계층적 설계 환경에서 일관된 타이밍 분석을 위한 분할 및 제한 조건 생성 기술 개발

한 상 용†

요 약

VLSI의 집적도가 계속 증가되고 있어 복잡한 칩 설계를 위해서는 설계의 계층성 이용이 매우 중요하다. 계층설계는 대규모의 설계 데이터를 기능의 계층성을 이용하여 분할 설계하기 때문에 오랫동안 이용되어 왔다. 그러나, 계층 설계에서는 분할 설계후 다시 통합하기 때문에 원래의 설계 데이터와 분할·통합한 설계 데이터 사이에 타이밍 분석 결과의 차이가 발생할 수 있고 이는 칩 개발 시간을 지연시키는 주요 요인이 된다. 본 논문에서는 계층설계에서 타이밍 문제를 공식화하였고, 타이밍 분석시 flat설계와 차이가 나는 원인들을 분석하였다. 일관된 타이밍 분석이란 개념을 정의하였고 일관성유지를 위한 분할 기법을 제안하였으며, 제안한 알고리즘을 구현하여 기존의 설계툴에 접목하여 일관성 향상을 얻었다.

Partitioning and Constraints Generation for the Timing Consistency in the Hierarchical Design Method

Sang-Yong Han†

ABSTRACT

The advancements in technology which have lead to higher and higher levels of integration have required advancements in the methods used in designing VLSI chip. A key to enable a complicated chip design is the use of hierarchy in the design process. Hierarchy organizes the function of a large number of transistors into a particular, easy- to-manage function. For these reasons, hierarchy has been used in the design process of digital functions for many years. However, there exists differences in a design analysis phase, especially in timing analysis, due to multiple views for the same design. In timing analysis of the hierarchical design, every path is analyzed within partitioned modules independently and the global timing analysis is applied to the whole design considering each module as a single timing component. Therefore, timing results of the hierarchical design could not be same as those of non-hierarchical flat design. In this paper, we formulate the timing problem in the hierarchical design and analyze the possible source of timing differences. We define a new terminology of "consistent result" between different views for the same design. We also propose a new partitioning algorithm to obtain the consistent results. This algorithm helps to enhance the design cycle time.

* This study was supported by the academic research fund of Ministry of Education, Republic of Korea through Inter_ University Semiconductor Research Center(ISRC 97-e-2010) in Seoul National University

† 정 회 원 : 중앙대학교 컴퓨터공학과 교수

논문접수 : 1999년 9월 30일, 심사완료 : 1999년 12월 10일

1. Introduction

The early design methodology was flat in both the synthesis and physical design environments. Even though, the logic written in VHDL or Verilog was done on a partition basis, the entire design was flattened into a single piece before synthesis was performed. The single file was then taken to physical design and placed and wired as a single entity. This process was very flexible in supporting design changes and easy for a timing analysis. But, a continuous trend in the design of VLSI chip is the increase of circuit size accompanied by decrease of feature sizes and we are now facing the design of up to ten million transistors. In the new design environments, abstraction and hierarchy is a key to manage complex technical and organizational problems.

The intuitive notion of hierarchy is simple. One decomposes a large problem into a number of smaller parts. While each of the parts can be expected to exhibit only limited complexity, it is the expectation that the integration of the parts will not lead to a significant increase. Such hierarchical decomposition techniques have long been in use for IC design supported by CAD tools. Many of these were devised for obtaining a reduction of the amount of the design data. However, hierarchy affects the number of interactions between components to be taken into account as well as the number of configurations to be considered during design. Interface and consistency in the hierarchy design is very serious and critical in the timing analysis [8].

There are two different kinds of nets in the hierarchical design. Some nets belong to the same partition and others go to more than one partition. We refer to these as local and global nets respectively. Since the delays of global nets are not known during timing analysis of individual partitioned module, they could be the source of the timing differences between two design views [6]. We formally analyze the timing difference problem and formulate the concept of "consistent result" and observability. In addressing the consistency problem, we also propose the partitioning algorithm to guarantee the consistency of timing results between different views of design.

