Section IV: Timing Closure Techniques

IBM Contributions to this presentation include:

- T.J. Watson Research Center
- Austin Research Lab
- ASIC Design Centers
- EDA Organization

* For more detailed information see references at the end of this presentation, which include a wide variety of IBM and External publications covering these areas.

Jan. 2003 ASPDAC03 – Physical Chip Implementation 1
Overview

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms (skip)
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques (skip)
- Congestion avoidance / mitigation techniques
- Routing optimization

Timing Closure

- Many aspects of a design contribute to performance, power, and density
  - Architecture / Logic Implementation
  - PD Design Style (Flat, Hierarchical, etc)
  - Clocking Paradigm / Test / Circuit Family
  - Floor Plan / Synthesis / Placement / Routing
- Design Automation for timing closure is more significant than ever before
  - Designs are larger
  - Wires are longer, invalidating statistical synthesis models, and requiring lots of buffers
  - Cycle times are more aggressive
Design Automation Tools are Individually Mature

- Timing analysis
- Synthesis / Technology mapping
- Placement / Routing
- Floor Planning
- Extraction / Analysis

Challenge is to integrate them into one cooperative application

Physical Synthesis

Netlist in

Place&route

synthesis

timing

Completed Design
Design Flow Evolution:

1. Tech independent optimization
2. Tech mapping
3. Timing correction

Integrated Placement, Synthesis & Routing

1. Physically aware optimizations
2. Physically aware timing correction
3. Timing / Noise aware routing

Purpose of this Section:

- Provide users with an intuitive feel of the inner workings of the major timing closure tools
- Demonstrate the advancements in timing closure tools technology via example designs
- Explore a variety of significant design choices
What you should expect:

- High level concepts presented are generally applicable across a wide range of tools / methodologies (ie: not IBM specific)
- Specific tool internals used in this tutorial are taken from IBM tools. They should provide a reasonable “feel” as to how things are done in the industry.

Worldwide ASIC/PLD Sales
Top 5 Suppliers for 2001

- IBM $ 2758 growth 1.2%
- Agere $ 1310 growth -43.5%
- LSI $ 1243 growth -38.2%
- NEC $ 1243 growth -35.2%
- XLIINX $ 1149 growth -26.3%

Revenue: Millions of U.S. Dollars
Source: Gartner Dataquest (March 2002)
### IBM ASIC Supplier #1 since 1999

<table>
<thead>
<tr>
<th>Year</th>
<th>NEC</th>
<th>LSI Logic</th>
<th>Fujitsu</th>
<th>Lucent</th>
<th>NEC</th>
<th>IBM</th>
<th>IBM</th>
<th>IBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>NEC</td>
<td>NEC</td>
<td>Lucent</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
</tr>
<tr>
<td>1997</td>
<td>LSI Logic</td>
<td>IBM</td>
<td>IBM</td>
<td>Lucent</td>
<td>NEC</td>
<td>LSI Logic</td>
<td>LSI Logic</td>
<td>NEC</td>
</tr>
<tr>
<td>1998</td>
<td>Fujitsu</td>
<td>NEC</td>
<td>NEC</td>
<td>LSI Logic</td>
<td>NEC</td>
<td>LSI Logic</td>
<td>NEC</td>
<td>NEC</td>
</tr>
<tr>
<td>1999</td>
<td>Lucent</td>
<td>Fujitsu</td>
<td>LSI Logic</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
</tr>
<tr>
<td>2000</td>
<td>IBM</td>
<td>LSI Logic</td>
<td>Fujitsu</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
<td>IBM</td>
</tr>
<tr>
<td>2001</td>
<td>TI</td>
<td>Altegra</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
</tr>
<tr>
<td></td>
<td>Toshiba</td>
<td>VLSI</td>
<td>Xilinx</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
<td>Altera</td>
</tr>
<tr>
<td></td>
<td>Xilinx</td>
<td>Toshiba</td>
<td>TI</td>
<td>STM</td>
<td>Toshiba</td>
<td>Toshiba</td>
<td>Toshiba</td>
<td>Toshiba</td>
</tr>
<tr>
<td></td>
<td>Hitachi</td>
<td>Altera</td>
<td>Toshiba</td>
<td>VLSI</td>
<td>STM</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
</tr>
<tr>
<td></td>
<td>VLSI</td>
<td>Xilinx</td>
<td>VLSI</td>
<td>TI</td>
<td>TI</td>
<td>Agilent</td>
<td>Agilent</td>
<td>Agilent</td>
</tr>
</tbody>
</table>

### Section Outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization
Static Timing Analysis

Timing Analysis Basics:

Why static timing since simulation is more accurate?

