I, Shaun K Peter, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:
A Performance Driven Placement System Using an Integrated Timing Analysis Engine

Student's name: Shaun K Peter

This work and its defense approved by:

Committee chair: Ranganadha Vernuri, Ph.D.
Committee member: Wen Ben Jone, Ph.D.
Committee member: Carla Purdy, Ph.D.
A Performance Driven Placement System
Using an Integrated Timing Analysis Engine

A Thesis Submitted to the
Graduate School of University of Cincinnati
in partial fulfillment of the requirements for the degree of

Master of Science

in the School of Electronic and Computing Systems
of the College of Engineering and Applied Sciences
July 2014

Shaun K. Peter

B.Tech in Electronics & Communication Engineering
National Institute of Technology Calicut, 2007

Committee Chair: Dr. Ranga Vemuri
Abstract

With the increase in design sizes the physical design of VLSI circuits continues to become more challenging. Achieving timing closure on such large designs with tight timing, area, power and design rule constraints is a difficult task. It is therefore important to come up with a placement which addresses these constraints to a good extent in order to facilitate timing closure.

Popular placement algorithms like analytic placement rely on total wire length as a metric to optimize placement. But total wire length is not an accurate measure of critical path delay and the resulting placement may not meet timing requirements. Timing driven placers try to address this issue by incorporating a more complex metric that takes into account the timing requirements. But most of these approaches are either restricted to detailed placement or the timing information supplied is not sufficient or accurate enough.

In this work we come up with a timing driven placement system that incorporates a relatively fast and accurate incremental timing engine with a placement engine. We show that starting with an unplaced netlist, using the information from the timer, we can produce placements which give better timing results post routing compared to wire length based placement. Although it is at a cost of some runtime, the better post routing results could give an overall better turnaround time as the number of iterations through the physical design flow could be reduced.
# Table of Contents

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

1.1 Physical Design Flow................................................................................................................1

1.2 Timing Closure..........................................................................................................................4

1.3 Placement Overview..................................................................................................................7

1.3.1 Traditional Placement Methods..........................................................................................8

1.3.2 New Approaches to Placement............................................................................................12

1.4 Timing Driven Placement.........................................................................................................13

1.5 Proposed Work.........................................................................................................................15

2 Static Timing Analysis Engine....................................................................................................17

2.1 The Timer Overview.................................................................................................................17

2.2 Core Timer Architecture and Implementation.............................................................................17

2.2.1 Level Assignment in Timing Graph......................................................................................18

2.2.2 Forward Propagation of Timing Data..................................................................................19

2.2.3 Slack Calculation and Path Tracing....................................................................................20

2.3 Timing Arc Delay Modeling......................................................................................................20

2.3.1 Wire Delay Calculation.......................................................................................................21

2.3.2 Cell Delay Calculation........................................................................................................23

2.4 Incremental Analysis................................................................................................................24

2.5 Verification of Timer Accuracy................................................................................................26

2.5.1 Comparisons with Industry Timer......................................................................................27

2.5.2 Error Trajectory...................................................................................................................29

3 The Timing Driven System..........................................................................................................31
3.1 Modeling of the Placement Problem .......................................................... 31
3.2 Base Placement Algorithm ........................................................................... 32
   3.2.1 Placement Metric .................................................................................. 32
   3.2.2 Legalization Method ............................................................................. 33
3.3 Review of Timing Driven Approaches .......................................................... 35
3.4 The Timing Driven Algorithm ....................................................................... 39
   3.4.1 Timer Interface for Placer .................................................................... 40
   3.4.2 Weight Assignment .............................................................................. 41
3.5 Comparisons ................................................................................................. 45
   3.5.1 Description of Benchmarks ................................................................. 45
   3.5.2 Placement Quality and Runtime Results .............................................. 46
4 Two-Stage Timing Driven Approach ................................................................ 49
   4.1 Overview of the Approach ....................................................................... 49
   4.2 Stage I of Placement ................................................................................ 49
   4.3 Stage II of Placement .............................................................................. 51
   4.4 Experimental Results ............................................................................. 53
      4.4.1 QOR Comparisons ......................................................................... 54
      4.4.2 Runtime Comparisons .................................................................... 56
5 Conclusion ....................................................................................................... 58
   5.1 Summary ................................................................................................. 58
   5.2 Future Directions ..................................................................................... 59

BIBLIOGRAPHY ................................................................................................. 61
List of Figures and Tables

Fig. 1.1 General physical design flow

Fig. 1.2 Timing checks

Fig. 1.3 Wire length estimation models

Fig. 2.1 Timing graph (DAG)

Fig. 2.2 Levels in timing graph

Fig. 2.3 RC network for star model of wire

Fig. 2.4 Example of a liberty delay lookup table

Fig. 2.5 Incremental analysis cone

Fig. 2.6 Sample report generated by timer

Fig. 2.7 Worst path slack (ns) comparison between our timer and PT

Fig. 2.8 Comparison of average slack (ns) of top 10 critical paths

Fig. 2.9 Error trajectory plot for design cordic with 10 placement scenarios

Fig. 2.10 Error trajectory plot for design stormcore with 10 placement scenarios

Fig. 3.1 Hypergraph to clique conversion

Fig. 3.2 Row legalization

Fig. 3.3 Flow chart of the TD system

Table 3.1 QOR comparisons of TD system with wire length based placer
Table 3.2 Runtime comparisons of TD system with wire length based placer

Fig. 4.1 Slack selection for weight computation

Fig. 4.2 Placement plot for the benchmark seq_align

Fig. 4.3 Critical path delay (ns) comparisons between custom timer and PT post-route for FD

Fig. 4.4 Critical path delay (ns) comparisons between custom timer and PT post-route for two-stage TD

Table 4.1 QOR comparisons

Table 4.2 Runtime comparisons
Acknowledgements

First and foremost I thank God for letting me reach a stage where I can write this acknowledgement.

I express my sincere gratitude to my thesis advisor Dr. Ranga Vemuri for guiding me throughout my work. I am grateful to Dr. Wen-Ben Jone and Dr. Carla Purdy for kindly agreeing to be on my thesis defense committee.

I am grateful to all my awesome DDELMates for the good times we had as part of the lab. All the benchmarks used in this work are generated using the framework developed by fellow DDELMites Nakul and Tuhin for their work. It was great to have such a good framework to use straight out of the box. I thank both of them for making it available for use by others in the lab and also for all the interesting discussions we had during our time together in the lab. I also thank Ujwal for helping with the generation of benchmarks. I would also like to thank Prabanjan for reviewing my thesis draft and providing valuable feedback.

I would like to make a special mention for all my roommates with whom I spent most of my time at Cincinnati. I also thank all my other colleagues at University of Cincinnati for making my time here memorable.

My parents have been my motivation throughout my studies and words cannot express how grateful I am for their support.
1. Introduction

1.1 Physical Design Flow

The design of a VLSI circuit involves various stages starting with a high level description of the functionality of the circuit and culminating in a detailed layout of the circuit that can be manufactured on a semiconductor wafer. The high level definition of a circuit is done using a hardware description language like Verilog or VHDL. These high level designs use generic logic blocks as building blocks. The functionality of the design is typically verified using simulations where different values are given for the inputs and output signals are checked for correctness. Once the circuit has been functionally verified, it has to be synthesized into a netlist which is the representation of the circuit in terms of logic gates or cells and the connectivity between them. Logic synthesis involves mapping of the logic in the high level description to standard cells in a library. These cells are pre-characterized for properties like delay, power and area. In addition to mapping logic to a given technology library it also does optimizations on the logic to improve timing, area and power. There are usually different cells to implement the same logic with a variety of area, power and delay characteristics. The availability of multiple such cells helps in meeting particular design constraints by trading off on some other quality. For instance, a larger cell would increase the area used by the design as well as the power consumed but would bring down the delay of the circuit because of the higher driving strength.
There are many other logic optimization techniques explored by the logic synthesis tool.