Partitioning and placement in the physical design phase has mostly been concerned about obtaining a wirable layout rather than timing accuracy and performance. Recently several approaches have been developed which optimize the timing and performance but timing consistency [9]. We now introduce a new approach whereby the partitioning process is being governed by the ability to guarantee the timing consistency between the flat design and hierarchically partitioned design.

Section 2 will introduce hierarchy and the basic ideas behind it. Hierarchy as it is presently exploited in VLSI design will be the subject of discussion, leading to problems as they can be observed in practice. In this section we also discuss the nature and origin of timing constraints and its relation to VLSI design.

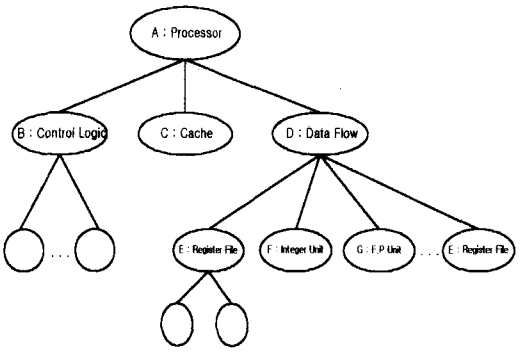
In section 3, we formulate the timing problems in the hierarchical design and define the concept of "consistent result" and "critical observability". Nets are also classified based on their position in timing analysis. Section 4 will investigate the partitioning problem to satisfy the timing requirements to guarantee the consistent result. Characteristics of suppressed paths are examined and timing constraints are also discussed. Partitioning algorithm is given and its complexity is discussed. Finally, section 5 will discuss the experimental results of the proposed algorithm.

2. Hierarchical timing analysis model

2.1 Flat design vs. hierarchical design

The basic semi-custom approach used by the industry is a hierarchical design methodology. In a hierarchical methodology the VLSI design is split into a tree-like hierarchical structure, each node of which is a subdesign to be built. Each design on the node of the tree becomes one of the building blocks for the parent node of the design (see (Fig. 1)). In this way, complex VLSI design may be built up from less complex underlying objects.

There are two major advantages of a hierarchical



(Fig. 1) hierarchical structure

design over flat design. The first one is easy to express a particular style of a design and the second is to reduce the problem size. A common choice of child-child hierarchical boundaries in VLSI design is terrain. A terrain is a region of a chip that expresses a particular style of physical design. Examples of terrains are random logic, arrays, data path, etc. One of the difficult problems with a flat design is generating placement and wiring algorithms that operate well on different styles of logic. For instance, random logic region and data flow regions require a completely different style of wiring. By choosing hierarchical boundaries such that different terrains are contained in separate hierarchical objects, specialized algorithms may be individually applied to each object.

Hierarchy is also used to reduce the problem size for VLSI designs. Both run time and storage requirements for flat designs are rapidly becoming unmanageable as transistor counts climb from the hundred's of thousands to the millions transistor and above range. Building a hierarchical model of the design allows run time and storage requirements for processing the hierarchical objects to be much less than that of a flat model. The flat model tends to force the designer into an "all or nothing" approach to the design. The hierarchical model lends itself easily to incremental design without the complexity of managing incremental update capability in all the individual tools [1].

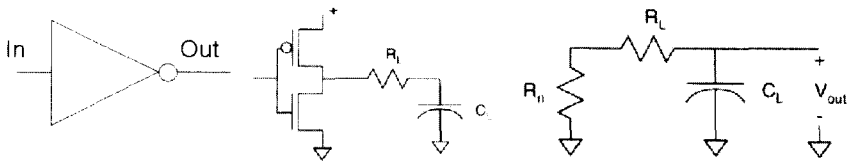
But, the use of a hierarchical design model introduces some new problems. The major problem encountered is the increased data model complexity of the design and timing analysis consistency between two views (flat and hierarchical view) of the same design. Instead of a single representation of the design, we were required to manage a complex hierarchy of these representations and guarantee on final build that we have assembled all the correct versions and our timing analysis is consistent.