- Simulation has a number of key drawbacks
  - requires input state vectors
  - long runtimes

- A simple example:

  How would one calculate the worst case rising delay from a to z?

<table>
<thead>
<tr>
<th>b=0</th>
<th>b-z delay</th>
<th>a-z delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>b-z delay1</td>
<td>a-z delay2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b=1</th>
<th>b-z delay</th>
<th>a-z delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>b-z delay3</td>
<td>a-z delay4</td>
<td></td>
</tr>
</tbody>
</table>

  Exponential explosion as possible design input states grow!
Timing Analysis Basics:
Definition of basic terms

- Arrival time (AT) -- the time at which a pin switches state
- Slew -- the rate at which a signal switches
  - usually difference of 10% and 90% on voltage curve

\[ \text{slew} = \text{time}_{90} - \text{time}_{10} \]
\[ \text{AT} = \text{time}_{50} \]

- Required arrival time (RAT) -- the time a signal *must* arrive at in order to avoid a chip fail
- Slack = Required arrival time - Arrival time
  - Positive slack good, negative slack bad

Example Problem: What is slack at PO?

Slack = -1
**Timing Analysis Basics:**

- **What is Incremental Timing?**
  - Enabling small incremental changes without full retiming
  - Only direct fanin/fanout cone is processed

![Diagram of timing analysis](image)

**Early Mode Analysis**

- Definitions change as follows
  - longest becomes shortest
  - slack = arrival - required

![Diagram of early mode analysis](image)
Timing Correction

- Fix electrical violations
  - Resize cells
  - Buffer nets
  - Copy (clone) cells
- Fix timing problems
  - Local transforms (bag of tricks)
  - Path-based transforms

Local Synthesis Transforms

- Resize cells
- Buffer or clone to reduce load on critical nets
- Decompose large cells
- Swap connections on commutative pins or among equivalent nets
- Move critical signals forward
- Pad early paths
- Area recovery
Transform Example

Delay = 4

Double Inverter Removal

Delay = 2

Resizing

A

B

C

0.026

0.035

0.2

0.2

0.2

0.3

Jan. 2003 ASPDAC03 - Physical Chip Implementation
Cloning

Buffering
Redesign Fan-in Tree

Redesign Fan-out Tree

Longest Path = 5

Longest Path = 4
Slowdown of buffer due to load
Decomposition

Swap Commutative Pins

Simple Sorting on arrival times and delay works
Move Critical Signals Forward

Based on ATPG
- linear in circuit size
- Detects redundancies efficiently

Efficiently find wires to be added and remove.
- Based on mandatory assignments.

Section outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization
Placement Objective:

- Find optimal relative ordering of cells
  - minimize wire length and congestion
  - maximize timing slack

- Find optimal spacing of cells
  - eliminate wiring congestion problems
  - provide space for post placement synthesis
    - clock trees
    - buffer insertion
    - timing correction

- Find optimal Global Position

Optimal Relative Order:
To spread ...

.. or not to spread
Place to the left

... or to the right
Optimal Relative Order:

Without “free” space the problem is dominated by order

Placement Footprints:

Standard Cell:

Data Path:

IP - Floorplanning
Placement objectives are subject to User Constraints / Design Style:

- Hierarchical Design Constraints
  - pin location
  - power rail
  - reserved layers
- Flat Design w/Floor Plan constraints
- Fixed circuits
- IO connections
Floor planned Placement

Congestion MAP
Advantages of Hierarchy

- Design is carved into smaller pieces that can be worked on in parallel (improved throughput)
- A known floor plan provides the logic design team with a large degree of placement control.
- A known floor plan provided early knowledge of long wires
- Timing closure problems can be addressed by tools, logic design, and hierarchy manipulation
- Late design changes can be done with minimal turmoil to the entire design

