Examensarbete

A Cadence layout wrapper for MATLAB

Examensarbete utfört I Elektroniksystem
av

Tsirepli Ismini

LITH-ISY-EX-ET--06/0324--SE
Linköping 2006
A Cadence layout wrapper for MATLAB

Examensarbete utfört I Elektroniksystem
på Institutionen för systemteknik
vid Linköping tekniska högskola
av

Tsirepli Ismini
LITH-ISY-EX-ET--06/0324--SE

Handledare: Emil Hjalmarson

Examinator: Lars Wanhammar
Linköping 2006-06-20
Abstract

In this thesis, the focus is on creating a wrapper between MATLAB and the Cadence Virtuoso design environment. The central idea is to use the wrapper and write the code for an entire analog layout as scripts in MATLAB. Basically, we will implement a set of necessary commands for performing the most fundamental tasks in layout generation from within MATLAB.

Keywords

SKILL, MATLAB, wrapper, layout generation,
Acknowledgements

First, I would like to thank you my supervisor, Professor Lars Wanhammar, for his support and for granting me the opportunity to pursue this thesis work in Linköping University.

Further, I would like to direct special gratitude to Emil Hjalmarson, for his immense help in learning Cadence/Skill Language, for his valuable discussions and for his precious supporting in this project. Special thanks to my supervisor in Greece, Dr. Ganetsos Theodore, who prompted me to achieve my final work abroad and supported me in each difficulty that I faced.

A big thanks goes out to the staff of Linkoping’s university for their friendship and their support.

Finally, I would like to thank you family and especially my father, Ignatios, and my enchanting boyfriend Dimitrios for their love, the courage that they gave me and for always believing in me.
# Table of Contents

1. Introduction ................................................................................................................. 1

   1.1 Analog Design ........................................................................................................ 2

   1.2 Problems ................................................................................................................ 4

   1.3 Motivation ............................................................................................................... 4

   1.4 Introduction to MATLAB ..................................................................................... 5

   1.5 Introduction to Cadence ......................................................................................... 5

   1.6 Work Description .................................................................................................. 5

       1.6.1 Detailed Description ..................................................................................... 6

2. Commands in Cadence ............................................................................................... 9

   2.1 SKILL functions .................................................................................................... 9

       2.1.1 Our SKILL functions ................................................................................... 12

   2.2 Code Examples ..................................................................................................... 13

       2.2.1 Our Code Example ....................................................................................... 14

3. Basic Codes in MATLAB .......................................................................................... 17

   3.1 Essential MATLAB functions ............................................................................. 17

   3.2 Example of usage ................................................................................................. 19

4. Conclusion .................................................................................................................. 21

References ..................................................................................................................... 23
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>1</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>17</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Throughout the last years, a rapid growth in Electronics business has been observed. The technology scale down has led to integration of complete systems in a single integrated circuit. Figure 1.1 shows the impact of this scale down over time. This trend is better known as Moore’s Law. This entire tendency therefore led to the growth of VLSI field.

Fig.1.1: The National Technology Roadmap for Semiconductors 1998. [1]

Due to the high complexity of VLSI systems, it is difficult for a designer to handle all constraints sufficiently. Hence, the use of computers and Computer Aided Design (CAD) tools are necessary. In the case of electronics design, the term Electronic Design Automation (EDA) is used for such design tools. The aims of these tools are to support the designer during the process of realizing an integrated circuit (IC). Integrated circuits are used in a large variety of applications, e.g. microprocessors, communication, audio and video applications. In addition, the trend of integrated circuit design is toward designing mixed analog and digital systems. The digital parts of mixed signal can efficiently be designed by computer aided design software. In the case of analog and mixed-signal circuit design, specialist knowledge and years of experience, is required.
Here we focus on Layout generation, which is one of the most important and time consuming steps in the analog design process. There are, however, other challenging tasks in analog design, e.g. topology selection and device sizing.

1.1 Analog design

Analog design is the intention and disposition of the parts or details of something according to a target specification. With analysis, we introduce an analog system and the suitable properties (static, dynamic, physical performance or process supply voltages, temperature constrain) for the solution. The solution (electrical, physical, testing) is always unique. With design, we specify a set of properties and we try to find a system processing for them. Here the solution is not unique. In Fig.1.2, we can see that these two terms are interdependent.