Fig. 1.1 General physical design flow
Once we have a synthesized netlist, then we go to physical design. Physical design mainly refers to the process of generating a layout of the circuit that can be physically manufactured on the wafer. This involves finding the best locations to place the cells of the design in a given area and determining the exact routes for the wires connecting the pins of the cells. This should be done in such a way that it meets certain design constraints such as performance or timing, power, area and manufacturability restrictions. The various steps that are part of a general physical design flow are captured in Fig 1.1.

Input to the placement stage is an unplaced netlist that has been synthesized using a standard cell library. In standard cell based designs, cells usually have the same height. The placement area in row-based placement consists of multiple rows of fixed height. This height corresponds to the height of cells in the standard cell library. The number of placement sites in a row could vary. The input to routing is a placed netlist where the best possible locations of all the cells have been determined and fixed. The routing step determines the best possible routes for the wires connecting the pins of the design. There are various constraints on routing such as the number of metal layers that can be used as well as other design rules related to manufacturability such as minimum spacing between wires. Once routing is done we can get the exact delay of nets by extracting the parasitic values for a net based on the geometry of its segments and assigned layers. The verification of timing performance is done by static timing analysis. The design rule checking (DRC) stage checks for violations related to manufacturability of the layout. Once the layout produced after routing
has been signed off by the timing analyzer and it does not violate any design rules, it can be manufactured in a fab.

1.2 Timing Closure

Most circuits are designed to meet certain operating frequencies. Timing closure is the ability of the placed and routed design to meet the performance requirements. Static timing analysis (STA) is used to verify the timing characteristics of the circuit. STA fundamentally tries to detect two types of violations in a design namely setup and hold violations as shown in Fig. 1.2.

Hold tests check if the data captured at sequential elements could be contaminated by the presence of very small combinational delay between launching and capturing points. This is required to ensure the functional correctness of the circuit irrespective of the operating frequency. Performance related issues are covered by setup tests which check if data reaches early enough to be captured by the clock at the capturing sequential element.

STA is preferred over exhaustive simulation for large circuits because of the runtime efficiency of this method. Exhaustive simulation has to look at an exponential number of input scenarios to ensure that the circuit works in all cases. But STA tries to capture the worst case timing characteristics of the circuit by finding an upper bound on the critical path delays in the circuit. In order to account for process variations involved in deep sub-micron technologies, usually STA adds more margins to its delay calculations to make the critical path delay predictions pessimistic. But this could be overkill and may make timing closure of
the design impossible. So there have been attempts at more accurate modeling of variations in STA such as by using statistical analysis where probability density functions of the delay values are used to calculate critical path delays instead of deterministic values with a lot of margin built into them.

Fig. 1.2 Timing checks

Timing analysis is usually done after each stage in the physical design process and even at the logic synthesis stage. Since the amount of information available
about the actual layout is less in the early stages, the estimation of critical path delays is also less accurate. During the placement stage we do not have any information about the actual routing of the wires in the design. So, various techniques are employed to estimate the delays for timing analysis at placement stage. After routing, we know the exact length of the wires in the design. Extraction of parasitic values can be performed once we have the post-routed layout. These extracted resistance and capacitance values can be used to calculate accurate delay values for the nets during timing analysis.

Since the first pass through the steps in physical design may not always result in timing closure, these steps are repeated over and over again until we have the desired results. Whether we repeat the whole flow or start from an intermediate stage such as post placement depends on the severity of the timing violations and the difficulty in fixing them. Going back to a much earlier stage gives more options to fix the issues. At synthesis stage we could utilize buffer insertion or cell resizing to reduce the delay in critical paths. At placement stage we could move the cells from a highly congested area to a lesser one to improve the routing quality and thereby reducing the delay of long wires through the congested area.

At routing stage we could rip up and reroute certain nets. These iterations are time consuming and so it is important that the models used to approximate the post layout physical effects at the early stages are accurate enough without costing too much runtime in order to make the overall turnaround time of the flow reasonable.
1.3 Placement Overview

Traditional placement algorithms use the minimization of total wire length in the design as their objective. By reducing the total wire length it is assumed that it will reduce the net delays as the delay of a net is proportional to the square of the net length. This should also reduce the cell delays as the wire load seen by a driving cell decreases.

![Wire length estimation models](image)

Fig. 1.3 Wire length estimation models

Since the actual route of a net is not known at the placement stage, different net models are used to estimate wire length. Some of the popular net models that have been described in existing literature are shown in Fig 1.3. In half perimeter wire length (HPWL) model, the semi-perimeter of the bounding box of the pins of the net is the estimate for net length. This is the most widely used estimate as it
is very fast to calculate. The star model has a connection from all pins to the centroid point. Clique is a complete graph with all the pins of the net as the vertices. Steiner model is computationally expensive but is an accurate estimate of the optimal route. Typically rectilinear Steiner tree is used in physical design as routes are only in horizontal and vertical directions. There are other models like single trunk Steiner which is an approximation of Steiner estimate that is faster to compute.

The more accurate and computation intensive ones may not be required at the placement stage if runtime is a big concern. It could also be less accurate if the design is highly congested and the number of routing layers available is less. In such cases, except for the nets that are initially routed, many nets could have less than optimal routing thereby making Steiner approximation optimistic.

### 1.3.1 Traditional Placement Methods

**Simulated annealing**

Simulated annealing is a probabilistic hill climbing algorithm which has many applications. In TimberWolf [1] implementation of simulated annealing for placement, the annealing process consists of two stages. In the first stage, cell movement is allowed between different rows and within the same row. Overlaps are allowed between cells. How far a cell can be moved is restricted to a region R. Initially R is large enough to include the whole placement area and as annealing progresses it shrinks. The area of R is proportional to the log T where T is the control parameter in simulated annealing called temperature. When R
becomes too small to include only one row the second stage begins. The overlaps are removed and annealing continues by allowing only interchanges between adjacent cells of the same row.

Three types of moves are allowed for cells in TimberWolf – displacing a cell to a new location (M1), interchanging two cells (M2) or changing the orientation of a cell (M3). These moves are selected with different probabilities. First a selection is made between M1 and M2 with certain probabilities p1 and p2 respectively. If a move M1 is a rejected for a cell, then move M3 is selected with probability p3. The primary cost function used is the semi-perimeter wire length minimization. There are additional penalty functions to minimize overlap between cells and to control row lengths. The annealing schedule is given by \( T_k = r(k) \times T_{k-1} \) for \( k = 1, 2, 3, \ldots \) where \( r(k) \) is a factor less than one. The value of \( r(k) \) increases slightly initially and then goes down as the procedure progresses. A total of \( K.N \) move attempts are made at each temperature \( T_k \) where \( N \) is the number of cells and \( K \) is a user defined constant.

There have been other simulated annealing based approaches like Dragon [2]. The simulated annealing based approach gives good results but is slow to converge. The quality of the results also depends on the cooling schedule for the parameter \( T \) and this could vary from design to design.

**Partition based methods**

Partition based placements rely on partitioning algorithms like FM or multilevel partitioning techniques like hMetis [3] to distribute the cells in the placement.
region. Academic placement tools like CAPO [4] and Feng-Shui [5] are good examples of such placers. The idea behind this approach is to bring together cells which are closely connected by repetitively dividing the circuit into sub-circuits such that the cut value across these partitions is minimized. The placement region is also partitioned accordingly. Bisection procedure is generally used for partitioning standard cell based designs although other technique like quadrature procedure is used for designs with high routing density at the center.

Terminal propagation is required to capture the correct cost of partitioning in partition based placement. The cost associated with two identical partitions could vary depending on the relative position of cells within the two partitions. Terminal propagation helps in pulling the cells connected to external cells into a region closer to the external connections within the same partition. There are other additional techniques used to improve the quality of the results such as in CAPO where different partitioning algorithms are used at different stages of recursive bisection. They use multi-level Fiduccia-Mattheyses (FM) algorithm when the number of cells for partitioning is large. FM algorithm is used when the size is below a certain number and finally and branch and bound is used when the number of cells is very small. Although partition based algorithms are fast and produce good results in terms of wire length and congestion, they may not be stable.