Disadvantages of Hierarchy

- Results depend on the quality of the hierarchy. The logic hierarchy must be designed with PD taken into account.
- Additional methodology requirements must be met to enable hierarchy. Ex. Pin assignment, Macro Abstract management, area budgeting, floor planning, timing budgets, etc
- Late design changes may affect multiple components.
- Hierarchy allows divergent methodologies
- Hierarchy hinders DA algorithms. They can no longer perform global optimizations.
Physical Synthesis Flow

- Wire-load Models → Synthesized Netlist
- Unplaced
  - Physically “unaware” timing
  - Cleanup: Remove buffers, nominal power levels on gates
  - Initial “basic” placement
- Logical + Placement optimizations
  - Physically aware logic optimizations
  - For minimal wire-length, min-cut, Steiner tree estimates, physically aware timing
- Timing-driven placement
  - For minimal netweights, based on the timing of the net
  - Placed Netlist

Example of Logical + Placement Optimizations

- Start with a placed or unplaced netlist
- Do recursive partitioning
- During and following each partition action, apply logic optimizations such as:
  - timing corrections
  - rebuffering
  - repowering
  - cloning
  - pin swapping
  - move boxes
  - … etc
Summary of Placement Methods

- **Simulated annealing**
  - (+) High-quality, arbitrary objectives and constraints, parallelizable, easy to implement
  - (-) Doesn’t scale

- **Quadratic (or, “analytic”)**
  - (+) Mathematically clean, fast (ConjGrad) solvers
  - (-) Solving “the wrong problem”, highly illegal solutions must be legalized, fixed “anchors” needed
  - Example: Alpert, Nam, Villarubia QUAD+ACG placer (ICCAD-02)

- **Partitioning-based**
  - (+) Fastest, scales well if multilevel used, good quality
  - (-) Must be heavily tuned (hMetis, MLPart), difficult to constrain, unstable results (same quality but different structure) (?)
  - Example: Capo (http://gigascale.org/bookshelf/)

Section Outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization
Overview of Common Placement Algorithms:

- Simulated Annealing
- Quadratic Placement
- Partitioning

Simulated Annealing:

```c
for(temp=high; temp > absolute_zero; temp -= increment)
{
    make a random move
    score the move
    use temp dependent probability to decide to accept or reject
}
```

Note: Clustering can be used to improve performance
Annealing:

Pro's:
- ease of implementation, dumb moves / smart scoring
- can easily accommodate new constraints - just add them to the scoring function
- great quality
- can be made to run on parallel processors

Con's:
- very long run time

Quadratic Placement
Cost = \((x_1 - 100)^2 + (x_1 - x_2)^2 + (x_2 - 200)^2\)

\[
\frac{\partial}{\partial x_1} \text{Cost} = 2(x_1 - 100) + 2(x_1 - x_2)
\]

\[
\frac{\partial}{\partial x_2} \text{Cost} = -2(x_1 - x_2) + 2(x_2 - 200)
\]

Setting the partial derivatives = 0, we solve for the minimum Cost:

\[
Ax + B = 0
\]

\[
\begin{bmatrix}
4 & -2 \\
-2 & 4
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
\begin{bmatrix}
200 \\
-400
\end{bmatrix}
= 0
\]

\[
\begin{bmatrix}
2 & -1 \\
-1 & 2
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
\begin{bmatrix}
-100 \\
-200
\end{bmatrix}
= 0
\]

\[
x_1 = \frac{400}{3} \quad x_2 = \frac{500}{3}
\]

Interpretation of matrices A and B:

- The diagonal values \(A[i,i]\) correspond to the number of connections to \(x_i\).
- The off-diagonal values \(A[i,j]\) are 1 if object \(i\) is connected to object \(j\), 0 otherwise.
- The values \(B[i]\) correspond to the sum of the locations of fixed objects connected to object \(i\).
Why formulate the problem this way?

