# Introduction to RWCap

## Overview

The RWCap is a parallel program suitable for extracting the capacitances among 3-D rectilinear-shaped conductors. The algorithm inside is the floating random walk (FRW) algorithm. The RWCap is free of discretization error, and suitable for the capacitance extraction of large-scale Manhattan VLSI layout.

# Chapter 1 Input File Format

3D format file is used to describe 3D structures for accurate extraction. RWCap uses the 3D format file as the input file.

The input file statements consist of comments and/or commands.

* Lines beginning with /\* and ending with \*/ are comment lines and can be ignored.
* All commands may be defined by the set description statements. The set description statements use the signs similar with HTML. It should begin with <> and end with [/].
* The following commands are available in RWCap which can be classified into 3 groups:
  1. Identified commands: **qbem.**All the other commands should be included in this command.
  2. Primary commands**: window, medium, conductor, task.**
  3. Basic geometry definition commands: **block, poly.**The geometry definition commands can be included in the primary commands “**medium, conductor”.**
  4. Task definition commands: **capacitance**The task definition commands can only be included in command **task.**
* The input to RWCap is case insensitive, meaning two instructions when written with all lowercase or all uppercase letters are same.

The syntax and usage of each command are described in this section.

## Identified Command

## qbem

The **qbem** command is the symbol of the RWCap input format file. The file should start with <qbem>, and end with [/qbem].

## Primary Commands

## window

The **window** command sets the simulation cube size in three-dimension.

<**window**>

**v1** <*vector*>

**v2** <*vector*>

**[boundary condition]**

[/**window]**

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| V1 | vector | Coordinates of the left bottom of the window |
| V2 | vector | Coordinates of the right top of the window |
| Boundary condition | neumann or dirichlet | Boundary condition on the six surfaces of the simulation window, the default is neumann |

## medium

The **medium** command describes one dielectric in three-dimension when compute the capacitance. It can contain one or more basic geometry description statements.

<**medium**>

**<***BASIC GEOMETRY DESCRIPTION***>**

**…**

**diel** <*VALUE>*

[/**medium]**

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| Basic Geometry | definition | Geometry description statements, now only BLOCK is supported |
| diel | value | Permittivity of dielectric |

## conductor

The **conductor** command describes one conductor in three-dimension. It can contain one or more basic geometry description statements.

<**conductor**>

**name** *<NAME>*

**<***BASIC GEOMETRY DESCRIPTION***>**

[**<***BASIC GEOMETRY DESCRIPTION***>]**

**…**

[/**conductor]**

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| name | character | Name of the conductor. **NAME** must begin with an alphabetic character, but may include up to 100 alphanumeric characters. |
| Basic Geometry | definition | Geometry description statements, such as : BLOCK, POLY. |

## task

The **task** command defines the specific computational task to be carried on. Four different tasks can be executed including: CAPACITANCE, CHARGE, RESISTANCE and CURRENT.

<**task**>

<*TASK DEFINITION*>

[*TASK DEFINITION*]

…

[/**task**]

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| Task Definition | definition | Task definition statements, such as : capacitance |

## Task Definition Command

## capacitance

The **capacitance** command defines task to compute the self capacitance on the specific conductor and all the coupling capacitance between it and all other conductors.

<capacitance>

<*CONDUCTOR NAME*>

[<*CONDUCTOR NAME*>]

…

[**RWCAP\_ALL**]

[/capacitance]

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| Conductor NAME | character | The NAME of the specific conductor for capacitance computation. If the NAME is “RWCAP\_ALL” means all the capacitances will be extracted. |

## Basic Geometry Definition Commands

## block

The **block** command defines a rectangular box. Here *vector* is the collection of three numbers X,Y,Z, separated by commas. These numbers are coordinates of a vector on a global coordinate system (See Figure 1).

<**block**>

**basepoint** <*VECTOR*>

**v1** *<VECTOR>*

**v2** *<VECTOR>*

**hvector** <*VECTOR*>

**[/block]**

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| basepoint | Vector | Coordinate of the base point of the block.  (**default units:** microns) |
| V1 | Vector | Direction vector of the length axis. This vector is defined in the *local coordinate system* according to the base point. The magnitude of this vector is the length of the block.  (**default units:** microns) |
| V2 | vector | Direction vector of the width axis. This vector is defined in the *local coordinate system* according to the base point. The magnitude of this vector is the width of the block.  (**default units:** microns) |
| Hvector | Vector | Direction vector of the height axis. This vector is defined in the *local coordinate system* according to the base point. This vector is always perpendicular to the surface determined by V1 and V2. The magnitude of this vector is the height of the block.  (**default units:** microns) |

Figure 1 clarifies the meanings of these parameters.



