

# A Practical and Efficient Integrated System for VLSI/ULSI Physical Design \*

Yu Hu, Tong Jing, Xianlong Hong, Qiang Zhou, Ming Shen

Tsinghua University, Beijing 100084, P. R. China

Email: matrix98@mails.tsinghua.edu.cn

**Abstract**--This paper studies and implements an integrated system for VLSI/ULSI physical design. The hierarchy mechanism is used to get great efficiency and speedup. The physical design system is integrated in three different layers, which are called the data interface layer (DIL), the data management layer (DML) and the data display layer (DDL). A quad-tree data structure is used in DDL to accelerate the graphics showing and the interaction between users and the system.

**Keywords:** physical design, system integration, hierarchy mechanism, quad-tree

## I. INTRODUCTION

In recent years, the very large scale integrated circuit / ultra large scale integrated circuit (VLSI/ULSI) technology has profoundly advanced [1]. To meet the needs of advanced integrated circuit (IC) design, the electronic design automation (EDA) tools have been developed to provide more and more different kinds of functions in very large scale. We can see that physical design has been performed by subdividing it into several sub-processes such as partitioning, floorplanning, placement, resource estimation, global routing, layer assignment, crosspoint assignment, and detailed routing, etc.

Each designer in the physical design phase may only be in charge of his own algorithms for his own sub-processes. However, he usually wants to test his algorithms and needs the results of previous processes. He also wants to know the performance of his algorithms or check out bugs in his programs with the help of a friendly graphic user interface (GUI). So, a powerful integrated EDA system instead of many separate tools is very useful for all designers to perform all the chip design processes and review whole physical design results. That is, it needs an integrated EDA system for physical design to contain the whole design flow and support the cooperation of all sub-processes, which is very useful for all the IC designers and the cooperation among them.

The work of integrating a VLSI/ULSI physical design system is related to some technologies in the software

engineering and system design. Some previous works in these aspects have been done. Yun-Fu Cao *et al* [2] proposed a paradigm of software development for mass customization on the basis of domain specific engineering, component based software engineering and concurrent engineering. Shuan-Zhu Du *et al* [3] presented an interface component relation chart based on the interface component relationship feature. In EDA software development, Ding-Jun Chen *et al* [4] designed a circuit behavior simulation editor, which provides a software workbench for software developer to debug embedded software. Qiang Zhou *et al* [5] introduced an integrated solution based on the extensible physical design data markup language PhyD-XML. Narendra V *et al* [6] gave an interval-based approach to the problem of designing a database for the non-partitioned routing problem, which manages the memory efficiently.

The main contribution of this paper is the design and implementation of an integrated VLSI/ULSI physical design system. The rest of this paper is organized as follows. In Section II, the hierarchy mechanism is introduced. In Section III, the strategies and approaches used in the system integration are discussed in detail. Section IV shows some experimental results and the GUI.

## II. THE HIERARCHY MECHANISM

When we develop a physical design integrated system, we may face the following questions.

- 1) How to divide the whole project into several sub-modules so that the team members can work together efficiently?
- 2) How to make all of the sub-processes cooperative against the different output file format?
- 3) How to provide a uniform GUI to display the results of all sub-processes?
- 4) How to handle the ultra large scale data with the minimal run time?

However, designing the system with the hierarchy mechanism is a good solution.

In fact, the IC physical design itself is a hierarchy process. Meanwhile, in the recent decades, the increase of the IC scale has exceeded the capability of physical design tool development, which makes it is important to design more powerful tools. The hierarchy mechanism can minimize the problem scale, shorten the developing period, and improve the robustness of the system. Further more, it

---

\* This work was partially supported by Hi-Tech Research and Development (863) Program of China under Grant 2002AA1Z1460, National Natural Science Foundation of China under Grant No.60373012, and SRFDP of China under Grant No.20020003008.

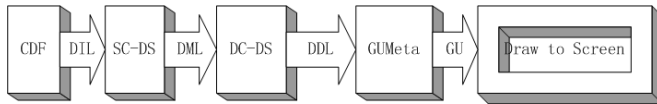
provides a platform that all designers can work together for one chip but use different tools and functions and in different layers [7].