2.2 Timing analysis

The simple inverter circuit is shown in (Fig. 2). The load on the inverter is a single resistor-capacitor (RC) circuit; the resistance and capacitance come from the logic gate connected to the inverter's output and the wire connecting the two. While the circuit in (Fig. 2) has only a few components, a detailed analysis of it is difficult due to the complexity of the transistor's behaviour. Typically a single equation is used to describe the behavior of a circuit, both in terms of cell pin-to-pin delay as well as the transition time of the signal at the output pin. The typical delay equation [2] is of the form

$$r = f(t_r, C_l) = A*t_r + B*C_l + C*t_r*C_l + D \qquad (1)$$

The circuit delay, r, is a function of the input transition time, t_r, and capacitive loading, C_l, on the output port.



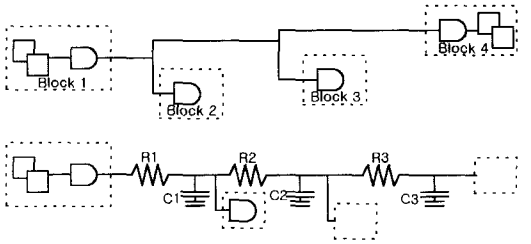
(a) gate model (b) inverter (c) delay model

(Fig. 2) The simple inverter circuit

Characterization constants, A thru D, is predetermined and stored in the table. Characterization data is used by synthesis and timing verification.

Analyzing the delay of a single gate isn't sufficient to know the delay through a complex network of logic gates. For the purpose of a timing analysis, a synchronous system can be viewed as a collection of blocks and nets interconnecting them. We will distinguish four types of blocks, namely; combinational, synchronizing (storage, registers), primary inputs (PIs) and primary outputs (POs). The signals in the system originate in the PIs or synchronizing elements, travel through the combinational blocks and wires interconnecting them and terminate at the POs or synchronizing elements. Thus, loosely speaking, the timing analysis problem is to determine that all the PI/storage - - PO/storage path delays do not exceed the pre-specified values.

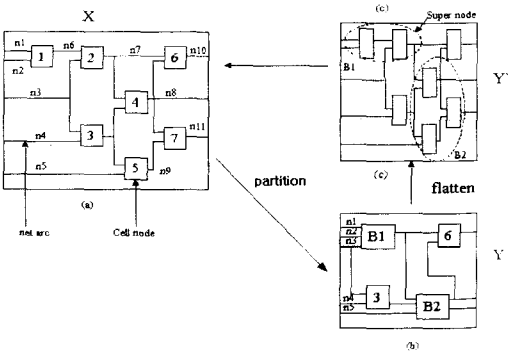
(Fig. 3) shows an example of gate and circuit model of the partitioned combinational network. It is important to maintain the signal integrity for long global interconnect line which traverses multiple blocks. Assigning proper timing budgets to each partitioned block is of major concern in a timing analysis of a hierarchical design.



(Fig. 3) Circuit model of global wire for combinational logic

2.3 Partitioning

A graph G = (V, E) in (Fig. 4(a)) consists of 7 nodes and 11 edges. Each node in (Fig. 4(a)) might represent a transistor, a cell, or a custom circuit. Therefore, a graph in (Fig. 4) is also a hypergraph. A hypergraph H = (V, E) consists of n nodes and m hyperedges. A hyperedge e in E is defined by a subset of nodes Ve in V. Circuit partitioning is abstracted and formalized



(Fig. 4) flat net list and 2 level partition net list

Partition : X → Y, X is a hypergraph and so is Y.
Flatten(Y)= Y' is identical to X, i.e.,
h: Y'→X is a topological, functional isomorphism.

as an operation on hypergraphs. An exact or an approximation algorithm for hypergraph partitioning is also an exact or an approximation algorithm for circuit partitioning. Thus, the algorithms described in this paper for hypergraph partitioning apply for circuit partitioning as well. (Fig. 4) shows a typical example of "partition" and "flatten" process. In a partitioning process, node 1 and 2 are merged. Also, nodes 4, 5, and 7 are also combined as one super node. The above partitioning process makes a new hypergraph with a netlist Y which has four nodes. Net 6 in X does not exist anymore in a new netlist Y. The design process, synthesis and analysis, is applied to a new netlist Y. Sometime later, Y should be back to its original form. We call this process as "flatten". A new netlist Y' in (Fig. 4) is a netlist after flattening Y. Formal relation between three netlists are defined as follows;