![Diagram](image1)

**Fig. 1.2:** Analysis versus synthesis (design)

Figure 1.3 shows the design flow of an analog design automation tool. In the first step, we enter the performance specification. The design steps then consists of topology selection, device sizing and layout generation. In the topology selection, a suitable circuit topology is selected. This topology must be selected
with respect to the specific requirements. Device sizing is the parameter design. This step is time consuming because of the complex and non-linear relation between performance measures and design parameters (device sizes). Typically, an optimization algorithm is used to determine the device sizes in an analog design automation tool. The final step is layout generation. In this step, we generate the corresponding layout in accordance with the preceding steps.

Analog design is traditionally done in a not well-defined manner and makes little use of previous designs. To be successful the design process must be highly structured and organized. For the complexity, we need to use as much hierarchy as possible and appropriate organization techniques. Therefore, we must document the design in
an efficient manner and perform relevant simulations. So if we increase the complexity we are able to make appropriate simplifications and assumptions, we are able to learn from failure.

Although all these things they are not always so simple. If one circuit changes in an analog design all the manual work may have to be repeated and verified again. The performance of the circuit, the parasitic effects may have to be validated and adjusted. Hence, all these are time consuming and demand a lot of knowledge in order to keep the analog design within the performance requirements.

1.2 Problems

One problem in analog and mixed-signal design is the effect of layout parasitics. Compensating for these parasitics include time-consuming transistor sizing and layout changes. To increase design and yield efficiently we can use a parameterized geometrical layout. Although the approach has limited flexibility and the estimation of parasitics is time consuming, it will increase the reliability of the design flow.

Many universities have conducted researches for compaction, routing and calculation of sensitivity in analog integrated circuits. The researches are applied to small analog integrated circuits because it is easier to illustrate the approached and to attribute suitable results [3]. The researches in analog design automation differ from that at in the digital domain due to its complexity and large degree of design freedom [4].

1.3 Motivation

Summarizing the previous discussion, we conclude that the layout generation for IC design is a laborious and time-consuming process. For having a good layout generation, we must control the effects of the parasitics on circuit performance and make sure that the circuit after layout still performs within its specification. Consequently, there is a need for improvement in the field of layout generation and analog synthesis in general.
### 1.4 Introduction to MATLAB

Matlab (Matrix laboratory) is an interactive software system for numerical computations and graphics. As the name suggests, Matlab is especially designed for matrix computations: solving systems of linear equations, computing eigenvalues and eigenvectors, factoring matrices. In addition, it has a variety of graphical capabilities, and can be extended through programs written in its own programming language.

Matlab is designed to solve problems numerically, that is, in finite-precision arithmetic. Therefore it produces approximate rather than exact solutions, and should not be confused with a symbolic computation system (SCS) such as Mathematica or Maple. It is a tool designed for different tasks and is therefore not directly comparable.

### 1.5 Introduction to Cadence

Cadence provides an Electronic Design Automation (EDA) environment that allows integrating in a single framework different applications and tools. These tools are completely general, supporting different fabrication technologies. Cadence can be run on UNIX terminals or PCs loaded with Linux or X Windows servers.

Cadence is the largest supplier of electronic design automation products, methodology services, and design services used to manage the design of semiconductors, computer systems, networking and telecommunications equipment, consumer electronics, and a variety of other electronics-based products.

### 1.6 Work description

In this thesis, the focus is on creating a wrapper between MATLAB and the Cadence Virtuoso design environment. The central idea is to use the wrapper and write the code for an entire analog layout as scripts in MATLAB. Basically, we will implement a set of necessary commands for performing the most fundamental tasks in layout generation from within MATLAB. One of the major reasons for
creating the wrapper is that with the implemented codes in MATLAB we can export the layout to several VLSI design platforms. In addition, the proprietary language of Cadence, SKILL, is an obstacle for final year workers that have good knowledge of MATLAB but little knowledge of SKILL.