We integrate the whole layout system in three different

layers, which are called the data interface layer (DIL), the data management layer (DML) and the data display layer (DDL), respectively. Fig.1 shows the relationship among them.

Table 1: Views and graphic meta data in each sub-process

Sub-process	View	Graphic Meta Data
Placement	Placement Result	Cell, Source Pin, Sink Pin
	Sites	Site
	Over Connections	Cell, Net over cell
Resource Estimation	Global Routing Grid	Cell, GRGrid
	Channel Resource Report	Source track
Global Routing	Global Routing Result	Net connection, GRGrid
	Resource Usage	Source track, Used track
Layer Assignment	Resource Usage in each layer	
Crosspoint Assignment	Crosspoint Assignment Result	Crosspoint, Cell, Source Pin, Sink Pin, GRGrid
	Crosspoints in the given GRC	
Detailed Routing	Detailed Routing Result for all nets	Net path, Crosspoint, Cell, Source Pin, Sink Pin, GRGrid
	Detailed Routing Result for a single net	
	Detailed Routing Result for a given layer	
	The Detailed Routing Result for a given GRC	



- CDF: Circuit Description Files
- SC-DS: SC Data Structures
- DC-DS: Display Components Data Structures
- GUMeta: GU Meta Data Structure
- DIL: Data Interface Layer
- DML: Data Management Layer
- DDL: Data Display Layer
- GU : GU Library

Fig. 1 The relationship among three layers

### III. SYSTEM INTEGRATION

#### A. Data interface layer (DIL)

We use a set of basic data structures and I/O interface in SC system <sup>1</sup> to denote the circuit information in the whole physical design process. The function of DIL is to read the circuit description files and record these files in memory by some kinds of data structures. Unfortunately, there are more than one kind of HDL (hardware description language) at present, and our system should unify these different kinds of formats into ones, which is the SC file format. We've developed some subsystems to convert the Verilog (with LEF) into DEF/LEF format [8], the DEF/LEF into SC format, and the GDSII into SC format, etc. We've also developed a subsystem to convert all the SC physical design sub-processes result files into

<sup>1</sup> SC (standard cell) system is developed by our Lab., which is a platform for designing and testing physical design algorithms. SC provides many kinds of fundamental data structures and interfaces to describe a circuit.

DEF/LEF format to meet the requirements of industrial customers.

#### B. Data management layer (DML)

The function of DML is to convert the SC data structures into the display components data structure. We divide the VLSI physical design into the following sub-processes in our system: placement (consist of floorplanning), resource estimation, global routing, layer assignment, crosspoint assignment, and detailed routing [9]. Every sub-process needs at least one "View" (see Table 1). Designers can switch all different "Views" in the current sub-process. Every "View" consists of its own controller GUI and display meta elements (i.e., circuit elements), which is shown in Table 1. The DML uses the interface provided by DDL to convert the SC data structures into graphic meta data, and then adds these data into display component data structure.

#### C. Data display layer (DDL)

As it's mentioned above, DDL provides a set of interfaces to convert the SC data structures into display components data structure and draw them on the screen. We introduce our display components data structure firstly.

##### 1. Data Structure for Display Components

We use a data structure called "display chain" to organize all graphic meta data in each "View". For some extending requirements of this system <sup>2</sup>, we use the quad-tree to store these display chains, which makes the system run efficiently. The display components data

<sup>2</sup> Such as querying some items in the layout with mouse clicking and editing the layout in GUI.

structure is shown in Fig.2.