**Analytical methods**

Analytical placement approaches try to model the placement problem as a mathematical problem which can be directly solved to get a placement result.
The wire length minimization problem in placement can be modeled as a quadratic program since the sum of squared wire length is quadratic. It has been shown that quadratic program is convex and therefore polynomial time solvable. The advantage of formulating the placement problem this way is that there is only one solution and it is global optimum. On the other hand, the locations that are obtained as solution cannot be directly accepted as there could be numerous overlaps and legalization is required.

One of the most popular analytical placement algorithms is Gordian [7]. Gordian involves a global optimization step where the quadratic program that is formulated is solved and a legalization step where new overlap constraints are added to the current solution. This sequence is repeated until an optimum solution is obtained without any overlaps.

\[ \text{Cost of net connecting cell } i \text{ and cell } j, \quad C_{ij} = \frac{1}{2} w_{ij} \left[ (x_i - x_j)^2 + (y_i - y_j)^2 \right] \] (1.1)

\[ \text{Total cost, } \quad C = \frac{1}{2} x^T Q x + d_x^T x + \frac{1}{2} y^T Q y + d_y^T y + \text{const} \] (1.2)

The solution to the quadratic programming formulated will have many overlaps. Gordian uses a recursive partitioning and adding center of gravity constraints to distribute the cells evenly. Global optimization is done prior to every level of partitioning with the added center of gravity constraints. The quadratic program for global optimization is solved using conjugate-gradient method. Some partitions could be merged and repartitioned to reduce excessive overlap.

The analytical method discussed is mathematically well behaved and is efficient to find a globally optimum solution. But fixed ports are a requirement for this
technique to work else all the cells could be cluttered at the same location. They have also studied the benefit of using a linear objective function unlike the quadratic objective function discussed above. Quadratic objective function is found to be more desirable as it tends to make long nets shorter than linear objective function at some expense of short nets. Another analytical approach is force directed placement which is used as the base placement algorithm in our system and will be discussed later in chapter 3.

1.3.2 New Approaches to Placement

A hybrid net model which is based on a combination of clique and star models is used in FASTPLACE[8]. The net model depends on the fanout of the net and it enables faster solution to the quadratic program. The convex quadratic program is solved using incomplete Cholesky conjugate gradient method. The runtime for solving the quadratic problem is proportional to the number of non-zero entries in the Q matrix given in equation 1.2. Each of the non-zero entries corresponds to a 2-pin net. Modeling nets using a clique model would generate many 2-pin nets for high fanout nets and therefore the quadratic program becomes computationally more expensive to solve. Using the hybrid model where high degree nets are represented by star model helps to reduce the number of non-zero entries and thereby speeds up the solution. They also employ other techniques called cell shifting and iterative local refinement. Cell shifting is a technique used to spread the cells after the global optimization. Iterative local refinement tries to reduce the HPWL by iteratively going through all the cells one by one and computing a score for moving each cell in four directions. All the
moves with maximum positive scores are accepted. The distance that a cell can move reduces for successive iterations. Pseudo pins and nets are added to keep the cells that have been moved from collapsing back during the global optimization stage. This yields a very fast placement at the cost of a little wire length degradation.

All these approaches to placement could yield good results with respect to total wire length. But reduction in total wire length may not always correspond to the reduction in delay of timing critical paths. The ability to route the design is another consideration during placement. If the cells that need a lot of routing resources are concentrated in one region of the chip, it can lead to high routing congestion. We will now look at an approach to placement called timing driven placement which tries to address the timing requirements of the circuit and help in achieving post routing time closure.

1.4 Timing Driven Placement

As we go to smaller technology nodes, the gate delays scale down whereas interconnect delays remain almost the same. This results in wire delay playing a significant part in the path delays of a circuit in advanced technology nodes. Wire length based placement approaches may reduce the overall wire length of the circuit but may not necessarily be addressing the nets on the timing critical path. Timing driven placement tries to reduce the critical path delays by focusing on the length of timing critical nets. A cell is usually connected to multiple other cells in the circuit and therefore multiple paths can pass through a given cell. Trying to reduce the critical path delay by targeting a particular net may cause other paths
that pass through the connected cells to become critical. Therefore a balanced approach is required to perform timing aware placement.

There are various timing metrics that can be used by timing driven placers. The calculation of these metrics requires timing modeling and analysis. There has to be a tradeoff between the accuracy of the models required and the runtime. More accurate delay models better aid the placement engine to achieve timing closure but at the cost of runtime. Generally Elmore delay based net delay modeling is accurate enough at the placement stage. A basic timing quantity used for identifying the critical paths is slack. It is the difference in the actual arrival time of a signal at a point and the required arrival time at that point. The most timing critical path in a design can be obtained by tracing back from the endpoint with the worst negative slack (WNS). WNS can be used as a timing closure metric as it gives a sense of how close we are to achieve timing closure. Another metric is the total negative slack (TNS) in the design which is the sum of all the negative slacks in the design. While WNS helps in identifying the most critical path, it does not give an idea about the state of the entire design with respect to timing closure. Even though two placements can have the same WNS, the number of slack violations in the two cases can be quite different. Once may have only a few violations while the other may have many paths with negative slacks close to the critical path slack. The latter case is more difficult to fix and would require much more work than the former placement. TNS therefore gives a better sense of the state of the design. Usually WNS and TNS are used together as timing metrics to enable better timing closure.
Timing driven techniques, according to [10], are generally classified into net based approaches and path based approaches depending on how the timing metrics are utilized to guide the placement process. Net based approaches target individual nets on timing critical paths. The timing metrics could be used to help in weighting the nets during placement or placing some constraints on the maximum net length for those nets. Net weighting assigns higher weights to nets on critical paths which causes the placement engine to bring the connected cells closer and shorten the targeted nets when minimizing the weighted wire length is the objective. Net length constraints could be generated by distributing the total path delay among the different nets on the path in such a way as to obtain a zero slack solution for the entire path. Path based approaches try to optimize the path delays by formulating the problem into a mathematical program and solve for all paths or for the most critical paths simultaneously. This approach has better control over the results compared to net based approach but it could be very time consuming because of the large number of paths that must be solved simultaneously. There are also hybrid methods that use a combination of both these techniques to improve timing quality of the placement.

1.5 Proposed Work

Many of the existing timing driven approaches do not have a feedback mechanism to ensure that the timing quality improves throughout the placement process. Even if there is a feedback mechanism, most often it is not accurate enough to model the post layout timing quality of the design. Accurate incremental timing analysis based placement is typically used only during
detailed placement stage where the placer starts with an initial placement that has good timing quality and the range of placement changes is limited.

The objective of our work is to come up with a timing driven placement system that starts with an unplaced design, has an efficient timing feedback mechanism and models the timing information accurately enough to provide good timing results post routing. We utilize an integrated lightweight and accurate incremental timer to provide the feedback mechanism with which we can control and guide the weight assignment scheme more accurately. This can take advantage of the speed of net based approach as well the accurate timing view of path based approach without much impact to runtime. In chapter 2 we will discuss the timer that we have developed to be the backbone of the placement system. In chapter 3 we will start with a discussion on the base placement algorithm that we use. Then we will review the various existing timing driven algorithms followed by a description of our timing driven approaches and their results.
2. Static Timing Analysis Engine

2.1 The Timer Overview

The backbone of our timing driven placement system is a reasonably fast and accurate static timing analysis engine (STA). We have built a lightweight STA engine from scratch which is integrated into the placement system by creating an interface which the placer can utilize efficiently. This is what provides the timing related feedback to the placement engine.

The verification of circuits for timing requirements basically involves checking for setup and hold violations as shown earlier in Fig. 1.2. Our timer deals with only setup tests as we are interested in optimizing the performance aspect of the circuit during placement. The timing analysis tool takes two inputs – 1) the synthesized netlist of the circuit to be analyzed in bookshelf format and 2) the standard cell library used to synthesize the circuit in liberty format. The information about the setup tests to be performed is obtained from the input standard cell library. Now we will describe how the circuit is modeled in the timer and how the relevant data is propagated in order to perform the timing tests.