1.6.1 Detailed description

Within this thesis project, some of the most fundamental layout commands should be added to the wrapper. Typical examples of MATLAB commands that should be implemented are

**layoutPath(<layer>, <width>, <coordinates>,<shield>)**
Create paths, wires, in the layout

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer</td>
<td>name of the layer</td>
</tr>
<tr>
<td>width</td>
<td>width of the path</td>
</tr>
<tr>
<td>coordinates</td>
<td>coordinates of the layer</td>
</tr>
<tr>
<td>shield</td>
<td>Shield added, e.g. 0 means that no shield is used, 1 means that a metal layer can be placed underneath the path.</td>
</tr>
</tbody>
</table>

**layoutPin(<name>, <layer>, <size>, <coordinate>)**
Add pins, terminals, to the layout

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>pin name</td>
</tr>
<tr>
<td>layer</td>
<td>name of the layer</td>
</tr>
<tr>
<td>size</td>
<td>size of the pin</td>
</tr>
<tr>
<td>coordinate</td>
<td>center location of the pin</td>
</tr>
</tbody>
</table>
layoutContact(<layer1>, <layer2>, <size>, <coordinate>)
Create contact, i.e., a via between user defined layers.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer1</td>
<td>name of the first contact</td>
</tr>
<tr>
<td>layer2</td>
<td>name of the second contact</td>
</tr>
<tr>
<td>size</td>
<td>size of the contact in any layer</td>
</tr>
<tr>
<td>coordinate</td>
<td>center point of the contact</td>
</tr>
</tbody>
</table>

layoutInstance(<name>, <library_name>, <cell_name>, <coordinate>)
Create an instance of an analog cell

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>the name of the instance</td>
</tr>
<tr>
<td>library_name</td>
<td>the library name of the instance</td>
</tr>
<tr>
<td>cell_name</td>
<td>the cell name of the instance</td>
</tr>
<tr>
<td>coordinate</td>
<td>center point of the instance</td>
</tr>
</tbody>
</table>
Introduction
Chapter 2

Commands in Cadence

In the first chapter we briefly discussed layout generation and the need for automation to obtain high-quality layouts. Here our goal is to implement a set of commands for performing the most fundamental tasks in layout generation from within MATLAB. In this chapter we will introduce some of the SKILL functions used in this work.

Also some of our custom SKILL functions developed within this these work are discussed. At the end of the chapter there are some code examples illustrating the use of these SKILL functions.

2.1 SKILL functions

dbOpenCellViewByType
Open a cell view. If the cell view don’t exist a new one is created, the line below shows how the command can be used to create a new layout.

\[
\text{cv} = \text{dbOpenCellViewByType}("\text{modulgeneratorlab2" } \text{"andxor" } \text{"layout" } \\text{"maskLayout" } \text{"w") } \text{"layout" } \text{"maskLayout" } \text{"w")}
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;modulgeneratorlab2&quot;</td>
<td>This statement tells us that the design to open is located in the modulgeneratorlab2 library.</td>
</tr>
<tr>
<td>&quot;andxor&quot;</td>
<td>&quot;andxor&quot; is the name of the cellview.</td>
</tr>
<tr>
<td>&quot;layout&quot;</td>
<td>Tells Cadence to create / open the design as a layout.</td>
</tr>
<tr>
<td>&quot;maskLayout&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;w&quot;</td>
<td>This statement indicates in which mode the layout is to be opened.</td>
</tr>
</tbody>
</table>
**dbCreateInst**
dbCreateInst creates a new instance of an existing design in the cellview.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cv</td>
<td>The cellview where we want to insert the instance.</td>
</tr>
<tr>
<td>and</td>
<td>The cellview that will be inserted as an instance in cv.</td>
</tr>
<tr>
<td>‘nil</td>
<td>Here we can give a specific name to the new instance.</td>
</tr>
<tr>
<td>list(0 0)</td>
<td>Creates a list with the two coordinates of the insertion point.</td>
</tr>
<tr>
<td>1</td>
<td>The number of instances to insert, if we have specified a name for the instance this will override the number of copies specified here because of the fact that all names in a design is unique.</td>
</tr>
<tr>
<td>“R0”</td>
<td>The angel of rotation for the block to be inserted.</td>
</tr>
</tbody>
</table>

**rodCreatePath**
rodCreatePath creates a wire. In this case no width is specified for the wire, hence the minimum width will be used.