- Because we can
- Because it is trivial to solve
- Because there is only one solution
- Because the solution is a global optimum
- Because the solution conveys “relative order” information
- Because the solution conveys “global position” information

However:

- Solution is not legal
- Solution depends of fixed anchor points
- Solution does not minimize linear wire length, congestion, or timing
- Solution is generally highly overlapping w/ high density (ie needs to be spread out)
What does the solution look like?

- To get an intuitive feel for the solution, examine the relaxation method for solving $Ax + B = 0$
- Actual program implementation may use other solution methods (that are generally less intuitive).

Solution of Quadratic using Relaxation:
Constrained Solutions:

- Sometimes we want to solve for the minimum wire length subject to a constraint.
- Example: Using quadratic for partitioning, we may want the quadratic placement to be "centered"
Constrained Solutions

To minimize Cost = f(x) subject to a constraint g(x) = 0 we can use Lagrangian multipliers to modify the Cost function as follows:

\[
\text{Cost} = f(x) + \lambda g(x)
\]

\[
\frac{\partial}{\partial x} \text{Cost} = \frac{\partial}{\partial x} f(x) + \frac{\partial}{\partial x} \lambda g(x)
\]

Using CG as a constraint:

\[
g(x) = \left( \bigoplus_{i=1}^{n} s_i x_i \right) - CG
\]

where:

\[
s_i \text{ is the size of object } i
\]

\[
n \text{ is the number of objects}
\]

\[
\frac{\partial}{\partial x} g(x) = \frac{s_i}{N}
\]

where we use N to represent the constant \( \bigoplus_{i=1}^{n} s_i \).

We have already shown that

\[
\frac{\partial}{\partial x} f(x) = 0 \quad \text{leads to the system of equations} \quad A x + B = 0
\]

Therefore solving the constrained problem

\[
\frac{\partial}{\partial x} \text{Cost} = \frac{\partial}{\partial x} f(x) + \frac{\partial}{\partial x} \lambda g(x) = 0
\]

leads to:

\[
A x + B + \frac{\lambda}{N} = 0
\]

Constrained Solutions (cont):

To solve \( A x + B + \frac{\lambda}{N} = 0 \) we could use a packaged solver and add the additional unknown \( \lambda \) and equation \( CG = \bigoplus_{i=1}^{n} s_i x_i \) to our matrices and solve.

Here is an alternative way to solve the system:

by substitution we let \( x = x_u + 2 x_l \)

where \( x_u \) is the unconstrained solution (ie the solution to \( A x + B = 0 \))

Assuming we can solve the unconstrained problem, \( x_u \) is known.

By substitution we get:

\[
A (x_u + 2 x_l) + B + \frac{\lambda}{N} = 0
\]

which becomes:

\[
A 2 x_l + \frac{\lambda}{N} = 0 \quad \text{or} \quad A x_l + \frac{\lambda}{N} = 0
\]
Constrained Solutions (cont):

We need to solve: \( Ax_l + s_i N = 0 \)

Note: The A matrix is the same as the A matrix for the unconstrained solution. Since the A matrix is the netlist connectivity specification, we have A.

The B matrix here is \( \left[ s_i \right] \) instead of the sum of fixed location connects.

Interpretation:

The solution to \( Ax_l + s_i N = 0 \) can be obtained by modifying the original netlist and placement such that:

1. All fixed objects are moved \( x = 0 \)
2. A constant force vector is applied to each object. The constant force vector for the \( i \)th object has magnitude \( s_i N \)

Then use the same solver as was used to solve \( Ax + B = 0 \)

Constrained Solutions (cont):

We also need to solve for \( \Theta \)

From the CG relationship and \( x_{ui} + \Theta_i \) we get:

\[ CG = \sum_{i=0}^{N} s_i (x_{ui} + \Theta_i) \leq N \]

\[ N = \sum_{i=0}^{N} s_i (x_{ui} + \Theta_i) \]

since we have solved for \( x_{ui} \) and \( x_l \) the only unknown is \( \Theta \)

we get:

\[ \Theta = \frac{\sum_{i=0}^{N} s_i x_{ui}}{\sum_{i=0}^{N} s_i} \]
Constrained Solutions (summary):

To minimize $f(x)$ (the weighted squared cost function) subject to a CG constraint we do the following:

1.) Solve for $x_u$ by solving $Ax_u + B = 0$ using relaxation or some other method.