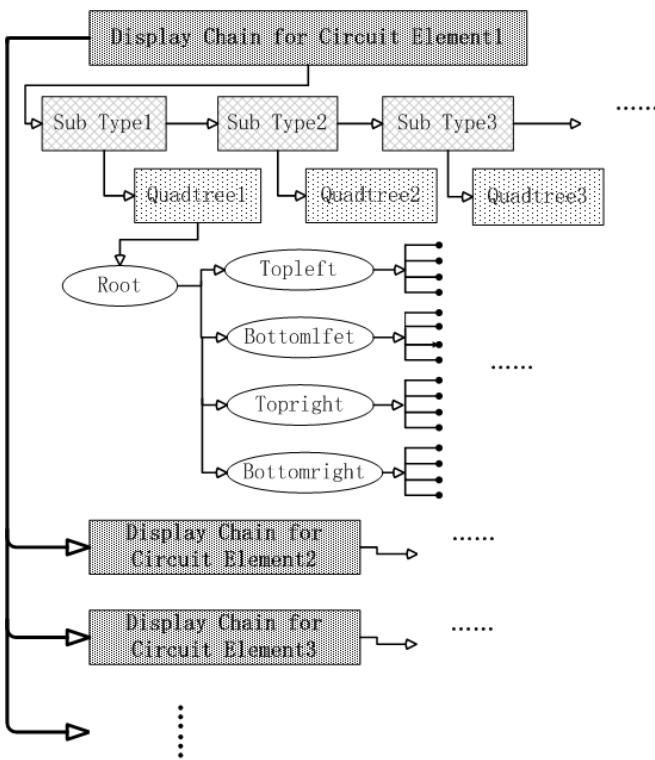


Fig.2 The data structures for display components

From Fig.2, we can see the whole system is made up of some display chains, and each of them denotes a kind of circuit element (such as cell, port, net etc.) which is also a kind of graphic meta data in a “View”, and the display chains record these circuit elements by their topological types (such as cell by rectangle, net by segment, and pin by polygon, etc.). We can use a uniform interface to display all kinds of circuit elements by their topological types. Note that each of the display chains contains more than one “sub-type” because some attributions of some kinds of circuit element is different<sup>3</sup> and each sub-type records its own special attributions. Moreover, sub-type records the geometrical information (such as width and height) of the circuit elements belong to it. The sub-type is based on quad-tree data structures that will be described in detail in the next sub-section.

### 2. The Quad-tree Data Structure

Customers of our system need to query some circuit element by simply clicking this element in GUI using a mouse. For the efficient implement of the requirements like this, we use the quad-tree data structure to shorten the searching time, which makes the GUI of the system be very friendly.

<sup>3</sup> For an instance, in multi-layer routing, the net segments after detailed routing will be in different layers, and we must use different colors to distinguish segments in different routing layers in the GUI “View”.

The quad-tree is a data structure to organize multi-objects in two-dimension, which can accelerate the search for the given object. We construct a quad-tree under the following rule.

In a given two-dimension plane area, we add all objects in this area into a chain list if their number are not more than a threshold which is given as constant in the beginning. Otherwise, we divide the area into four uniform sub-areas, reorganize the objects in the chain list by putting them into the chain list of the sub-area they belong to, and leave the objects that belong to none of the four sub-areas in the old chain list. We can add all objects into a quad-tree under this rule recursively. A simple example for a quad-tree construction is shown in Fig.3.

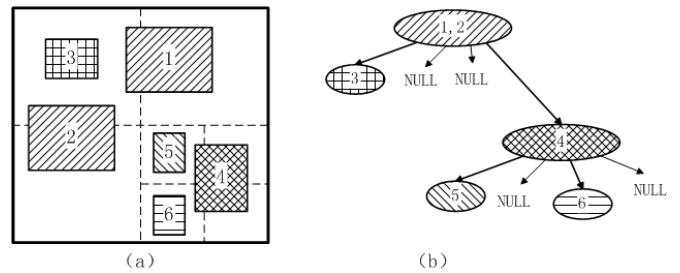


Fig.3 There are 6 objects in the plane area shown by (a)

The threshold is 2 (that is to say, there are not more than 2 objects in one chain list). As the rule given above, object 1 and 2 belong to the root node, because they are in the separate line between the sub-areas. Object 3 belongs to the top-left area node and object 4-6 belong to the bottom-right area node. The number in this sub-area is more than the threshold, so we divide this sub-area into 4 sub-sub-areas. Then, put object 5 and 6 into the sub-sub-area nodes they belong to. The result of quad-tree is shown in Fig.3 (b).