Examples:

The following commands generate a block 3 units long in the X direction, 2 units long in the Y direction and 2.5 units long in the Z direction with the base point at (1,1,2).

<block>

basepoint (1,1,2)

v1 (3,0,0)

v2 (0,2,0)

hvector (0,0,2.5)

[/block]

## poly

The **poly** command defines a polyhedron with **n** vertices in the polygon base (bottom surface) and by extrusion of this base (See Figure 2). The <*POINT*> is the collection of two numbers X,Y, separated by commas.

<**poly**>

**basepoint** <*VECTOR*>

**v1** *<VECTOR>*

**v2** *<VECTOR>*

**hvector** <*VECTOR*>

<**coord**>

<*POINT*> <*POINT*> …

[/**coord]**

**[/poly]**

|  |  |  |
| --- | --- | --- |
| Parameter | DataType | Definition |
| basepoint | Vector | Coordinate of the base point of the block.  (**default units:** microns) |
| V1 | Vector | This vector is used to define the X axis of the *local coordinate system* according to the base point.  (**default units:** microns) |
| V2 | vector | This vector is used to define the Y axis of the *local coordinate system* according to the base point.  (**default units:** microns) |
| Hvector | Vector | Direction vector of the extrusion. This vector is in the *local coordinate system*. This vector is arbitrary oriented. The magnitude of this vector is the height of the extrusion.  (**default units:** microns) |
| Coord | point | x,y coordinates of the vertices of the polygon base expressed in the *local coordinate system* with origin **v1** and **v2.** The polygon base lies in the xy-plane of this local coordinate system. A minimum of 3 coordinate points should be given.  (**default units:** microns) |

Figure 2 clarifies the meanings of these parameters.



Examples:

The following commands generate a polyhedron. The polygon is defined by four vertices, with a height of 2.5. The origin of the local coordinate system is the same as the origin of the global coordinate system; the direction perpendicular to the plane of the polygon is along the global Z axis.

<poly>

basepoint (1,1,2)

v1 (1,0,0)

v2 (0,1,0)

hvector (0,0,2.5)

<coord>

(2,2) (2,4) (4,4,) (4,2)

[/coord]

[/block]

## Constraints of the Input File

The structures described in the input file will have the following constraints:

* All the conductors should not be overlap with others.
* The geometries defined in different command sets of MEDIUM should be separate. The overlap of them is not allowed.

## Example of the Input File

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

<qbem>

<window>

v1 (10.7500, 12.2000, -0.0100)

v2 (11.5500, 12.9250, 9.9330)

dirichlet

[/window]

<medium>

<block>

basepoint (10.7500, 12.2000, -0.0100)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 0.3450)

[/block]

diel 3.9000

[/medium]

<medium>

<block>

basepoint (10.7500, 12.2000, 0.3350)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 0.7500)

[/block]

diel 3.9000

[/medium]

<medium>

<block>

basepoint (10.7500, 12.2000, 1.0850)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 1.5980)

[/block]

diel 3.9000

[/medium]

<medium>

<block>

basepoint (10.7500, 12.2000, 2.6830)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 1.5000)

[/block]

diel 3.9000

[/medium]

<medium>

<block>

basepoint (10.7500, 12.2000, 4.1830)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 5.7500)

[/block]

diel 7.5000

[/medium]

<conductor>

name groudplane

<block>

basepoint (10.7500, 12.2000, -0.0100)

v1 (0.8000, 0, 0)

v2 (0, 0.7250, 0)

hvector (0, 0, 0.0100)

[/block]

[/conductor]

<conductor>

name conductor1\_61\_1

<block>

basepoint (10.9500, 12.4000, 0.3350)

v1 (0.6000, 0, 0)

v2 (0, 0.1250, 0)

hvector (0, 0, 0.2500)

[/block]

<block>

basepoint (10.9500, 12.2000, 0.3350)

v1 (0.4000, 0, 0)

v2 (0, 0.2000, 0)

hvector (0, 0, 0.2500)

[/block]

<block>

basepoint (10.7500, 12.5250, 0.3350)

v1 (0.8000, 0, 0)

v2 (0, 0.2750, 0)

hvector (0, 0, 0.2500)

[/block]

<block>

basepoint (10.7500, 12.8000, 0.3350)

v1 (0.5750, 0, 0)

v2 (0, 0.1250, 0)

hvector (0, 0, 0.2500)

[/block]

[/conductor]

<task>

<capacitance>

conductor1\_61\_1

[/capacitance]

[/task]

[/qbem]

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Chapter 2 API Document

Use **make** commandto compile the whole project, then **librwcap.a** and **librwcap.so** will be generated in out-static and out-shared directorys respectively. Src/include/ directory contains all the header files.