`rodCreatePath( ?layer "MET2" ?pts list(term1~>"cC" car(term1~>"cC"):cadr(term2~>"cC") term2~>"cC")?cvId cv)`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?layer</td>
<td>The metal layer that the wire will use.</td>
</tr>
<tr>
<td>?pts</td>
<td>The points of envelop for the wire.</td>
</tr>
<tr>
<td>?cvId</td>
<td>The cellview where to place the wire.</td>
</tr>
</tbody>
</table>
**leCreateContact**

Create a contact.

\[
\text{leCreateContact}( \text{cv } "\text{VIA}_C" \text{ insertPt } "\text{R0}" 0.5 0.5 1 1 0 0 "\text{center}" "\text{center}" )
\]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cv</td>
<td>The cellview in which the contact is to be created.</td>
</tr>
<tr>
<td>&quot;VIA_C&quot;</td>
<td>Type of contact (Same as when a contact is created manually).</td>
</tr>
<tr>
<td>insertPt</td>
<td>The coordinates where the contact is placed.</td>
</tr>
<tr>
<td>&quot;R0&quot;</td>
<td>Rotation.</td>
</tr>
<tr>
<td>0.5</td>
<td>The width of the contact.</td>
</tr>
<tr>
<td>0.5</td>
<td>The length of the contact.</td>
</tr>
<tr>
<td>1</td>
<td>Specifies the number of rows if an array of contacts is to be created.</td>
</tr>
<tr>
<td>1</td>
<td>Specifies the number of columns if an array of contacts is to be created.</td>
</tr>
<tr>
<td>0</td>
<td>Specifies the distance between the columns in a contact array.</td>
</tr>
<tr>
<td>0</td>
<td>Specifies the distance between the rows in a contact array.</td>
</tr>
<tr>
<td>&quot;center&quot;</td>
<td>Specifies the horizontal location of the contacts origin.</td>
</tr>
<tr>
<td>&quot;center&quot;</td>
<td>Specifies the vertical location of the contacts origin.</td>
</tr>
</tbody>
</table>
2.1.1 Our SKILL functions

Contact(via box cv)

In Contact we create a contact, we set the contact reference list and we associate the contact name with the via layer name. The order is list (<contact name> <via layer name>).

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>via</td>
<td>The name of the contact</td>
</tr>
<tr>
<td>box</td>
<td>Creates a list with the two coordinates of the insertion point.</td>
</tr>
<tr>
<td>cv</td>
<td>The cellview where we want to insert the contact.</td>
</tr>
</tbody>
</table>

VvvGetContacts(M1 M2 cv)

VvvGetContacts return the intermediate contacts between two layers.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>name of the first metal, text string ,example &quot;M1&quot;</td>
</tr>
<tr>
<td>M2</td>
<td>name of the second metal, text string, example &quot;M2&quot;</td>
</tr>
<tr>
<td>cv</td>
<td>pointer to cellview</td>
</tr>
<tr>
<td>output</td>
<td>list of via names, list(&quot;VIA1&quot; &quot;VIA2&quot;....)</td>
</tr>
</tbody>
</table>

In VvvGetContacts we specify the name of VIA between POLYSILICON and METAL. In the example below we see how we can use this command to get the name of contacts connecting several metal layers. If we put as input "met1" and "met2" the output will be list("VIA1").
If we put as input “met1” and “met3” the output will be list(“VIA1 VIA2”)

and so on…

**fcontact(box M1 M2 cv)**

In fcontact, we can get the contacts between intermediate metal layers in the Cadence Layout environment.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M1</strong></td>
<td>name of the first metal, text string, example “MET1”</td>
</tr>
<tr>
<td><strong>M2</strong></td>
<td>name of the second metal, text string, example “MET4”</td>
</tr>
<tr>
<td><strong>cv</strong></td>
<td>pointer to cellview</td>
</tr>
<tr>
<td><strong>output</strong></td>
<td>list of via names, list(“VIA1” “VIA2” “VIA3”)</td>
</tr>
</tbody>
</table>