3. Problem Formulation

Traditional design flow starts with micro-architecture development with performance as a goal. A textual description is then completed, and this is followed by RTL. RTL generation phase places more emphasis on functionality than timing. It's only during the late circuit

design phase we get a reasonable exposure to full chip timing in any detail. This results in a significant amount of time and resource being spent after silicon is available, trying to fix critical paths. In this post-silicon timing design mode, since the die size, floorplan, and architecture are fixed, there is very little flexibility to correct these timing errors. In theory, complete layout data is required to do a high confidence timing analysis. We are actually in a "chicken and egg" situation. So we need to start timing estimations early and accurate, and continue to refine them.

In a typical timing analysis of a hierarchical design, timer is applied to each partitioned block with timing budget and local RC information and is also applied to the parent block with child blocks represented by black blocks. This step continues as we reach the top level. In this process, there exists timing differences compared to the method as would the application of a timer on a flat design. Assume that there exists a path p in X and q is the corresponding path in Y' in (Fig. 4). There might be a delay difference due to different view of the same path. Relation between these paths are formally defined as follows;

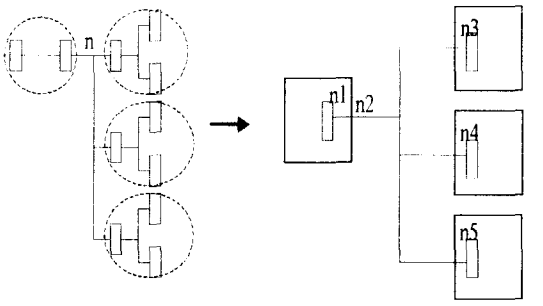
$$\varepsilon = \max \mid \text{delay}(p) - \text{delay}(q) \mid$$
$$\forall p \text{ in } X$$
$$\text{Consistent Result} = P_{observability} =$$
$$\forall [p \in P_{critical} \rightarrow q \in Q_{critical}, \mid \text{delay}(p) - \text{delay}(q) \mid \leq \varepsilon]$$

Now our problem can be stated as follows; *given a timer T, a flat netlist X, how to partition X into sub_blocks, such that the application of T on the sub_block and on the top level block with the sub_blocks represented by black boxes yields "consistent result" as would the application of T on X.*

If the sub_block boundaries cut net n into i segments, write $n = n_1 \mid n_2 \mid \ldots \mid n_i$ as in (Fig. 5).

We now show relationship of timing results on the application of timer at the partitioned circuits.

Define $T(n_1|n_2|....|n_i) = T(n_1) \cdot T(n_2) \cdot ..... \cdot T(n_i)$, where $T(n_i)$ is a piecewise evaluation. Then, the problem formulation can be written as



(Fig. 5) a net is divided by five segments

$$|T(n)-T(n_1|n_2|....|n_i)| < \varepsilon \qquad (2)$$

Equation (2) shows accuracy of piecewise evaluation as ε.

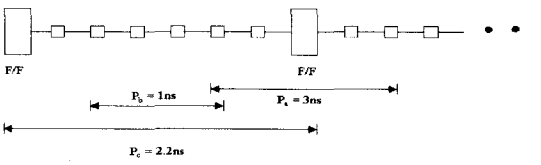
As we see in the preceding analysis, some critical paths are hidden on the application of timing analysis of the hierarchical design. As these critical paths are hidden until tapeout, it represents the worst case scenario in VLSI design. Also paths that are shown as critical but not true in a flat design represent false paths. These paths are also worrisome as lots of resources are wasted to correct non-critical paths. Section 4 propose a new algorithm to main the consistent results between the hierarchical and flat design.

4. EACPP Algorithm

4.1 Classification of paths in a F/F based design

A nontrivial path, p, is a complete path iff source terminal of p is either a chip pad or an output of a F/F and a sink terminal of p is either a chip pad or an input of a F/F. (Fig. 6) shows an example of complete and incomplete paths. P_a and P_b are not a complete path, but Pc is so as defined above.

