Defining Parameterized Cell using Cadence® SKILL

Vladimir Grozdanov[1], Marin Hristov[2]
1) Smartcom-Bulgaria AD, Sofia, Bulgaria
2) Technical University of Sofia, Sofia, Bulgaria

Abstract: *Parameterized cell is world industry leading method to organize Cadence technology library that in practice gives unlimited flexibility to the designers to use the most appropriate geometry for their topologies. Cadence® SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. SKILL leverages the investment in Cadence technology because the programmer can combine existing functionality and add new capabilities. It lets to customize and extend the design environment that includes parameterized cell design. This paper demonstrates the development of a Cadence parameterized cell using internal Cadence programming language SKILL.*

Keywords: *CAD, SKILL, Microelectronics, Cadence, p-cell, inductors*

1. INTRODUCTION

Recently there is tremendous interest in integrated passive components for RF and Wireless applications. Many microelectronics technologies with their Cadence technology libraries are not advanced enough to offer suitable passive components as inductors and capacitors. They have just several cells with fixed geometry which is a significant limitation or makes these components even useless in many cases.

2. SKILL LANGUAGE DESCRIPTION

Cadence® SKILL [3][6] is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. Because SKILL supports a more conventional C-like syntax, novice users can learn to use it quickly, and expert programmers can access the full power of the Lisp language. SKILL is as easy to use as a calculator as well as being a powerful programming language whose applications are virtually unlimited.

SKILL brings a functional interface to the underlying subsystems. SKILL has functions to access each Cadence tool using an application programming interface. SKILL lets quick and easy customization of existing CAD applications, helps to develop new applications or a custom parameterized cell.

The SKILL programming language lets to customize and extend the design environment. SKILL provides a safe, high-level programming environment that automatically handles many traditional system programming operations, such as memory management. SKILL programs can be immediately executed in the Cadence environment.

SKILL is ideal for rapid prototyping. The programmer can incrementally validate the steps of the algorithm before incorporating them in a larger program.

Storage management errors are persistently the most common reason cited for schedule delays in traditional software development. SKILL's automatic storage management relieves the program of the burden of explicit storage management. The programmer gains control of the software development schedule.

SKILL also controls notoriously error-prone system programming tasks like list management and complex exception handling, allowing the programmer to focus on the relevant details of the algorithm or user interface design. The programs are more maintainable because they are more concise.

The Cadence environment allows SKILL program development such as user interface customization. The SKILL Development Environment contains powerful tracing, debugging, and profiling tools for more ambitious projects.

SKILL leverages the investment in Cadence technology because the programmer can combine existing functionality and add new capabilities.

SKILL allows to access and control all the components of the tool environment: the User Interface Management System, the Design Database, and the commands of any integrated design tool. In SKILL, function calls can be written in either of the following notations:

- Algebraic notation used by most programming languages:
  func(*arg1 arg2* …)
- Prefix notation used by the Lisp programming language:
  (func *arg1 arg2* …)

SKILL is the command language of the Cadence environment. Whenever forms, menus and bindkeys are used, the Cadence software triggers SKILL functions to complete the task.

For example, in most cases SKILL functions can

- Open a design window
- Zoom in by a factor of 2
- Place an instance or a rectangle in a design

Other SKILL functions compute or retrieve data from the Cadence Framework environment or from designs. For example, SKILL functions can retrieve the bounding box of the current window or retrieve a list of all the shapes on a layer.

The programmer can enter SKILL functions directly on a command line to bypass the graphic user interface.

SKILL can be used to define a parameterized cell (p-cell) in Cadence to enhance technology capabilities.

3. WHAT IS A PARAMETERIZED CELL?

A parameterized cell [2], or *pcell*, is a graphic, programmable cell that lets the designer create a customized instance each time when it is placed. In the beginning this cell is just as any other layout cell; then you assign parameters to it. The created pcell is called a *master*. A master is the combination of the graphic layout drawn to make a cell and the parameters assigned to it. After compilation of the master, it is stored in the database in the form of a SKILL procedure.

The edits made to the master appear in the cell instances when compiling the master.

All changes have to be made to the master. An instance of a pcell cannot be edited.

For example, a transistor can be created with assigned parameters that control its width, length, and number of gates. When the designer places instances of the master, different values to each of these parameters can be assigned. According to the parameter values, each instance varies in size and composition.

The designer can create many variations of a transistor using the pcell master. The following master to create an instance with a stretched gate width, an instance with a repeated gate, and an instance with a stretched contact and gate length can be used.

When the designer places an instance of a pcell, the defaults can be accepted or any of the values for the assigned parameters can be changed. Using the default values creates an instance identical to the master cellview.

4. ADVANTAGES IN USING A PCELL

The designer uses pcells to

- Speed up entering layout data by eliminating the need to create duplicate versions of the same functional part [2]
- Save disk space by creating a library of cells for similar parts that are all linked to the same source
- Eliminate errors that can result in maintaining multiple versions of the same cell
- Eliminate the need to explode levels of hierarchy while changing a small detail of a design

Such a p-cell is created as an additional cell in AMS 0.35um CMOS 4 metals technology. The code needed for this is approximately 1500 lines. This includes:

Creating a Component Description Format (CDF)

A sample code for defining one CDF [4] [5]:

```
cdfCreateParam(cdfid
      ?name          "dout"
      ?prompt        "Outer Dimension"
      ?parseAsNumber  "yes"
      ?units         "lengthMetric"
      ?defValue       "200u"
      ?type          "string"
      ?callback      "vg_DOUT_CBF()"
);end cdfCreateParam
```