### 3. The efficiency of the Quad-tree Search

We can use the linear chain list to store the display components, but its slow searching speed makes us use the quad-tree instead of it. We’ll show this in the following paragraphs by analyzing the complexities of them.

We consider the mouse-query<sup>4</sup> function firstly. The basic problem is how to find an object by its location.

**Problem 1:** Given a circuit that consists of  $m$  cells,  $m_{sourcePin}$  source pins and  $m_{sinkPin}$  sink pins. The graphic meta data in the current “View” is cells, source pins and sink pins. Consider the situation that an object in the last display chain list (sink pin) is selected. What is the complexity of using linear chain list as a data structure and using quad-tree respectively?

It’s a linear algorithm to search objects using a linear chain list, and we must travel all over objects in the cell chain, in the source pin chain and some in the sink pin chain (in the worst case, we select the last object in the sink pin). So in the worst case, the total compare times is:

<sup>4</sup> Get information of some elements by using mouse in GUI.

$$CN_{chain} = (m + m_{sourcePin} + m_{sinkPin}) \quad (1)$$

In general, the number of source pins and sink pins is  $O(m)$ , we can rewrite equation (1) as:

$$CN_{chain} = O(m) \quad (2)$$

Now, we consider the situation of using quad-tree. Obviously, we can use at most  $h$  times to finish a search in a quad-tree with the height of  $h$ . So we can find an object in a  $N$ -objects list using quad-tree within the following compare times:

$$h = O(\log(N/T)) \quad (3)$$

where  $T$  is the threshold of the quad-tree node.

From equation (2), the total comparison times in the worst case is:

$$\begin{aligned} CN_{quadTree} &= (O(\log(m)) + O(\log(m_{sourcePin})) + O(\log(m_{sinkPin}))) \\ &= O(\log(m)) \end{aligned} \quad (4)$$

Given a circuit consists of 100,000 cells ( $m = 100,000$ ), the comparison times of using linear chain list and quad-tree is about 100,000 and 10 respectively from equation (2) and equation (4). The search speed of linear chain list is 10,000 times slower than the quad-tree, which make the system much more sensitive to users.

When display a layout of large scale circuit, the slow refreshing often make users boring. We now consider the advantage of the quad-tree over the linear chain list in screen refreshing. Sometimes the users hope to view the whole layout, which is not necessary to draw out all the elements in the circuit since some tiny elements can't be seen at all in such a scale. So we can minimize the shown elements to speedup the refreshing.

**Problem 2:** Given a circuit consists of  $m$  cells, and the minimum size can be seen in the current scale is  $p$ . What's is the comparison times to redraw the whole layout using linear chain list and quad-tree respectively?

By using chain list, we must travel the whole list to get all objects bigger than  $p$ . Obviously, the total number of comparison times is:

$$DN_{chain} = O(m) \quad (5)$$

By using quad-tree, we don't need to travel the rest of the sub-plane when we find some node is smaller than  $p$ <sup>5</sup>. We call this strategy "Prune". Because the node with the same height in the tree denotes the same area size, we can travel only the whole tree with the deep  $D/p$ , where  $D$  is the area size of the whole layout. Thus, the total number of compare times is:

$$DN_{quadTree} = O(4^{D/p}) \quad (6)$$

Suppose the minimum size of the elements in this circuit is  $p_m$ , the height of the quad-tree is  $D/p_m$ . The node number of the quad-tree is proportional to the number of objects in the quad-tree, so we have:

$$m = O(4^{D/p_m}) \quad (7)$$

<sup>5</sup> As we mentioned before, each node in the quad-tree denotes a plane area, and its every sub-node denotes the quarter of its area. So the sub-tree rooted by a node smaller than  $p$  doesn't need to be traversed any more.