2.) Solve for $x_f$ as follows: $A_{xf}x_i = 0$
   - Move all fixed objects to location $= 0$
   - Add a constant force vector to each object. The constant force vector for the $i$th object has magnitude $\frac{s_i}{N}$
   - Using relaxation or some other method, solve for $x_f$.

3.) Solve for $x_l$ using $\frac{\Delta CG - \Delta x_{CG}}{\Delta x_{CG}} = s_i$

4.) Compute the final placement using $x = x_u + x_f$.

Review:

From the previous example we know that the solution to:

$Ax_u + B = 0$

$x = \begin{bmatrix} 133.13 \\ 100.67 \end{bmatrix}$

with this solution the CG is $\frac{(100.56666667)(133.13)}{110} = 163.64$ (not 150).

Now we need to solve:

$A_{xf}x_i = 0$ which is the same as solving $A_{xf}x = 0$.

Recall that the B matrix represents the position of fixed objects. So, this equation represents the solution to:

$x = 0$
Constrained Solutions (summary):

Advantages of this approach:

1.) The Solver data structure is the netlist only 
   ie. no additional memory requirements
2.) Sometimes the unconstrained solution is by itself sufficient, 
   therefore we can avoid the additional overhead of producing 
   the constrained solution
3.) The numerical iterations in this method are NOT dependent on 
   the CG. We can solve for xu and xl, then try many different CG points 
   at very low cost.

Quadratic Techniques:

Pros:
- mathematically well behaved
- efficient solution techniques find global optimum
- great quality

Cons:
- solution of Ax + B = 0 is not a legal placement, so generally 
  some additional partitioning techniques are required.
- solution of Ax + B = 0 is that of the "mapped" problem, ie 
  nets are represented as cliques, and the solution minimizes 
  wire length squared, not linear wire length unless additional 
  methods are deployed
- fixed IOs are required for these techniques to work well
Partitioning

Partitioning:

Objective:

Given a set of interconnected blocks, produce two sets that are of equal size, and such that the number of nets connecting the two sets is minimized.
FM Partitioning:

list_of_sets = entire_chip;
while(any_set_has_2_or_more_objects(list_of_sets))
{
    for_each_set_in(list_of_sets)
    {
        partition_it();
    }
    /* each time through this loop the number of */
    /* sets in the list doubles. */
}

Object Gain: The amount of change in cut crossings
that will occur if an object is moved from
its current partition into the other partition

- each object is assigned a
  gain
- objects are put into a sorted
  gain list
- the object with the highest gain
  from the smaller of the two sides
  is selected and moved.
- the moved object is “locked”
- gains of “touched” objects are
  recomputed
- gain lists are resorted
FM Partitioning:
Partitioning:

Pros:
- very fast
- great quality
- scales nearly linearly with problem size

Cons:
- non-trivial to implement
- very directed algorithm, but this limits the ability to deal with miscellaneous constraints

FM Partitioning

- For large designs min-cut (FM) produces poor results

To Compensate, there are two widely used enhancements:

1.) Quadratic seeding
2.) Multi-Level partitioning
Partitioning:

```
while(there are clusters)
{
    partition_it;
    remove 1 cluster layer;
}
partition_it;
```

Global Placement - Multi-Level Partitioning:
MLP/FM Partitioning Cons:

- Does not know how to handle “free” space
- Results tend to be erratic, i.e., results from run to run have significant variation
MLP/FM Partitioning Pros:

- Handles designs that have no fixed connection points
- Very fast - can handle large designs

Hybrid Techniques

- Use both MLP and Quadratic techniques
- Results are more predictable due to quadratic cost function
- Partitioning is used for overlap removal
- Quadratic is used for “free” space handling and some relative order indications
Quadratic Partitioning
Analytical Constraint Generation

- Combine Quadratic techniques with MLP
- Use Quadratic solution to determine global position (i.e., balance)
- Use MLP to determine relative ordering of cells
Analytical Constraint Generation