Defining callback functions

Callback functions are attached to each CDF parameter. All restriction and parameter dependences are coded there. Fragment of one callback function defining some restrictions is shown below [4] [5]:

```
when(evalstring(dout~>value) > 0.001
       printf("\n*WARNING* Inductor Pcell: Value of parameter \"Outer Dimension\"
exceeds the limit - %s\n          Value will be reset to the limit - 1m" w~>value)
       dout~>value = "1m"
);end when > max
when(evalstring(dout~>value) < (evalstring(w~>value)+evalstring(s~>value))*2
       ;incorrect W value
       ;set W value to the min allowed
       printf("\n*WARNING* Inductor Pcell: Incorrect value for parameter \"Metal
Width\" - %s\n          Value will be reset to the minValue - %s" w~>value
pcExprToString((evalstring(w~>value)+evalstring(s~>value))*2))
       dout~>value = pcExprToString((evalstring(w~>value)+evalstring(s~>value))*2)
);end when
```

Creating a symbolic view

The symbolic view is created using Cadence Graphical User Interface.

Coding the p-cell behavioral (the master cellview)

A small piece of this code calculating coordinates of the inductor corner points [2][3] [5]:

```
procedure(vgCalculateIndPathPoints(dout w n s)
prog((points p1 p2 p3 p4 p5 offset point X Y)
p1 = 0:0
p2 = 0:dout-w/2+30
p3 = dout-w:cadr(p2)
p4 = car(p3):w/2+30
p5 = w+s:w/2+30
points = list(p1 p2 p3 p4 p5)
;number of points is 4*n+1
when(n>1
      for(i 6 int(4*n+1)
            point = list(0 0) ; current point
      if(i == int(4*n+1) then
            ;current point is the last point
            offset = w/2
      else
            ;current point is not the last one
            offset = 0
      );end if
      if(evenp(i) then
            ;i is even
            ;i point's X coordinate is the same as the i-1 point
            if(int(evalstring(sprintf(nil "%n.0/4" i)) )!= float(evalstring(sprintf(nil "%n.0/4"
i)) ) then
                  ;going UP
                  Y = cadr(nth(i-4 points))-w-s
                  point = list(car(nth(i-2 points)) Y)
            else
                  ;going DOWN
                  Y = cadr(nth(i-4 points))+w+s
                  point = list(car(nth(i-2 points)) Y)
            );end if going up or down
      else
            ;i is odd
            ;i point's Y coordinate is the same as the i-1 point
            if(int(evalstring(sprintf(nil "%n.0/4" i-1)) )!= float(evalstring(sprintf(nil
"%n.0/4" i-1)) ) then
                  ;going RIGHT
                  X = car(nth(i-4 points))-w-s+offset
                  point = list(X cadr(nth(i-2 points)))
            else
```

```
                    ;going LEFT
                    X = car(nth(i-4 points))+w+s-offset
                    point = list(X cadr(nth(i-2 points)))
                );end if going up or down
            );end if evenp(i)

                points = append1(points point)
        );end foreach point excluding the first five
);end when n>1
if(n==1 then
        ;Underpass not needed
        points = append1(points list(car(nth(4 points)) 0))
);end if n=1
return(points)
);end prog
);end procedure
```

This function calculates coordinates of the inductor corner points. These points depend on "Outer Dimension", "Metal Width", "Number of Turns" and "Space". The demostrated function is significant part of the code and it is called in the body of the function that defines the inductor parameterized cell. The whole code is quite long (approximately 1500 lines) but the comparison of the work needed to create it with the gained benefits of using it absolutely justifies all the efforts.

5. CONCLUSION

This work demonstrates the capabilities of the high-level popular artificial intelligence programming language Cadence® SKILL. A definition of what is parameterized cell is given with explanation of what are the benefits of using it. Sample code parts of the p-cell generation were provided to demonstrate SKILL syntax forms and how one SKILL function is defined.

A p-cell is created as an additional cell in AMS 0.35um Si CMOS technology. This p-cell is a square integrated inductor. It was made according all Austria Micro Systems design and recommendation rules [7] [8] [9]. All recognition layers were also added to make the cell extractable. The code needed for this is approximately 1500 lines. The cell can generate automatically the layout of inductors with the desired geometry parameters. This cell is a big technology capability extension.

Using SKILL it is possible to continue with the optimization of the presented p-cell with addition of octagonal form and support for not only single layer inductors, but also double or multilayer devices.

6. ACKNOWLEDGEMENT

The research described in this paper was carried out within the framework of the contract № BY – TH 115 / 2005.

7. REFERENCES

[1] Grozdanov V. (2007) Automatic Topology Generation Of Integrated Square Inductor, Sofia, Bulgaria, Technical university - Sofia.

[2] Cadence® Design Systems, Inc. (2004) Virtuoso® Parameterized Cell Reference, San Jose, CA, USA, Cadence® Design Systems, Inc.
[3] Cadence® Design Systems, Inc. (2004) SKILL Development Functions Reference, San Jose, CA, USA, Cadence® Design Systems, Inc.
[4] Cadence® Design Systems, Inc. (2005), Cadence® Design Framework II SKILL Functions Reference, San Jose, CA, USA, Cadence® Design Systems, Inc.
[5] Cadence® Design Systems, Inc. (2005) *SKILL Language Reference*, San Jose, CA, USA, Cadence® Design Systems, Inc.
[6] Cadence® Design Systems, Inc. (2005) *SKILL Language User Guide*, San Jose, CA, USA, Cadence® Design Systems, Inc.
[7] Austria Microsystems (2005) 0.35um CMOS C35 RF SPICE Models, Austria Microsystems.
[8] Austria Microsystems (2007) 0.35um CMOS C35 Process Parameters, Austria Microsystems.
[9] Austria Microsystems (2007) 0.35um CMOS C35 Design Rules, Austria Microsystems.