From equation (5) and equation (7), we have:

$$\frac{DN_{chain}}{DN_{quadTree}} = O(4^\alpha) \quad (8)$$

where  $\alpha$  is:

$$\alpha = D \cdot \frac{p - p_m}{p \cdot p_m} \quad (9)$$

The large scale of the circuit makes  $p \gg p_m$ , which makes  $DN_{chain} \gg DN_{quadTree}$ . That is to say, the refreshing using linear chain list is slower than the one using quad-tree. When the scale of the circuit is over 100,000, the slow refreshing using linear chain list is unendurable.

#### 4. The Interface of DDL

The function of DDL is to provide a set of uniform interfaces for DML, which converts the SC data structure into display data structure. Besides this, we also implement a basic class to manage the quad-tree data structure, which provide the basic function such as inserting an object into a quad-tree, deleting a object from a quad-tree, traversing all objects in a quad-tree, and finding a given object in a quad-tree, etc. The basic idea of constructing and managing a quad-tree has been explained in sub-section 2.

The basic interfaces that should be provided by DDL are shown in Table 2:

Table 2: Basic interfaces in DDL

Interface Name	Function Description
AddDispFigure	Add an object into display chain
AddDispObject	Add a new display chain
CleanDispObj	Delete all objects in a display chain
DrawObject	Draw all the objects in a display chain <sup>6</sup>
GetObjectByXY	Get the information of a given object by the mouse position

## IV. EXPERIMENTAL RESULTS AND CONCLUSIONS

By using the hierarchy mechanism, we divide the whole system into some relatively independent sub-systems, implement such sub-systems respectively, and assemble them into an integrated system. The quad-tree data structure helps us speedup the objects searching.

The placement result is shown in Fig. 4, in which the small objects have been skipped by the "Prune" method. We can see that there is nothing wrong in the pruned result, except of fast refreshing.

Fig. 5 shows the detailed routing results in one global routing grid. Users can change the color and visibility of a given routing layer as they like. If a user wants to get the information about a via, he can click the mouse over this via in the layout graph. This user can see the selected via is

<sup>6</sup> We use a graphic display interface named GU to help draw the basic graphic components. GU is a GDI library based on tcl/tk. We implement the DrawObject interface by converting our display data structures into GU's data structure, and using GU's refresh function to draw objects into the screen.

highlight with a hollow black rectangle, which is also shown in Fig.5.

the mouse-query by the quad-tree is much faster than by the linear chain list.

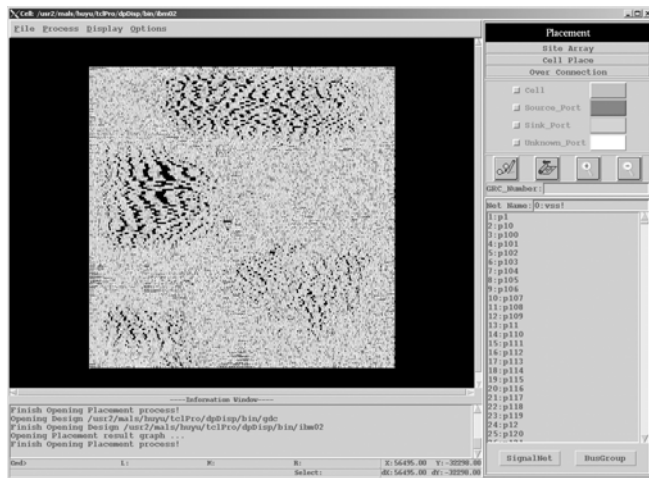


Fig.4 Placement View after “Prune”