Capacity = 2
ACG solution

Capacity = 2
Area = 1
Analytical constraint

Analytical Constraint Generation

Jan. 2003 ASPDAC03 - Physical Chip Implementation 129
Side by Side Comparison:

Original

ACG

Jan. 2003  ASPDAC03 - Physical Chip Implementation  154
Observations on Quadratic Placement

- placements are predictable and repeatable
- timing is inherently better
- wire length is not the best, but good
- run time: slower than MLP by 4x
- run time: faster than annealing by 4x
- excellent “free space” handling
- placements “feel” similar to those produced by annealing

Repeatability Example:

- One circuit
- Minimum linear length occurs for all solutions where y=50, 0 < x < 100
- Minimum quadratic length occurs for y=50, x=50
- Quadratic solution IS both minimum linear and minimum quadratic length

Jan. 2003 ASPDAC03 - Physical Chip Implementation 157
Section Outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization

Synthesis - Placement Interface
Partitioning Algorithm: Partition & Reflow

- Read Data
- Preprocessing
- While (any partition has > 2 cells)
  - Divide each Partition
  - Reflow across partitions
- Detailed Placement
- Done

What Synthesis Can do when Invoked:
- add boxes
- delete boxes
- add nets
- delete nets
- reconnect nets
- change box sizes
- query placement locations of boxes
- query "bin" statistics
- remove a box from a bin
- add a box to a bin
Placement and Synthesis Integration

- Loosely coupled: (methodology coupling)
  - do some synthesis, then write out data
  - do some placement, then write out data
  - ... Repeat
- Interleaved: (placement & synthesis in same process)
  - do pre-pd synthesis
    - for each placement step redo synthesis
- Tightly coupled: (simultaneous P&S aware transforms)

Loosely Coupled Placement & Synthesis:

Characteristics:
- Placement is treated as a black box
- Multiple placement runs are made
  - re-synthesize
  - Generate Constraints
Interleaved Placement & Synthesis:

Characteristics:
- the placement flow is the same as in a placement only methodology
- in between each step of the placement progression, synthesis is invoked

Tightly Coupled

- Placement and synthesis algorithms become co-dependent
- Placement algorithms have awareness of synthesis activity
- Synthesis algorithms have awareness of placement activity
Section Outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization

Summary of Techniques

- Placement-Driven X (PDX)
  - Cloning, Spreading, Sizing, Fanout Reclustering, …
  - (Constant-Delay Methodology)
- Buffer Insertion
  - Key problem: Max RAT at Source Interconnect Tree synthesis
  - Heuristic search over topologies + VanGinneken dynamic programming (with practical limits on polarity, buffer location, buffer library, etc. richness of formulation)
  - C-Tree (IBM), recent Q-Tree (UCSD), P/S/U-Tree (UIC), etc.
  - Early timing analysis (slew rate, cap load control): UCSD+IBM
- Buffer Block Planning + Buffered Global Routing
  - DAC-2001 Best Paper from IBM (buffer bays)
  - ASPDAC-2002 Best Paper from UCSD (delay-bounded floorplan evaluation with given buffer plan)
  - Primal-Dual Multi-Commodity Flow approximation
Placement Driven Cloning

Cloning to off-load non-critical path from critical path

Placement Driven Expansion

Expansion allows primitives to be placed in a more timing friendly way

AO

Logic

Logic

Logic

Logic

Expansion Transformation
Example:

Tightly Coupled Placement
Driven Expansion

Tightly Coupled Synthesis & Placement:
Tightly Coupled Synthesis & Placement Example:

Suppose the primary IO constraints look like this:

The placement of the synthesized netlist would look something like this:
Tightly Coupled Synthesis & Placement Example:
If we could re-synthesize the netlist, we could get something that looks like this.