**2.2 Code Examples**

We started by writing simple codes in Skill language. Some examples are these:

1) To load a piece of skill code:
   ```plaintext
   loadi(path)
   ```
2) Open a new cellview for our new layout
   `cv = dbOpenCellViewByType("skill.ex" "ex.a" "layout" "maskLayout" "w")`

3) Create an instance
   First we need to open the cellview of the instance to create.
   `instance = dbOpenCellViewByType("skill.ex" "A" "layout" "maskLayout" "r")`
   now we may place it
   `dbCreateInst(cv instance "OurFirstInstance" list(0 0) "R0")`

   Figure 2.1 shows the result of this place operation

---

4) Create a path
   `rodCreatePath( ?layer "MET1" ?cvId cv ?pts list(20:20 20:30 30:30) )`

   The result is shown in figure 2.2

---

Fig.2.1: Place an instance

Fig.2.2: Create a simple path
5) Create a contact

```csharp
leCreateContact(
    cv
    "VIA1_C"
    list(30 30)
    "R0"
    0.5
    0.5
    4
    6
    1
    1
    "center"
    "center"
)
```

Figure 2.3 shows the result of the contact

![Contact Result](image)

**Fig.2.3:** Create a contact

### 2.2.1 Our Code Example

```csharp
1); load a function
loadi("/home/tde/ismts499/cadence/skill/Contact.il")
; open a cellview
cv = dbOpenCellViewByType("lernskill" "test" "layout"
"maskLayout" "r")

;create variable for the contact name
via = "VIA1"
;list with the two coordinates of the insertion point.
box = list(list(-25 -5) list(25 5))
```
viaList = list(list("VIA1" "VIA1_C") list("VIA2" "VIA2_C") list("VIA3" "VIA3_C")) ;create contact Contact(via box cv)

2) load a function
load("/home/tde/ismts499/cadence/skill/fcode.il") ;open a cellview cv = dbOpenCellViewByType("skill" "ex.a" "layout" "maskLayout" "a")

;set the contact boundary box box=list(list(-2 -3) list(2 3)) ;set metal name M1="MET1" M2="MET4"

fcontact(box M1 M2 cv).

The result is shown in figure 2.4. In figure 2.4, we have manually separated the contacts for visibility. In reality, all contacts are placed on top of each other.

![Fig.2.4: Create several contacts](image-url)
Chapter 3

Basic Codes in MATLAB

The idea here is to write a function in MATLAB. This function should generate a skill file (.il) that contains executable SKILL code. The SKILL code makes use of the commands discussed in the previous chapter to generate the layout. Using this approach a wrapper between Cadence and MATLAB is implemented. Figure 3.1 shows this interface between MATLAB and Cadence.

Fig 3.1: The interface between Cadence and MATLAB.

3.1 MATLAB functions

We have implemented some of the most basic commands needed for analog layout generation.

CreateContact(<file_id>,<layer1>,<layer2>,<box>)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_id</td>
<td>a file id is a reference to an open file. To open a file write: file_id = fopen('/this/is/the/contact/to/my/file');</td>
</tr>
<tr>
<td>layer1</td>
<td>name of the first layer, this is given as text string e.g. “MET1”</td>
</tr>
<tr>
<td>layer2</td>
<td>name of the second layer, this is given as text string e.g. “MET2”</td>
</tr>
<tr>
<td>box</td>
<td>MATLAB vector defining the perimeters of the contact. The box is defined with [x1 y1 x2 y2].e.g., [1 1 2 2];</td>
</tr>
</tbody>
</table>
Example:

