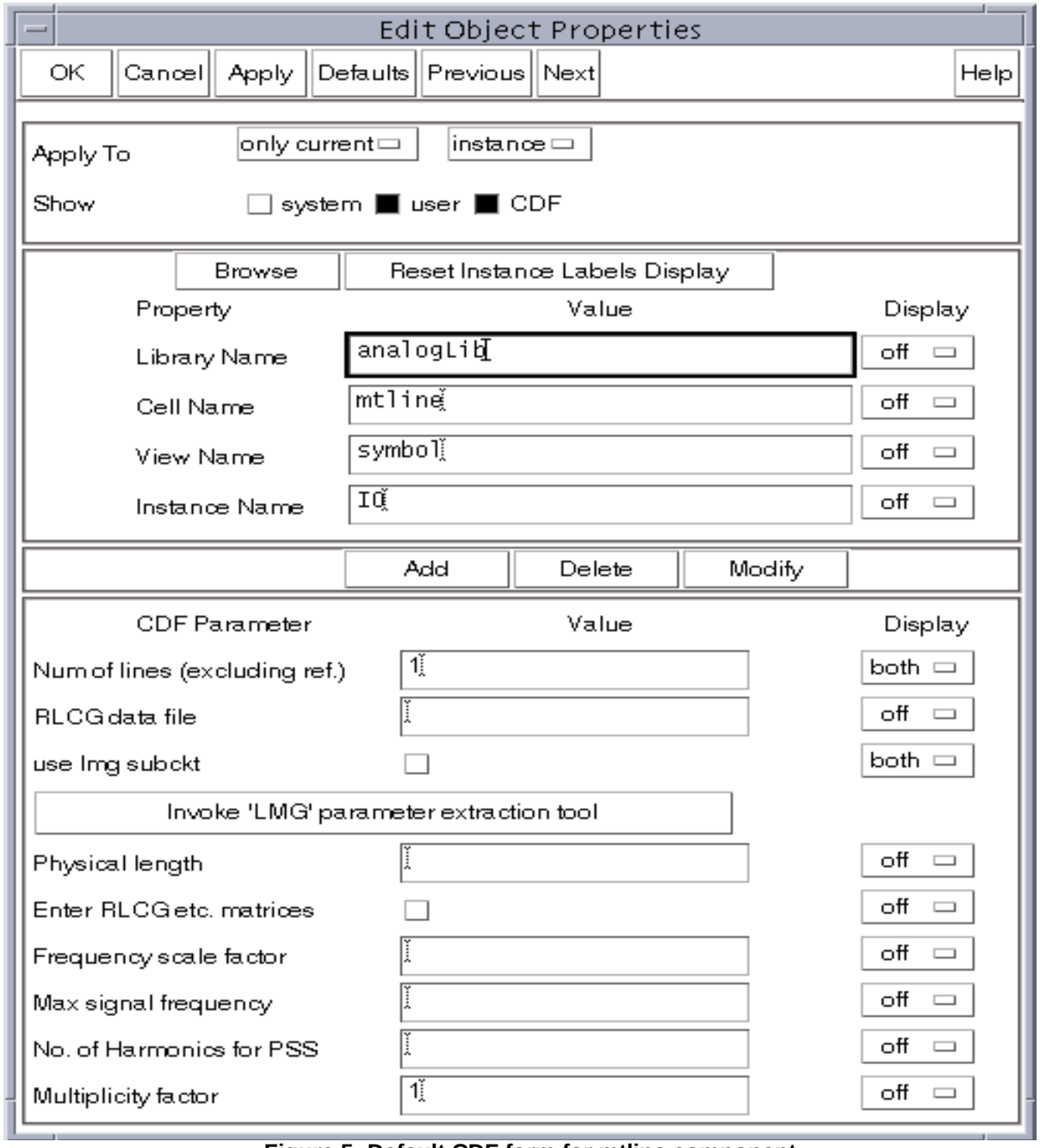


Figure 5: Default CDF form for mtline component

Figure[5] shows a default CDF form for mtline component, note that the default CDF parameter values for n=1 and uselmg=nil. These values match with the snapshot shown in Figure[3a].