Three classes are provided to interact with rwcap: **configure**, **problem\_manager** and **rwcap\_result**.

At last, we provide an example of using the APIs.

## class configure

**Constructor**

* configure::configure();

**Member functions**

* configure & confiugre::**set\_input\_file**(const std::string & input\_file)

**Parameters**

input\_file: the name of input file

**Return value**

The configure object (\*this)

**Note**

You must call the function to set the input file, no default input file spectified.

* configure & confiugre::**set\_table\_path**(const std::string & path)

**Parameters**

path: path to the GFT/WVT tables

**Return value**

The configure object (\*this)

**Note**

If you don’t call the function, the default table path “./tables” will be used.

* configure & confiugre::**set\_parallel\_num**(int n)

**Parameters**

n: parallel number, any number less than 1 will be considered as 1

**Return value**

The configure object (\*this)

**Note**

If you don’t call this function, the default parallel number is 1.

* configure & confiugre::**set\_variance**(double variance)

**Parameters**

variance: the threshold of relative error of self capacitance

**Return value**

The configure object (\*this)

**Note**

If you don’t call this function, the default variance is 0.5

* configure & confiugre::**set\_couple\_variance**(double variance)

**Parameters**

variance: the threshold of relative error of couple capacitances

**Return value**

The configure object (\*this)

**Note**

If you don’t call this function, the default couple variance is 0, which means no couple variance control is used

* configure & confiugre::**set\_log\_file**(const std::string & log\_file)

**Parameters**

log\_file: log file name, rwcap will append to this file. There are two special value **stdout** and **stderr,** stdout means the log will output to the standard output stream, and stderr means the log whill output to the standard error streamZ

**Return value**

The configure object (\*this)

**Note:**

If you don’t call this function, the default log file is empty, which means no log will output

* configure & confiugre::**set\_overwrite\_tasks**(const std::string & tasks)

**Parameters**

tasks: overwrite tasks sperated by white space, such as “conductor1 conductor2”, which means conductor1 and conductor2 will be considered as master conductor respectively. There is a special value “RWCAP\_ALL”, which means all conductors will considered as master conductor respectively.

**Return value**

The configure object (\*this)

**Note**

You can use the function to overwrite the <task> field of the input file and the tasks specified in the input file will be ignored.

## class problem\_manager

**Constructor**

* problem\_manager::**problem\_manager**(const configure & config)

**Parameters**

config: a configure class instance, it setups the config enviroment of extracting process. See **confiugre** class for more details.

**Member functions**

* bool problem\_manager::**extract**()

**Parameters**

null

**Return value**

true if extract process successfully, or return false.

**Note**

You should always check the return value.

* const std::string & problem\_manager::**errmsg**() const

**Parameters**

null

**Return value**

Error message

**Note**

If extract() method return false, the error message will be set to indicate what went wrong.

* const std::vector<rwcap\_result> & problem\_manager::**get\_result**() const

**Parameters**

null

**Return value**

Capacitance results

**Note**

The size of the return vector is equal to the extracted master conductors. See **rwcap\_result** class for more information.

* void problem\_manager::**reset**(const configure &config)

**Parameters**

config: reset the problem\_manager’s configure to this one

**Return value**

void

**Note**

If you want to reuse the problem\_manager instance, you can use this function.

## class rwcap\_result

**Constructor**

* rwcap\_result::**rwcap\_result**()

**Member functions**

* const std::vector<item> & **items**() const

**Parameters**

void

**Return value**

Capacitance items. The first item is always the master capacitance, others are couple capacitances. **Item** is a struct type defined in the rwcap\_result class as below.

struct **item** {

std::string name; // conductor name

double cap; // capacitance value

double variance; // capacitance variance

}

* const std::string & **master**() const

**Parameters**

void

**Return value**

Master conductor name for the extracting task

* double **cost\_time**() const

**Parameters**

void

**Return value**

Time costed for the extracting task

* long long **nwalks**() const

**Parameters**

void

**Return value**

Totalwalk number for the extracting task

* long long **nhops**() const

**Parameters**

void

**Return value**

Totalhop number for the extracting task

## Example