2.2 Core Timer Architecture and Implementation

The timer models the entire circuit netlist as a directed acyclic graph (DAG) with pins and ports as vertices (timing points) and timing arcs as edges. A timing arc could either be the internal connection from input to output of a cell or a wire connecting the pins of two cells. Each of these arcs has a delay associated with it. The timing data is propagated starting from the input ports through the graph
to the timing endpoints in the graph where timing tests are performed. Endpoints can either be output ports or data pins of sequential elements.

Fig. 2.1 Timing graph (DAG)

2.2.1 Level Assignment in Timing Graph

In order to perform forward and backward propagation of timing data in an efficient manner, the graph nodes have to be topologically ordered. Therefore each timing point in the graph is assigned a level number based on its dependencies. In arrival time calculation, for example, timing point with higher level number depends on one with lower level number. Fig. 2.2 shows the timing graph after level assignment for the circuit in Fig. 2.1. Level assignment is done using topological sorting algorithm. Since topological sorting will work only for DAGs we must ensure that the graph we have is one by checking for cycles. We use depth first search (DFS) algorithm to detect loops in the graph. Currently, cycles are not supported as we do not expect circuits to have loops unless
transparent latches are used. This behavior can be extended to break the cycles detected to make the graph a DAG.

![Timing Graph Diagram](image)

**Fig. 2.2 Levels in timing graph**

### 2.2.2 Forward Propagation of Timing Data

Arrival times and delays are calculated during forward propagation. A priority queue is used for this purpose. First, all the timing points are pushed into the priority queue. Then they are popped and processed one by one. The priority for retrieval is based on the level number of the timing point. Lower the level number, higher the priority. So, the timing points are processed in the increasing order of level number which was assigned in the previous step. This ensures that arrival times at all source nodes are up to date before sink node is processed.

Cell delay and output slope calculation is lookup table based whereas net delay calculation is Elmore delay based. These will be explained in detail in subsequent sections.
2.2.3 Slack Calculation and Path Tracing

Slack at a timing point is calculated as the difference between the required arrival time at the point and the forward propagated arrival times. The required arrival time at the data input pin of a register is the ideal clock period less the setup time specified in the cell library. Required arrival times are propagated backwards starting from the timing endpoints. Similar to forward propagation, we use a priority queue for backward propagation. But unlike forward propagation, the priority increases as level number increases. To speed up required arrival time calculation, delays computed during forward propagation are cached and reused. At endpoints, tests specified in the library or user defined tests are performed to compute slacks. Slacks are also automatically propagated backwards. A min heap is created using the endpoints based on their slack values. This ensures that we can retrieve endpoints in the increasing order of slack. The worst path is traced back from each endpoint and stored in a path data structure which can be used for reporting or transferring the worst paths information to the placement engine.

2.3 Timing Arc Delay Modeling

There are two types of delay arcs, namely net arcs and cell arcs. Delay calculation is different in each case. We look up cell delays from the standard cell library that the timer takes as input and the accuracy of cell delays depends on the values characterized in the library. Since the nets in a design are not routed at the placement stage, the net delays have to be modeled without the actual routing information. The accuracy of net delay modeling determines how well the
post-route timing results match with the results predicted by our timer. This in turn determines how well our timer can guide the placement engine.

2.3.1 Wire Delay Calculation

The timer models wires in the circuit using a star model as shown in Fig. 2.2. In a star model, in addition to the nodes that are the pins of the net, there is an extra node to which all the other nodes connect. The coordinates of this node is the centroid of the coordinates of all the other nodes. Each segment of the wire that connects to the centroid is modeled as an RC segment. The values of resistance and capacitance per unit length used are derived from the Milkyway reference design database that was used to generate our benchmarks. The values used for our analysis purposes are $R = 2e-9 \text{ fF/nm}$ and $C = 1.5e-4 \text{ kΩ/nm}$. Pin capacitances are obtained from the values in the standard cell library.

Fig. 2.3 RC network for star model of wire
The delay of the resultant RC network is estimated using Elmore delay. Elmore delay method is a fast first order approximation of the actual delay of the network and should be accurate enough for optimization purposes. The 50% delay value at any sink pin is given by

\[ D = \ln(2) \times T_{RC} \]  \hspace{1cm} (2.1)

where \( T_{RC} \) is the Elmore delay from the source pin to the sink pin.

The slew values are calculated based on the 20% to 80% transition time of signals. When the signal at the output of a gate passes through a wire to reach the input pin of a gate, there is degradation in the slew value. The actual value of transition time at the input of a gate \( S_i \) is calculated as in [12].

\[ S_i = \sqrt{(S_o^2 + S^2)} \]  \hspace{1cm} (2.2)

where \( S \) is the step response for the RC network for 20% to 80% transition time and \( S_o \) is the slew of the gate output in the previous stage. \( S \) is given by

\[ S = \ln(4) \times T_{RC} \]  \hspace{1cm} (2.3)

Since most timing critical nets are typically routed first, we give some credit to nets that could be critical when calculating their delay since our star model could be overly pessimistic. We choose high fanout as criteria to identify such nets. The nets with high fanout (>10) therefore would have a net delay lesser than what the star model would calculate.
2.3.2 Cell Delay Calculation

Gate delays and slews are calculated using already characterized values available as lookup tables in the standard cell library used for synthesizing our benchmarks. There are lookup tables for all the relevant arcs within the cell and for each of rise and fall transitions. Gate delay depends on the input slew and output load. The delay and slew lookup tables are two dimensional tables with input slew and output load as the two indices.

Fig. 2.4 Example of a liberty delay lookup table

Delay and output slew values are only characterized for a finite set of values of input slew and output load. The actual value for any given values of input slew
and output load are obtained using bilinear interpolation or extrapolation. We consider the slew of the signal with the worst arrival time for slew propagation.

A lumped capacitance is used to model the load seen by driving gate. The output load at a cell output pin is obtained by adding up all the net capacitances and sink pin capacitances. This is a pessimistic approach to output load calculation as the effective capacitance seen by the driver is usually lesser than the sum of all load capacitances due to resistance shielding effect.

### 2.4 Incremental Analysis

An essential attribute of a timer to aid timing optimization is that it has to be able to perform fast incremental updates to the timing graph for any change in the design. For our placement system the changes are limited to changes in the location of cells. Changes in cell locations directly affect net delays, net slews and output loads of cells which in turn affect cell delays and cell output slews. So, for any incremental change in placement, new arrival times and slews have to be propagated for the region of the design that is affected. Limiting the part of the timing graph that needs update is the key to fast incremental analysis. We deploy mainly two techniques to reduce the work involved in incremental timing analysis.

We queue up only those timing points in the output cone of the incremental change for updating arrival time and slew. Since changes are limited to the changes in cell location in our case, identifying the source pins of the nets feeding into the cell that has moved its position is sufficient to perform an
incremental update of the timing graph. The algorithm for incremental analysis starts with these points and queues up all downstream points affected by change in arrival time or slew at these points as the forward propagation progresses.

Dominance limiting [13] is another technique used to further reduce the number of points that are queued up for incremental processing. If there is a timing point outside the forward cone of change whose arrival time dominates that of a timing point inside the cone, then change in arrival time at the timing point inside the cone is not going to affect downstream timing values. So, forward propagation can stop at such points thereby cutting down the size of the region in the timing graph that would need timing updates. Fig. 2.5 illustrates the downstream region in a timing graph that is affected by an incremental change after taking dominance limiting into account.