3.3 Adding Simulation Information and netlisting support

Simulation information (simInfo) is necessary to simulate a component by a given simulator. This information is unique and specific for each simulator and is based on the simulator internal reference. Details on simInfo are available in [1,5,6]. In general different categories are included in simInfo section for a component instParameters, otherParameters, componentName, termOrder, termMapping, propMapping, netlistProcedure etc.

Pcells are different from primitive components in the sense that their terminals, pins and shapes can vary based on the type of the CDF parameters chosen. In order to take care of all the variants, special netlisting procedures are used for pcells. The netlisting procedure takes care of the formatter object and prints the instance CDF parameters based on the terminals, nets etc. connected to it. Netlisting procedure should be smart enough to handle error conditions which might occur in case of a netlist generation from a user. A common example of such an error is the absence of the required instance parameter on the device. In such cases netlister should flag an error message.

3.4 How to identify a pcell component

The ITKDB (Integrators Tool Kit Data Base) Guide[4] provides various db (database) calls which can be used.

```
cv=dbOpenCellViewByType("analogLib" "res" "symbol")
cv~>isParamCell
=> nil
cv1=dbOpenCellViewByType("analogLib" "mtline" "symbol")
cv1~>isParamCell
=> t
```

Other PIs for dumping and writing the pcell data from the database can be used. These are widely documented and available for customers. Some of such functions include e.g. cdfDump, dbWriteSkill etc.

4. LIMITATIONS, CHALLENGES AND FUTURE WORK

There are various safety rules which should be used while designing schematic pcells. These details are mentioned in [2]. Apart from them following precautions should also be used while designing schematic pcells:

- Use msfb or PD domain workbench. e.g. in icms you can not create a pcell.
- Remember that in a pcell code no function can be re-defined, although you can do this for your local testing. In some cases this causes extra markers to show up on the schematic symbol and cause compilation failures
- Remember to always use the fourth argument to the pcDefinePcell of type equal to schematicSymbol, because by default pcDefinePcell infers the argument to be a masklayout type and this will cause problems when you translate the CDBA pcell on OpenAccess database.
- More precautions are listed in [2].

There are a couple of limitations in the usage of pcells. First one is Virtuoso® AMS (Analog Mixed-Signal) product does not fully supports schematic pcells. Because it does not fully supports netlisting of schematic pcell based designs. Another issue with pcell is the requirement of dynamic simInfo. Dynamic simInfo is a newer term which came into existence with pcells. Since pcells can grow and shrink in size so their termOrder and termMapping can not be static. In order to comply with this dynamic simInfo having a varying simInfo with respect to the component is required. If dynamic simInfo is not supported then the component can be simulated but the terminal currents/voltages for all its pins can not be provided. Currently a workaround of this problem is to enter the termOrder and termMapping values directly in the user's CDF.

The future work in this area can be challenging since schematic pcell is an emerging technology and their ease, usage and dynamics make most analog designers attracted towards them. AMS support and dynamic simInfo in the traditional simulator integration interfaces could be a major work in future to reinforce the usage of pcell simulation flow and its advancements.

5. CONCLUSION

We have described ways to create schematic pcells and have listed some of the pitfalls and precautions which are required to be taken care by CAD designers. Future work related to the field of schematic pcell

design is also described. Challenges in fields of Virtuoso® AMS and dynamic simInfo to make the pcell flow seamless are also discussed.

REFERENCES

1. Component Description Format User Guide, Product Version 5.0 - http://voyager.cadence.com/techpubs/ic50/cdfuser/cdfuserTOC.html
2. Virtuoso® Parameterized Cell Reference, Product Version 5.0 - http://voyager.cadence.com/techpubs/ic50/pcellref/pcellrefTOC.html
3. Cell Design Tutorial, Product Version 5.0, Chapter 6 - http://voyager.cadence.com/techpubs/ic50/celltut/celltutTOC.html
4. Integrator's Toolkit: Database Reference Manual, Product Version 5.0 - http://voyager.cadence.com/techpubs/ic50/itkdbase/itkdbaseTOC.html
5. Feature Functional Specifications for mtline-lmg workflow in Artist (via analogLib mtline component) - Cadence Internal Document.
6. Functional Functional Specification for analogLib/IBIS_buffer component- Cadence Internal Document.