Now, we will show the meaning of a complete path



(Fig. 6) example of complete and incomplete paths

in a timing analysis. Assume that clock cycle time is 2 ns and total delay of P_c in (Fig. 6) is 2.2 ns. If a clock signal arrives at the source F/F 0.1 ns earlier and arrives at the sink F/F 0.1 ns late, then a path P_c is not a critical path on a flat timing analysis. But, if a source and a sink F/F is partitioned into different modules, then as a skew in the source F/F is hidden P_c might be classified as a critical path in the hierarchical timing analysis. Therefore, a complete path is the key to the observability and we derive the following definition.

A complete path, p, such that $p=t|q$ or $q|t$ where t is a trivial path, then q is called an *almost complete path*. A similar classification is used for paths in sub_blocks, too. A nontrivial path, p, in sub_block is a *full path* iff source terminal of p is either an input of a partitioned block or an output of a F/F and sink terminal of p is either an output of a partitioned block or an input of a F/F.

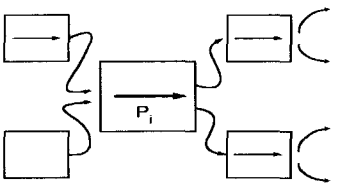
Assume that q is a path in an original flat design and a critical path. Depending on the partition structure, q can be easily observed in the partitioned design or can be the other way. In the following section, we define properties or structures that are required on partition such that q can be observed on a sub_block timing analysis, $T(B_i)$, and a global timing analysis, $T(Y)$.

4.2 Constraints generation

Run timing analysis on the partitioned logic block requires two constraints, input arrival time and output required arrival time. Usually the constraints are generated a number of times as designs of each individual block progress during the layout steps so that the interactions between the blocks can be reflected. We refer this concept as the dynamic timing constraints. The dynamic timing constraints generate a dynamic set of constraints to enable the design to converge a better optimal solution. The earliest time that constraints are to be discovered is at the global timing analysis after the first iteration of timing analysis of every sub_block in a design. The procedure to allocate and generate timing constraints (or timing budget) is outlined as follows;

1. *run timing analysis on each block independently.*
2. *run global timing analysis using the results of step1.*
3. *allocate timing budget on each block.*
4. *apply timing analysis tool on the first block and identify the critical nets. designers fix the critical paths. Update the constraints.*
5. *select a block where all the constraints are available. Do step 4.*
6. *repeat step 5 until the convergence test is satisfied.*

As we see in (Fig. 7), to generate "proper" constraints for P_j, we need knowledge of all connected timing paths. As the constraints on each sub_block are consistently updated, assigning proper constraints is not an easy problem. The specification should be proper and consistent and path correction of one sub_block should not overkill at the same time.



(Fig. 7) Sensitivity of constraints

From this analysis we can conclude that a timing methodology which must maintain/update a large set of *complex constraints* has low probability of achieving observability. If observability is not achieved, undiscovered critical paths will show up sometime after tapeout and the complexity of a timing methodology depends on partitioning.

A constraint specification is called a *simple constraint* iff it can be "properly" generated before timing analysis, i.e., before $T(B_i)$ & $T(Y)$. A constraint specification is called a *complex constraint* iff it can only be properly generated after some i-th iteration of $T(B_i)$ & $T(Y)$.

4.3 Partition complexity

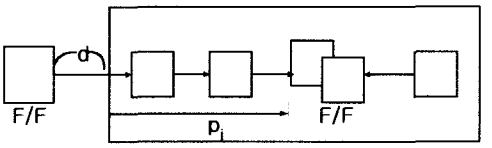
Given par:X → Y, let p be a complete path in Y. If partition cuts p into i segments, then observing p requires (i-1) "proper" constraint specifications. Define

$$m(par) = \sum_{P \in P_{complete}} |P|_{cuts}, \text{ where } |P|_{cuts} =$$

$$\begin{cases} 0 \text{ if par does not cut a path p} \\ i-1 \text{ if par cuts p into i segment} \end{cases}$$

Therefore, m(par) becomes the number of constraints that must be generated/maintained/updated by a timing methodology which employs par:X → Y. We use m(par) to measure the complexity of par:X → Y such that the optimal par has m(par)=0, i.e., which is a trivial partition.