Fig. 2.5 Incremental analysis cone
Simplified pseudo code for incrementally updating arrival time (AT) of node N:

```
WHILE level(lowest entry in AT queue) <= level(N)
    Remove first item X from among the lowest entries in AT queue
    Re-compute AT for X
    IF AT of X has changed
        Queue all successors of X
    END IF
END WHILE
Return AT(N)
```

2.5 Verification of Timer Accuracy

The timer can report worst slacks and the corresponding paths for analysis in addition to transferring this information via an interface to the placement engine. A sample timing report generated by our timer is shown in Fig. 2.6. Paths are sorted in the increasing order of slack for reporting. In addition to the endpoint slack, it reports the transition times, stage delays and arrival times at all nodes along the path. This information is used to compare the results of our timer with the results reported by an industry timer before and after routing on our benchmarks. This is required to ensure that the timing results of the timer are accurate enough to guide the placement engine such that it gives good post-route timing results. We also plot the error trajectory of our timer with respect to the post routing timing results reported by the industry timer at various stages of our placement procedure. This is to ensure that these values are well correlated throughout the course of the placement procedure.
Fig. 2.6 Sample report generated by timer

2.5.1 Comparisons with Industry Timer

To validate the accuracy of our timer, we compare the timing results reported by our timer with that of the industry timer PrimTime (PT). We took 4 placed benchmarks of various sizes and compared the WNS values reported by our timer and PT. We also performed routing on the design using an industry place and route tool IC Compiler (ICC) and again compared the WNS reported by PT post routing with the value predicted by our timer.
Our objective was to as accurately predict the post routing timing results as possible without much runtime hit. We can see from Fig. 2.7 that WNS values predicted by our timer closely follow post-route results reported by PT. PT uses wire load model for net delay computation during pre-route timing. This could be the reason why pre-route WNS reported by PT is much different from the post-route WNS. The large difference could also be possibly contributed by the high routing congestion in these benchmarks. We also plotted a similar comparison using the average slack of the ten most critical paths in these designs.
2.5.2 Error Trajectory

To further ensure that this close matching with post-route timing occurs throughout a placement procedure, we performed an iterative placement procedure on a couple of designs. We write out placements at an interval of 5 iterations during the procedure until we have sufficient number of placements that capture the trajectory of the placement procedure. These placements are routed and then we plot the error values between the post-route PT results and timing results predicted by our timer.
Fig. 2.9 Error trajectory plot for design cordic with 10 placement scenarios

We see that error values fluctuate around 0 and are within 0.5 ns. Similar plot for a bigger design is given in Fig. 2.10.

Fig. 2.10 Error trajectory plot for design stormcore with 10 placement scenarios
3. The Timing Driven System

Our aim is to come up with a placement engine that efficiently utilizes the timing information from the timer we have built to guide the placement process. We will first introduce a force directed placement algorithm that serves as the base upon which the timing driven system is built. The modeling of placement objects and placement area as well as the core force directed algorithm are identical for the base and timing driven versions. All comparisons will be with respect to this base implementation.

3.1 Modeling of the Placement Problem

The netlist is modeled as a hypergraph where the vertices represent the cells and hyperedges represent the nets. We use a clique model as shown in figure to convert a hyperedge to an equivalent set of binary edges.

Fig. 3.1 Hypergraph to clique conversion
The weight of each edge of the complete graph that represents the hyperedge in a clique model is $1/K$ where $K$ is the number of vertices in the hyperedge. This connectivity information is internally stored as an adjacency matrix. These edge weights are fixed weights but we also maintain a separate set of variable weights for the edges and that will be later used by the timing driven placement algorithm.

The placement area is of fixed size and is row-based. Each row has a fixed capacity and the total cell area in a row cannot exceed this capacity. Before the actual placement algorithm begins, the cells are randomly placed along the placement rows without exceeding the row capacities. In our placement engine we initially allow for horizontal overlaps between cells but overlaps between cells of different rows are not allowed.

### 3.2 Base Placement Algorithm

The base placement algorithm used in our tool is force directed (FD) placement which is an iterative optimization approach where we try to find and assign to each cell its zero-force location. The force between two cells is modeled using Hooke’s law and is proportional to the distance between the cells. Various techniques are used to resolve conflict between multiple cells. Since this approach computes the local optimum, we need to have multiple iterations to converge to a good placement globally.

#### 3.2.1 Placement Metric

We use half perimeter wire length (HPWL) as the metric to decide if a placement is better than the other. The ultimate goal is to minimize HPWL. We accept
degradation in HPWL to a certain extent with the aim of getting a placement that is close to the global optimum in further iterations. The optimum location \((x_i, y_i)\) for a cell \(i\) is computed by using the weighted distance from all other connected cells as below

\[
\sum_j (x_i - x_j) w_{ij} = 0 \quad \text{and} \quad \sum_j (y_i - y_j) w_{ij} = 0
\]  

(3.1)

where \((x_j, y_j)\) are the coordinates of all cells \(j\) connected to cell \(i\) and \(w_{ij}\) are the weights.

3.2.2 Legalization Method

![Fig. 3.2 Row legalization](image)

Since we allow horizontal overlaps between cells of the same row, we need to legalize the placement for it to be complete. We have observed that performing legalization at the end of each iteration of the FD algorithm gives a much improved result than performing a single legalization step at the end. To perform horizontal legalization we need to keep track of the relative position of the cells in a row. This can be achieved by ordering the cells by the x-coordinate of their left
edge. Then in order to perform legalization we arrange the cells in the order of their relative starting positions without any overlap starting from the leftmost site in the row. This approach is not the best if we do not want to alter the placement too much during legalization. We therefore have a mechanism wherein we keep track of the leftmost cell in each row that has a location change during an iteration of the placement algorithm. Horizontal adjustment of the placement starts only from this cell in each row. Only if this strategy causes the cells to overflow the row, we resort to full row legalization. This approach is especially useful for our timing driven placer as it will help to effectively utilize the incremental nature of the timer.

A simplified pseudo code for the placement algorithm:

```plaintext
performFDPlacement
    iter_count = 0, best_iter_count = 0, best_hpwl = MAX_FLT
    WHILE (iter_count < ITER_LIMIT)
        FOR each cell C in cell list L sorted by weighted connectivity
            Compute optimum location X of C
            Clear current placement of C
            Add cell size to current row capacity
            IF location X is free or occupied but not locked
                IF row capacity will not be exceeded
                    Place cell at X and lock site
                    Decrease row capacity by cell size
                ELSE IF row capacity will be exceeded
                    Find and place in closest row with sufficient capacity
                END IF
            IF a cell has been displaced
                Start a ripple until there is no displacement
        END FOR
    END WHILE
```
3.3 Review of Timing Driven Approaches

The net weighting scheme that we have discussed in chapter 1 has various variants that have been tried in previous works. They are mainly divided into static and dynamic net weighting approaches. In static net weighting, net weights are computed once before the placement starts and then remain unchanged.
throughout the procedure. The methods described in [14] and [15] assign net weights based on the slack of the net’s path. Nets with more critical slacks are assigned a higher weight. One of them uses a continuous slack based weight while the other uses a discretized slack based weight assignment. Path sharing is considered in another work where nets shared among multiple critical paths are assigned more weightage. Another approach that has been tried is sensitivity based net weighting. In [16] and [17], the authors try to estimate the impact of net weighting on timing using weight sensitivity to timing. This sensitivity is used to guide the net weighting. This could give slightly better predictability to the static net weighting approach. Irrespective of the type of net weight generation, the inherent disadvantage with static net weighting is that critical paths in the design do not remain the same as the placement changes. Weight assignment based on the timing analysis results on the initial placement will improve the original critical paths but could generate new critical paths.

Dynamic net weighting schemes were proposed to address the issue of changing critical paths associated with static net weighting. The net weights are adjusted at the end of each placement iteration using the timing information at that stage. Dynamically changing the weights without any control criteria can lead to oscillations of a net between critical and non-critical states. This could prevent convergence of the placement algorithm. Some works have proposed the use of criticality history of the net to decide the weight. In order to dynamically update weights, the timing has to be recomputed periodically. The work [15] computes incremental changes in slack due to change in wire length using delay
sensitivities to wire length. A more accurate timing analysis such as in [18] can also be used to periodically update timing.

Another approach to net based timing driven placement is using net constraints. Since net length is the major factor that determines net delay and output load of cells, putting constraints on the maximum length of nets can help in guiding the placement towards achieving timing goals. There are one-time as well as incremental approaches to net constraint generation. The former tries to first develop timing constraints for individual nets such that all the paths meet slack criteria for the design and these are then used to generate constraints on the net lengths. The zero-slack algorithm proposed in [19] computes delay bounds for each net in such a way that all paths have zero slack. Generation of net constraints is based on these delay bounds and is formulated as a linear programming (LP) problem which tries to maximize the permissible length for each net subject to the LP constraints that timing requirements are met. The incremental approach starts out with some net length constraints and keeps tightening these constraints with every iteration until the slacks are met.