Command call in MATLAB:

```
CreateContact(fo, "MET1", "MET3", [1 1 2 2])
```

Text written to SKILL file:

```
fcontact("MET1", "MET3", list(1:1 2:2), cv)
```

**CreatePath(<file_id>, <layer>, <width>, <points>)**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_id</td>
<td>reference to an open file.</td>
</tr>
<tr>
<td>layer</td>
<td>name of the layer used to draw the wire, this is given as a text string e.g. &quot;MET1&quot;</td>
</tr>
<tr>
<td>width</td>
<td>the width of the path e.g. 0.6</td>
</tr>
<tr>
<td>points</td>
<td>a vector containing coordinates for the path. Coordinates are defined by interleaving x and y coordinates [x1 y1 x2 y2 x3 y3 x4 y4 ...], e.g., [10 10 10 20 20 20 20 30 30 30].</td>
</tr>
</tbody>
</table>

Example:

Command call in MATLAB:

```
CreatePath(fo, "MET1", 1.2, [10 10 10 20 20 20])
```

Text written to SKILL file:

```
rodCreatePath(?layer "MET1" ?width 1.2 ?cvId cv ?pts list(10:10 10:20 20:20))
```

**openCellView(<file_id>,<libname>,<cellname>,<mode>)**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_id</td>
<td>reference to an open file.</td>
</tr>
<tr>
<td>libname</td>
<td>library name of the layout to open.</td>
</tr>
<tr>
<td>cellname</td>
<td>cell name of the layout to open.</td>
</tr>
<tr>
<td>mode</td>
<td>the mode in whitch to open the layout, e.g., &quot;w&quot;</td>
</tr>
</tbody>
</table>
Example of usage

Example:

Command call in MATLAB:

```
openCellView(fo, "skill", "example1", "w")
```

Text written to SKILL file:

```
cv = dbOpenCellViewByType("skill" "example" "layout" "maskLayout" "w")
```

```
placeInstance(<file_id>, <libname>, <cellname>, <coordinates>, <rotation>)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_id</td>
<td>reference to an open file.</td>
</tr>
<tr>
<td>libname</td>
<td>library name of the layout to place.</td>
</tr>
<tr>
<td>cellname</td>
<td>cell name of the layout to place.</td>
</tr>
<tr>
<td>coordinates</td>
<td>coordinates indicating where to place the instance, [x y], e.g. [10 10].</td>
</tr>
<tr>
<td>rotation</td>
<td>determines the rotation of the block, this is a text string, e.g., &quot;R90&quot;.</td>
</tr>
</tbody>
</table>

Example:

Command call in MATLAB:

```
placeInstance(fo, "skill", "example1", [10 10], "R0")
```

Text written to SKILL file:

```
instance = dbOpenCellViewByType("skill" "example" "layout" "maskLayout" "r")
dbCreateInst(cv instance list(10:10) "R0")
```

3.2 Example of usage

1) To open a cell view, place an instance, draw a path and place a contact the following commands should be written in MATLAB:

```
fo=fopen('output.il','w') %open an output file
```
OpenCellView(fo, "skill", "example1", "w")
placeInstance(fo, "skill", "example2", [10 10], "R0")
CreatePath(fo, "MET1", 1.2, [10 10 10 20 20 20])
CreateContact(fo, "MET1", "MET3", [1 1 2 2])
fclose(fo)

The output file (output.il) should contain the following:

cv = dbOpenCellViewByType("skill" "example1" "layout"
"maskLayout" "w")
instance = dbOpenCellViewByType("skill" "example2" "layout"
"maskLayout" "r")
cv = dbCreateInst(cv instance list(10:10) "R0")
rodCreatePath(?layer "MET1" ?width 1.2 ?cvId cv ?pts list(10:10 10:20 20:20))
fcontact("MET1", "MET3", list(1:1 2:2), cv)
Chapter 4

Conclusions

In this thesis, we tried to implement a set of necessary commands for performing the most fundamental tasks in layout generation from within MATLAB. The idea is to create a wrapper between MATLAB and the Cadence language SKILL. The designer should be able to use the wrapper and write the code for an entire analog layout as scripts in MATLAB.

It is clear that much work remains to perform layout generation from within MATLAB. Our aim is to improve and extend these commands to bridge the gap. This will include implementing additional commands.

In future versions additional commands will be added to support several VLSI platforms.
Conclusions
References


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtidiga ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns detlösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydda mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Tsirepli Ismini