Even though a path p_j in (Fig. 8) is not a complete path, specifying a delay, d, effectively moves the block boundary for p_j back to the output pin of the F/F. Therefore, p_j is essentially a complete path even though it is classified as an almost complete path. Therefore, the constraint for an almost complete path is a simple constraint since it can be generated prior to a timing analysis.



(Fig. 8) example of an almost complete path

4.4 Everywhere Almost Complete Path Partition (EACPP)

If every full path in X is either complete or almost complete, then a partition, par:X → Y, such that every full path in each sub_block, B_i, is an almost complete path is called an everywhere almost complete path partition (EACPP) and B_i is called an almost complete path sub_block. EACCP has the following properties;

1. m(par) <= $|P_{complete}|$.
2. all constraints are simple constraints and every sub_block is independent.

Let denote Critical(T(A)) = {p | p is critical in A and is found by T(A)}. Then the following independence lemma holds. The lemma shows the condition that no global timing analysis is required, which means X can be analyzed independently by analyzing each sub_block, Bi.

Lemmal (Independence)

let par:X → Y is an everywhere almost complete path partition that divides the design into n sub_blocks. Let them be B_1, B_2, . . B_n. If the application of all simple constraints on each Bi is such that critical(T(B_i')) = ∅ ∀i ⇒ critical (T(X)) = ∅ or critical(T(Y))=∅.

Proof

Assume for all i, Critical(T(B_i')) = ∅. If Critical(T(X)) ≠∅, then there exists q that is critical in X such that $h^{-1}(q) = p$ in Y′ is critical. By the definition of par:X → Y, $p = t|p'$ or $p'|t$ where p' is an almost complete path in some B_j in Y. Now the application of the simple constraint, delay(t), on p' effectively makes it a complete path. So if p is critical, T(B_j') would have found it, i.e., Critical(T(B_j')) ≠∅ which contradicts earlier assumption. Therefore, it must be the case that Critical(T(X)) = ∅. ■

Observe that if B_i has an everywhere almost complete path partition(EACPP), then B_i can further be decomposed into smaller, independent subproblems. But, there are many factors that hinder decomposing the circuit into EACPP. Careless or random decomposition might lead to uneven(too large or too small) or unrelated or incompatible sub_blocks. To achieve a balanced EACPP, design hierarchy and partition must be considered or planned in the early design phase. The main factor which hinders the creation of a balanced EACPP is a clock signal.

Assume all data paths in B are almost complete or complete. To make B into an almost complete path sub_block, we need to specify constraints on the input clock pins to B so that all clock paths become effectively complete. The clock constraints are complex constraints, which means that they can only be generated after T(clock network). Therefore, we separate the clock and data paths and generate all the clock network delay upto the clock pin on the boundary of each sub_block. EACPP algorithm is shown as follows;

- *abstract clock network from T(Bi′)*
  /* separate the clock logic and data signals */
- *T(clock network) for skew.*
  /* *do a timing analysis on clock signals only* */
- *account for skew.*

```
/* calculate clock skew on each F/F */
• loop /* repeat clustering */
  construct a sub_block with a single cell.
  /* it could be a F/F or any other cell */
  loop
    add a neighbour cell.
    if there exists a full path, p, in a sub_block
    that is neither complete nor almost complete, then extend p
    by adding to Pi other nets and cells until p is either complete
    or almost complete
  until a sub_block becomes an almost complete path sub_block.
until all the cells belong to a block.
```