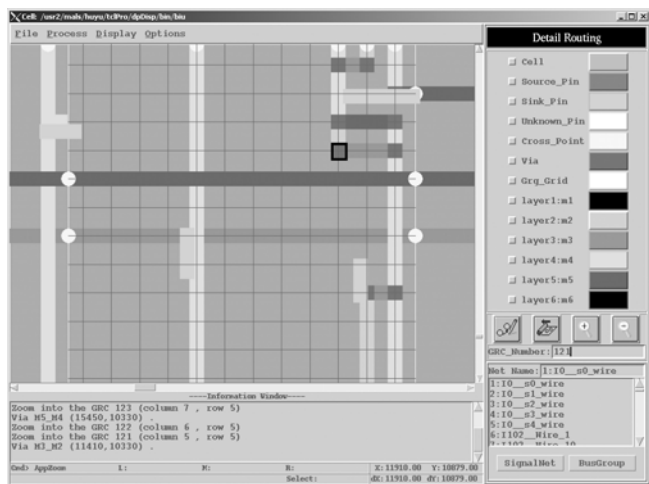


Fig.5 Detailed Routing View with a via selected by mouse

Table 3: Mouse-query time (quad-tree vs. linear chain list)

Circuit Name	Cell Number	Quad-tree Time	Linear Chain List Time
C2	1000	55	20972
C5	1500	92	38157
C7	2000	212	42174
BIU	1200	104	50234

Table 3 shows the average CPU time of mouse-query in four test cases (three MCNC benchmarks: C2, C5, and C7, an industrial case: Biu) by using linear chain list and quad-tree, respectively. All results are obtained from a Sun-fire880 workstation and Solaris V5.0. Each instance is tested for ten times. The time unit is  $10^{-6}$  s. We can see that

Table 4: The refresh time using “Prune”

Name	C-scale	$u=1$	$u=1/2$	$u=1/4$
C2	1000	14822	8754	7445
C5	1500	18932	15066	14423
C7	2000	28914	22981	22446
IBM01	10000	252758	239312	183021

Table 4 shows the refresh time to draw the placement results of four test cases (three MCNC benchmarks: C2, C5, and C7, an industrial case: IBM01). The time unit is  $10^{-6}$  s.  $u$  is the display scale. We can see that cases using “Prune” are faster when  $u$  is getting smaller.

## REFERENCES

- [1] Tong Jing, Xian-Long Hong, Yi-Ci Cai, *et al*, “Data-Path Layout Design inside SOC”, In: *Proc. of Communications Circuits and Systems and West Sino Expositions (ICCCAS)*, Chengdu, China, 2002: pp.1406-1410.
- [2] Yun-Fu Cao, Jun-Wen Zhao, Yong-Sheng Han, *et al*, “A Paradigm of Software Development for Mass Customization”, *J. of Computer Research and Development*, 39(5): pp.593-598, 2002.
- [3] Shuan-Zhu Du, Jian-Rong Tan, and Guo-Dong Lu, “Software Functional Testing Technology Based on Interface Component Relating Chart”, *J. of Computer Research and Development*, 39(2): pp.148-152, 2002.
- [4] Ding-Jun Chen, Xiao-Dong Guo, Zheng Xu, *et al*, “The Design and Implementation of a Circuits Behavior Simulation Editor”, *J. of Computer Research and Development*, 36(7): pp.882-887, 1999.
- [5] Qiang Zhou, Qi Cai, Xianlong Hong, *et al*, “Design and Implementation of VLSI Physical Design Data Markup Language PhyD-XML”, *J. of Computer-Aided Design & Computer Graphics*, 15(7): pp.773-777, 2003.
- [6] Narendra V. Shenoy, William Nicholls, “An Efficient Routing Database”, In: *Proc. of Design Automation Conference (DAC)*, New Orleans, Louisiana, USA, 2002: pp.590-595.
- [7] [http://www.eetchina.com/ART\\_8800140572\\_617681\\_617682.HTM.a232e899](http://www.eetchina.com/ART_8800140572_617681_617682.HTM.a232e899).
- [8] Yu Hu, Tong Jing, Xian-Long Hong, *et al*, “Data Management for Data-Path Layout System”, *J. of Microelectronics*, 33(4): pp.301-305, 2003.
- [9] X. L. Hong, X. L. Yan, and C. G. Qiao, *The Theories and Algorithms for VLSI Layout Design*. Beijing: Science Press, 1998.