Path based timing driven algorithms try to simultaneously optimize delay for all the paths or a set of most critical paths in the design. Enumerating all the paths in the design for this purpose is not a viable solution as it is would be prohibitively time consuming for large designs. Selecting a set of critical paths does not guarantee that other paths will not become critical due to placement changes. In order to avoid enumerating all the paths, a linear programming (LP) framework based solution has been proposed. This relies on the use of timing graph and
introduction of auxiliary variables to develop physical and timing constraints. Linear models are used for cell and net delays and the objective function could be to maximize slack. The constraints could be net bounding box constraints, delay constraints or slack constraints. Since LP based solution can result in many overlaps, it is used in conjunction with partition based placer in [20]. Additional center of gravity based constraints have to be added after each partition. The LP based formulation can have a large number of constraints and it could not be solved efficiently for large circuits. A method called Lagrange Relaxation has been used to resolve this issue in [21]. It is an iterative technique to transform constrained LP-formulation into a set of unconstrained problems which can be solved efficiently.

Differential timing analysis is another technique that uses a timing graph. It tries to optimize placement by looking at the changes in delays, arrival and required times at all the pins of the circuit with respect to a reference timing result. This approach is only accurate for only small scale changes and therefore is more suitable for detailed placement. The Hippocrates [22] placement algorithm applies differential timing analysis to detailed placement. The differential timing analysis uses linear cell delay model and Elmore delay model for wires. It also uses timing constraints on pins to ensure that placement changes do not increase the arrival times at individual pins by more than an allowable amount.

With the increased complexity of circuits, net based approaches have an advantage due to their ease of computation. But they offer less control in terms of convergence. They also ignore impact of slew propagation. Effective net
weighting algorithms have to consider path based analysis in order to give better timing results since timing is inherently path based. Some hybrid timing driven placement methods have also been proposed in available literature. One of these combines net weighting with LP-based problem formulation. They use path based delay sensitivity as a basis for net weighting and also perform slew propagation for a limited number of stages.

Our approach to timing driven placement is a hybrid one which is based on dynamic weighting but it works on a set of critical paths identified by performing timing analysis using a timing graph. The problem of changing critical paths is addressed by incrementally updating the timing graph to obtain the most up to date critical path information before each iteration of weighted net placement. The way we model nets for delay calculation is similar to how it is done in SPEED[18]. The net weights are assigned based on the critical path slacks. We try to get a high quality of placement in terms of the ability to close timing post routing by performing as accurate timing analysis as possible without sacrificing much runtime.

3.4 The Timing Driven Algorithm

Timing driven algorithms utilize some metric that models the timing requirements. In the proposed timing driven placement system we utilize the information obtained as feedback from the integrated timing engine. There is a variety of information that can be passed to the placement engine since we have a basic timer. These include the most critical paths or endpoints, the worst negative slack (WNS) or the total negative slack (TNS)
The two placement quality metrics WNS and TNS are defined as

\[
WNS = \min \{ s : s < 0 \} \quad \forall s \in S \\
TNS = \sum \{ s : s < 0 \} \quad \forall s \in S
\]

(3.2) (3.3)

where S is the set of all endpoint slacks.

The timing closure criterion is typically to make WNS = 0 and that is what our placement system tries to achieve.

Fig. 3.3 Flow chart of the TD system

3.4.1 Timer Interface for Placer

The interface between the timer and the placer is a critical component of the system. Only with an efficient interfacing mechanism can the system work in the
intended way. The timer provides application programming interfaces (APIs) to query the worst slack, total negative slack and also to get a collection of critical paths in the design. There is also an API to update timing incrementally.

During the placement iterations, the positions of the cells keep changing and hence the timing at the different points in the timing graph. This information has to be conveyed by the placer to the timing engine in order to get the up to date timing result at each stage. Therefore there is also an API to invalidate and queue up select points in the timing graph for timing update. Since the placer knows when the position of a cell is changed, it can mark the cell as a point of origin for timing update. The placer calls the API for every such cell and the API internally invalidates the timing on all pins of the cell and pins on nets connected to these cells. The API also queues up the source pin of the nets feeding into the cell in the forward propagation queue. The incremental timing update algorithm will queue up only timing points in the forward cone of the pins in the forward propagation queue after applying dominance limiting to avoid unnecessary forward propagation calculations.

3.4.2 Weight Assignment

Each net in the design is replaced with an equivalent set of 2 pin edges using a clique model as mentioned earlier. A placement graph is thus generated with cells as vertices and the generated edges connecting these vertices. The weight of each of the individual edges is given by 1/K where K is the number of pins of the net. This is a fixed weight ($W_f$) for the entire placement procedure. The effective weight of an edge ($W_e$) in the connectivity graph is the product of this
fixed weight and a variable weight \((W_v)\) which captures the changing criticality of the path to which the edge belongs. The effective weight of the edge connecting cells \(i\) and \(j\) for the \(n^{th}\) iteration is given by
\[
W_{e_{ij}}(n) = W_{f_{ij}} * W_{v_{ij}}(n)
\]
(3.4)

After each iteration of placement, the \(N (=1000)\) most critical paths are identified by the timer. For each of the edges in a selected critical path, we use the path slack and a slack target value \((S_T)\) to calculate the new variable weight. The slack target is a tunable value and is set to a positive value close to 0 for best results. The variable weight for the \(n^{th}\) iteration is calculated as
\[
W'_{v_{ij}}(n) = W_{v_{ij}}(n - 1) + \Delta W_{ij}
\]
(3.5)

where \(\Delta W\) is the incremental weight change in the variable weight from the previous iteration and \(W_{v}(0) = 1\).

\[
\Delta W_{ij} = \begin{cases} 
S_T - S_p & \text{if } S_T > S_p \\
0 & \text{otherwise}
\end{cases}
\]
(3.6)

where \(S_p\) is the slack of the most critical path containing the edge connecting cell \(i\) and cell \(j\).

The incremental weight added is large if the path slack is much lesser than the target slack. The variable weight value is capped to a maximum value of \(W_{\text{max}}\). This prevents excessive shortening of nets in the critical path in a single iteration. We use a value of 10 for \(W_{\text{max}}\).
The advantage of using such a method to calculate the incremental weight is that initially we start with large weights on critical paths which bring down the critical path delays quickly. As the path delay nears the target slack, the increase in weight goes down resulting in smaller placement changes. Although initially the changing critical paths could cause oscillation, this helps to stabilize the placement to a good solution towards the latter stages of the placement iterations. For a design with a large number of critical and near critical paths and a WNS that is much less than 0, many edge weights could approach $W_{\text{max}}$ in the initial iterations. This could prevent the effectiveness of the timing driven weight assignment. So we have to keep track of the number of edges whose weights reach the maximum value and if this exceeds a certain threshold, we have to reset the weights and start over again. In addition to weights on the edges, the order in which the cells are processed during FD placement iteration also matters. So before an iteration of placement starts, we sort the cells based on the total weight of all the edges associated with a cell. The cell with the largest total weight is processed first.

A simplified pseudo code for the timing driven approach:

```
performTDPlacement
    iter_count = 0, best_iter_count = 0, best_wns = MIN_FLT
    WHILE (iter_count < ITER_LIMIT)
        FOR each cell C in cell list L sorted by weighted connectivity
            Compute optimum location $X$ of C
            Clear current placement of C
            Add cell size to current row capacity
            IF location $X$ is free or occupied but not locked
```
IF row capacity will not be exceeded
   Place cell at X and lock site
   *Queue up changes for timing update
   Decrease row capacity by cell size
ELSE IF row capacity will be exceeded
   Find and place in closest row with sufficient capacity
   *Queue up changes for timing update
END IF
IF a cell has been displaced
   Start a ripple until there is no displacement
END IF
ELSE IF location is locked
   Place C at nearest unlocked location from X
   *Queue up changes for timing update
IF (number of locked locations > ABORT_LIMIT)
   Clear all locks
   *Legalize horizontal overlaps and queue up changes for timing update
   *Update timing
   *IF (WNS > best_wns)
     *best_wns = WNS
     Save current design as best_design
     Save current iteration count as best_iter_count
     BREAK from FOR
   ELSE IF (iter_count – best_iter_count > TIME_OUT_COUNT)
     BREAK from WHILE
   END IF
END IF
END IF
END FOR
*Update weights based on new critical paths
Increment iter_count and CONTINUE
END WHILE
3.5 Comparisons

3.5.1 Description of Benchmarks

Our placement benchmarks have been developed using the designs available at opencores.org. The VHDL/Verilog files are synthesized using saed32nm standard cell library in Design Compiler to generate an unplaced netlist. This is then converted to the academic placement format called Bookshelf. The LEF file is used to obtain the cell dimensions. The remaining physical information required for generating the placement benchmark in Bookshelf format is derived from the DEF file of design.