5. Experiments and Discussion

The main goal of our approach is to achieve the timing closure as early in the design cycle as possible so that we avoid a significant design time increase. The proposed methodology which guarantees critical observability compares well with other physical design tools such as TDP(Timing Driven Placement) [3, 4, 5, 8]. Experiments were done on the Poughkeepsie serial chip using standard EDS tools. Experiments process were to compile all of the VHDL codes into a single entity. The logic is partitioned into segments of almost equal size. Before the segments are placed, circuits are migrated to make each segment EACPP as much as possible. The segments are then placed and static timing analysis tool generates critical paths. After segment placement, both wirability and timing are improved by moving individual circuits in the critical path. After each trial move affecting a critical path, acceptability of the trial move is performed on that path based on EACPP, timing, and wirability. The circuit has 11,500 cells and are partitioned into 60 segments. Twenty critical and almost zero-slack paths which become complete paths by circuit migration are selected. <Table 1> compares TDP-only approach with TDP-EACPP approach.

The complete design for a medium size chip which is guaranteed to meet all guidelines and performance constraints is a complex process involving placement, wiring, checking, and timing analysis. On average, 2 "minor" timing correction cycles were required. A minor "correction" involves analysis and manual wiring changes. On the other hand, without TDP and EACPP, we speculate that it would have been necessary to do 2 "major" corrections involving analysis, placement, and wiring. A major correction requires that placement and wiring be re-done.

<Table 1> Experimental results

	TDP-only*	TDP with EACPP*
Worst slack difference	4.8	0.9
Best slack difference	0.1	0.0
Average slack difference	1.9	0.2

*unit : % cycle time

6. Conclusion

In this paper, we address a potential timing problem in the hierarchical design methodology. As there is a high chance that some critical nets are not discovered during timing analysis on the hierarchical design and show up later after tape out. Therefore, we introduce the complete path concept that is a key to observability. Observability is defined as the ability to discover all the critical paths that the flat method would discover. Complete path also enables a design to be partitioned into independent sub_blocks prior to timing analysis. We also introduce and define paths which could be treated as complete paths and classified them accordingly. EACPP algorithm and experiments with it is also described. In future research, the tools or infrastructure to support the automatic generation of simple constraints are to be studied to achieve the global timing automation for EACPP designs. Also to achieve a useful and balanced EACPP, a research should be done to establish a method to plan hierarchy early in the design phase.

References

[1] Gary Coryer, "Semi_Custom VLSI Design Methodology," IBM Technical Memo, 1992.

[2] John F. Croix and D. F. Wong, "A Fast and

Accurate Technique to optimize Characterization Tables for Logic Synthesis," *Proceedings 1997 of Design Automation Conference*, pp.337-340.

[3] Sangyong Han and Wilm Donath, "Synergy and High Density Chip Design," *Proceedings on the 1993 International Conference on VLSI and CAD*, pp.233-235.

[4] Sangyong Han, "Timing Driven Placement," U.S. Patent 5,218,551 1993.

[5] Sangyong Han, "Automated Circuit Migration for Semi_conductior Chips," European Patent 91303351.0.

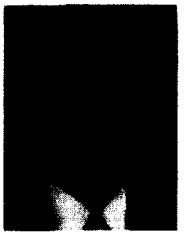
[6] Scott Hilker and Charlie Johnson, "Hierarchical Leverage in Chip Logical/Physical Design," Proceedings of 1992 EDA, pp.210-220.

[7] Soontae Kim, Dohyung Kim, and Sangyong Han, "DCRS : A Cell Library Characterization System," Journal of KISS(C), Vol.4, No.5, 1998, pp.755-761.

[8] C. Niessen, "Abstraction Requirements in Hiera rchical Design Methods," Design Methodologies edited by S.Goto, pp.151-181.

[9] S. Teig, R. C. Smith, and J. Seaton, "Timing Driven Layout of Cell Based IC's," VLSI Systems Design, May 1986, pp.63-73.

[10] Richard M. Warwick, "A Process for Timing Design and Analysis of Synchronous Interfaces," Proceedings of EDA ITL 1995, pp.110-117.



한 상 용

e-mail : hansy@cau.ac.kr

1975년 서울대학교 공과 대학 졸업 (학사)

1977년~1978년 KIST 연구원

1984년 미네소타 대학교 컴퓨터 공학과(공학박사)

1984년~1994년 IBM 연구소 연구원

1995년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : CAD, 가상프로토타이핑, 망 접속