The Bookshelf placement benchmark input consists of four files namely .nodes, .nets, .pl and .scl files. The .nodes file has the cell and cell size information, .nets has the connectivity information, .pl has information for the placement coordinates of all cells and .scl has the placement area information such as the number of rows. The input .pl file will have the origin as the coordinates for all the cells except the fixed ports since it is an unplaced netlist. The output of the placement tool is a .pl file with the final location coordinates of the cells in the design.

Timing information required to calculate cell delays are obtained from the liberty file for the standard cell library. In order to perform routing and compare post-
route results, the output .pl file is converted to a DEF file which can be fed to IC Compiler to perform routing. These benchmarks are not optimized for delay during synthesis and therefore tend to have large path delays.

### 3.5.2 Placement Quality and Runtime Results

In order to see the effectiveness of the timing driven system over the traditional wire length based placement we compared the wire length based force directed placement algorithm and the modified force directed placement algorithm that uses the integrated incremental timer.

<table>
<thead>
<tr>
<th>Benchmark (# of cells)</th>
<th>Worst Path Delay (ns)</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FD (HPWL)</td>
<td>TD</td>
</tr>
<tr>
<td>openfpu64 (14347)</td>
<td>2.4000</td>
<td>2.4025</td>
</tr>
<tr>
<td>cordic (20577)</td>
<td>1.4238</td>
<td>1.3699</td>
</tr>
<tr>
<td>reedsoldec (31179)</td>
<td>2.1619</td>
<td>1.7542</td>
</tr>
<tr>
<td>seq_align (44098)</td>
<td>4.2020</td>
<td>3.5869</td>
</tr>
<tr>
<td>stormcore (87662)</td>
<td>5.0112</td>
<td>4.1455</td>
</tr>
<tr>
<td>pairing (288622)</td>
<td>6.3827</td>
<td>4.7324</td>
</tr>
<tr>
<td>jpegenc (344646)</td>
<td>7.9953</td>
<td>6.4769</td>
</tr>
<tr>
<td>AES (428665)</td>
<td>7.9976</td>
<td>5.9885</td>
</tr>
</tbody>
</table>

Table 3.1 QOR comparisons of TD system with wire length based placer
The comparison of worst path delays show that on an average there is a 20% improvement in the worst path delay for larger circuits. This shows that at least for large designs our timing driven system can give much improvement in path delays. The larger benchmarks we used had high area utilization. This could have resulted in relatively high routing congestion. Using a star model for wire delay calculation enables us to be a bit pessimistic compared to more accurate but time consuming Steiner model. This helps in matching the post-route timing better during placement for congested designs.

In terms of cost of computation, our timer implements more complex algorithms than simple HPWL calculation. We also do some post processing of this timing data to generate dynamic weights for our placement graph. This will slow down the whole placement procedure. But interestingly, the delay improvements that we have shown in Table 3.1 does not cost too much runtime hit.

<table>
<thead>
<tr>
<th>Benchmark (# of cells)</th>
<th>Runtime (s)</th>
<th>Ratio of runtimes (TD/FD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FD (HPWL)</td>
<td>TD</td>
</tr>
<tr>
<td>openfpu64 (14347)</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>cordic (20577)</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>reedsoldec (31179)</td>
<td>45</td>
<td>34</td>
</tr>
<tr>
<td>seq_align (44098)</td>
<td>47</td>
<td>86</td>
</tr>
<tr>
<td>stormcore (87662)</td>
<td>192</td>
<td>279</td>
</tr>
</tbody>
</table>
Table 3.2 Runtime comparisons of TD system with wire length based placer

<table>
<thead>
<tr>
<th></th>
<th>Iterations</th>
<th>Wire Length</th>
<th>WNS Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>pairing (288622)</td>
<td>1120</td>
<td>1505</td>
<td>1.34</td>
</tr>
<tr>
<td>jpegenc (344646)</td>
<td>2027</td>
<td>2452</td>
<td>1.21</td>
</tr>
<tr>
<td>AES (428665)</td>
<td>1938</td>
<td>2803</td>
<td>1.45</td>
</tr>
</tbody>
</table>

The reason why degradation in runtime with the timing driven algorithm is not as huge as expected in many designs is that the wire length based algorithm performs much larger number of iterations compared to the timing driven algorithm to reach its final placement. We use a high TIME_OUT_COUNT value of 20 which is used to terminate the placement procedure if there is no improvement in wire length or WNS. If there is no improvement in wire length or WNS for 20 continuous iterations we terminate the algorithm and accept the best placement obtained so far as the final placement. The larger designs which show around 30 to 40 percent degradation in runtime reach their best solution in much smaller number of iterations with timing driven approach. For designs with more than 2x degradation in runtime, the number of iterations is comparable. Typically if both the algorithms run for equal number of iterations we will see around 3x degradation in runtime.
4. Two-Stage Timing Driven Placement

4.1 Overview of the Approach

In the timing driven placement method described in the previous chapter, we perform a simple dynamic weight assignment based on the critical paths obtained from the integrated timing analysis engine. Since we start with an unplaced netlist as input to our placement system, there could be a large number of critical or near critical paths after the initial random placement. This is especially true for very large designs. In such cases the rate of convergence of the worst slack is seriously limited by the number of critical paths that is considered during dynamic weight assignment. Enumerating all the critical and near critical paths is expensive in terms of runtime and therefore is not a viable option where a fast timing analysis is required as is the case in our placement system. We therefore need a better approach that can shake up the whole design during the initial stage to get better results instead using a limited number of critical paths. During the later stages we need to have a more targeted approach considering the critical paths. This requires a path based approach which can give better control than a simple weight assignment technique.

Now we will discuss the two distinct approaches that form the two stages of our timing driven placement system.

4.2 Stage I of Placement

The integrated timing analysis engine propagates the worst slack from the endpoints backwards through the timing graph. Therefore, at every pin, slack
information for the worst path going through the pin is available. This worst slack information at every pin can be used a way to avoid the need to do the enumeration of all the paths in the design.

For every net in the design, we use the slacks at the sink pins for calculating the variable weights for the corresponding edges in the placement graph. If there are multiple connections between two cells, we use the worst among all the sacks. The equation 3.5 is again used to obtain the incremental weights.

Unlike the procedure in chapter 3, we do not cap the maximum value of variable weight when updating placement graph edge weights. Another difference is that the slack target used for calculating the weights is not fixed for all designs. It is set for each design depending on the initial state of the design. Once the initial random placement is done, the WNS is calculated using the timer. This value gives an idea of the initial timing state of the design. We use a value of 0 for slack target if this WNS value is lesser than -5 and a value of 5 if it is greater than -5. There are two conditions for moving to stage 2 of the placement procedure.

Fig. 4.1 Slack selection for weight computation

Unlike the procedure in chapter 3, we do not cap the maximum value of variable weight when updating placement graph edge weights. Another difference is that the slack target used for calculating the weights is not fixed for all designs. It is set for each design depending on the initial state of the design. Once the initial random placement is done, the WNS is calculated using the timer. This value gives an idea of the initial timing state of the design. We use a value of 0 for slack target if this WNS value is lesser than -5 and a value of 5 if it is greater than -5. There are two conditions for moving to stage 2 of the placement procedure.
The first is when there is no improvement in WNS for a certain number of iterations. We also jump to stage 2 if the value (TNS/# of endpoints) is less than WNS/2. This is because when this condition is met, it indicates that average negative slack per endpoint is much less than the WNS and we could now concentrate on the most critical paths. Stage 2 of our placement system again works to improve the slacks on a set of critical paths instead of the entire design.

4.3 Stage II of Placement

Here again we resort to a path based approach similar to the one described in Chapter 3. The placer gets the worst N (=1000) critical paths from the timer after Stage 1. For each of the paths, we calculate an ideal delay allocation per timing arc by dividing the required signal arrival times at the endpoint by the number of the timing arcs in the path. If the actual delay of a net arc is greater than the allocated value we need to shorten it. If the delay of a net arc is greater than 80% of the delay allocated, then we increment the variable weight on the corresponding placement graph edge using the path’s slack. The factor 0.8 is used so as to be more inclusive of nets that nearly meet the delay allocation as well as to prevent the elongation of these nets if left out.

Unlike the earlier critical path based approach we select cells for FD only from the cells that are part of the critical paths returned by the timer. This procedure is repeated until there is no improvement in WNS over a certain number of iterations.
Simplified pseudo code for the two-stage timing driven algorithm:

```
perform2StageTDPlacement
    iter_count = 0, best_iter_count = 0, best_wns = MIN_FLT
Start Stage 1
    performFDPlacement
    Update timing
    If (WNS > best_wns)
        best_wns = WNS
        best_iter_count = iter_count
        Save current design as best_design
    ELSE IF (iter_count – best_iter_count > TIME_OUT_COUNT)
        Break from Stage 1
    ELSE IF (TNS/Num endpoints < WNS/2)
        Break from Stage 1
    END IF
END IF
FOR each cell Ci
    Find worst slack Sij among the input pins of Cj connecting to Ci
    Calculate variable weight for Edge CiCj using Sij
END FOR
Increment iter_count and continue Stage 1
End Stage 1
iter_count = best_iter_count = 0
Start Stage 2
Get N critical paths
FOR each path
    Ideal delay allocation per segment = Required time/(Num points in path -1)
    Increment edge weight using path slack if Delay > (0.8 * ideal delay allocation)
END FOR
performFDPlacement for only cells in critical paths
Update timing
IF (WNS > best_wns)
    best_wns = WNS
```
4.4 Experimental Results

Fig. 4.2 Placement plot for the benchmark seq_align

We used the same set of benchmarks we had described earlier to test the new two-stage timing driven system. The worst path delays obtained for these benchmarks with the new approach was compared with those obtained using the simple timing driven approach and the base FD placement.
### 4.4.1 QOR Comparisons

<table>
<thead>
<tr>
<th>Benchmark (# of cells)</th>
<th>Worst Path Delay (ns)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FD (HPWL)</td>
<td>Simple TD</td>
<td>Two-stage TD</td>
</tr>
<tr>
<td>openfpu64 (14347)</td>
<td>2.4000</td>
<td>2.4025</td>
<td>1.5721</td>
</tr>
<tr>
<td>cordic (20577)</td>
<td>1.4238</td>
<td>1.3699</td>
<td>1.2652</td>
</tr>
<tr>
<td>reedsoldec (31179)</td>
<td>2.1619</td>
<td>1.7542</td>
<td>1.5572</td>
</tr>
<tr>
<td>seq_align (44098)</td>
<td>4.2020</td>
<td>3.5869</td>
<td>2.9662</td>
</tr>
<tr>
<td>stormcore (87662)</td>
<td>5.0112</td>
<td>4.1455</td>
<td>3.5697</td>
</tr>
<tr>
<td>pairing (288622)</td>
<td>6.3827</td>
<td>4.7324</td>
<td>4.0742</td>
</tr>
<tr>
<td>jpegenc (344646)</td>
<td>7.9953</td>
<td>6.4769</td>
<td>5.2336</td>
</tr>
<tr>
<td>AES (428665)</td>
<td>7.9976</td>
<td>5.9885</td>
<td>5.1824</td>
</tr>
</tbody>
</table>

Table 4.1 QOR comparisons

We can see that this approach gives an average improvement of around 30% in the worst path delays for the benchmarks. These results are more consistent across the designs compared to the first timing driven approach that we experimented with.
To ensure that our results are in the ballpark with the actual post-route timing of the design, we also compared the timing results given in Table 4.1 for both FD and two-stage TD with the timing reported by PT for each case after routing. The final placed designs from each of the two approaches were routed with IC Compiler and timing reported on these design using PT to get the reference results. Figure 4.2 shows the comparison for FD and Figure 4.3 shows the comparison for two-stage TD. We can see from these figures that our timing results are well correlated with the actual routed timing of the designs in both the cases.
4.2.2 Runtime Comparisons

The runtime comparisons of the three approaches are shown in Table 4.2. Our two-stage timing driven approach could take up to 3x the runtime of the basic FD placement. These runtimes are design specific and could lower than 2x in some designs.

<table>
<thead>
<tr>
<th>Benchmark (# of cells)</th>
<th>Runtime (s)</th>
<th>Ratio of runtimes (Two-stage TD/FD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FD (HPWL)</td>
<td>Simple TD</td>
</tr>
<tr>
<td>openfpu64 (14347)</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>cordic (20577)</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>reedsoldec (31179)</td>
<td>45</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Time 1</td>
<td>Time 2</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>seq_align (44098)</td>
<td>47</td>
<td>86</td>
</tr>
<tr>
<td>stormcore (87662)</td>
<td>192</td>
<td>279</td>
</tr>
<tr>
<td>pairing (288622)</td>
<td>1120</td>
<td>1505</td>
</tr>
<tr>
<td>jpegenc (344646)</td>
<td>2027</td>
<td>2452</td>
</tr>
<tr>
<td>AES (428665)</td>
<td>1938</td>
<td>2803</td>
</tr>
</tbody>
</table>

Table 4.2 Runtime comparisons
5. Conclusion

5.1 Summary

In this work we have developed a timing driven placement system which takes an unplaced netlist as input and gives a placed design as output. Because of the timing driven nature of the placement approach, the resultant placement is optimized for timing results. In order to achieve this, we developed a relatively accurate and fast incremental timing analysis engine. This was integrated into the placement engine that we built so as to be able to get the most up to date timing information of the design at every stage of the placement. This timing information has been used to guide the placement engine further in the form of dynamic weights on the placement graph edges.

We implemented two timing driven approaches. The first one is a simple critical path slack based weight assignment where we look at a set of critical paths obtained from the timer and calculate the incremental weight change on the placement graph edges based on the path slacks. The second is two-stage approach where we make use of slack information at every node in the timing graph to come up the weights during placement. This tries to address the need to optimize the locations of all the cells in the early course of placement since we start with a random initial placement and restricting focus to a limited set of critical paths is not enough. Once the first stage stabilizes for the worst slack we move to the second stage were we selectively look to optimize a set of critical paths. The core placement algorithm in both implementations is force-directed placement.
We compared the two approaches with the base non-timing driven placement algorithm. The first approach seems to give close to 20% improvement in the worst path delay on the benchmarks we tested at the cost of sum runtime. The second approach gives even better results in terms of worst path delay compared to the first approach and seems to be more consistent across the various benchmarks. The runtime taken is much more and is of the order of 2 to 3 times the non-timing driven approach mainly because of the interactions with timing analysis engine. Although the runtime is larger, the timing driven approach presented could reduce the overall runtime in terms of timing closure because of the much improved timing quality of the results obtained.

5.2 Future Directions

There are mainly two directions in which this work can be further expanded. One is to improve the quality of placement in terms of the timing results and the other is to increase the efficiency and reduce the runtime.

In regards to quality of results, we could look at incorporating more stability into the algorithm by devising a more controllable approach. This could help the placement result to converge faster and prevent oscillations. This in turn also could improve the runtime. We could use a different algorithm for the base placement engine which could better exploit the integrated timer. We could also work on improving the accuracy of the timer. The use of more accurate delay models for net delay computation could be helpful.
The runtime could be reduced by modifying the placement algorithm to take better advantage of the incremental nature of the timer. Parallelization of certain aspects of the system can also help reduce runtime such as weight assignment using timing information and the timing analysis itself.
BIBLIOGRAPHY