Tightly Coupled Synthesis & Placement:
Tightly Coupled Synthesis & Placement example:

Map_TREE
For each cut
  partition_it
  For each partition
    If(partition number > M)
      {  
        if(related_node_count < N)
          merge_nodes
        if(related_node_count == 1)
          merge_node into neighbor
        partition
      }
    end
  end
end

Tightly Coupled Synthesis & Placement Example:
Tightly Coupled Synthesis & Placement Example:
Tightly Coupled Synthesis & Placement Example:
Tightly Coupled Synthesis & Placement Example:
Placement Driven Timing Correction

Redesign Fan-in Tree

- Arr(a) = 4
- Arr(b) = 3
- Arr(c) = 1
- Arr(d) = 0
- Arr(e) = 0

- Arr(a) = 4
- Arr(b) = 3
- Arr(c) = 1
- Arr(d) = 0
- Arr(e) = 0

- Arr(a) = 4
- Arr(b) = 3
- Arr(c) = 1
- Arr(d) = 0
- Arr(e) = 0

- Arr(a) = 4
- Arr(b) = 3
- Arr(c) = 1
- Arr(d) = 0
- Arr(e) = 0
Placement Driven Repowering

- Repowering is traditionally done using load based cell characterization
- Placement changes continuously during partitioning
- Need high efficiency algorithms to do repowering in this environment
- Solution: Use Gain Based Formulation

Delay Models

Load based formulation:
\[ d = \frac{\beta \cdot C_{\text{out}}}{1 + \beta \cdot C_{\text{in}}} + p \]
\[ d = k_1 \cdot C_{\text{out}} + p \]

Gain based formulation:
\[ g = \frac{C_{\text{out}}}{C_{\text{in}}} \]
\[ d = \frac{C_{\text{out}}}{1 + \beta \cdot C_{\text{in}}} + p \]
\[ d = l \cdot g + p \]

Jan. 2003 ASPDAC03 - Physical Chip Implementation 187

Jan. 2003 ASPDAC03 - Physical Chip Implementation 188
Area vs Delay Centric

- Load Based Paradigm
  - (load-based delay eq.)
  - sized

  Know:
  - Size of each cell
  - Total Area ->
    - area centric

  Don’t know:
  - Wire loads
  - Delay of each cell
  - Delay of a path

  Estimation error is in the delay:
  - Local 'path based' property.

- Gain Based Paradigm
  - (gain based delay eq.)
  - sizeless

  Know:
  - The delay of each cell.
  - The delay of a path ->
    - delay centric

  Don’t know:
  - Wire loads
  - The area of each cell
  - The total area

  Estimation error is in the area
  - Global property.

Design Flow

1. High Level Synthesis
2. Restructuring
3. Tech Mapping
4. Gain Based Opt
5. Discretization
6. Late Timing Corr

Load Based Delay (DCL)

Gain Based Delay
Power Levels (Gate Sizes)

Library (Gain) Analysis

\[ g = \frac{C_{out}}{C_{in}} \]

\[ d = 1.0 \times g + p \]
Area and Load Calculation

Start at primary outputs/ register inputs.
Much like static timing analysis.
Incremental.

Gain Calculation

Minimize $D$ such that: $G = \frac{C_{out}}{C_{in}}$

Minimize $D = \sum_{i=1}^{N} f_i + \sum_{i=1}^{N} p_i$

Solution: $f = f$
Example I

\[ C_{in} = 0.19 \]

NAND2: \( d = 0.0308 + 0.011 \times \frac{C_2}{C_0} \)

NOR2: \( d = 0.0496 + 0.021 \times \frac{C_2}{C_0} \)

INV: \( d = 0.0295 + 0.009 \times \frac{C_{out}}{C_3} \)

\[
F = f_1 f_2 f_3 = f^3 = L \frac{C_{out}}{C_{in}} - 0.00008
\]

\[ f = 0.0203 \]

Constant Delay Calculation

Inverter:
\[ d = l \cdot g + p \]
Set: \( g_e = 3.6 \)
Calculate: \( C_{out} = \frac{g_e}{C_{in}} \)
Measure: \( d_c \)
Set: \( C_{out} = 0 \)
Measure: \( d_o = p \)
Calculate: \( l \cdot g_e = d_c - p = f_c \)

Other gates:
\[ l \cdot g = f_c \]
\[ g_{nand} = 2.5 \]
\[ g_{nor} = 1.8 \]
\[ d_c = d_o = l \cdot g_{nand} + p_{nand} \]
Discretization

- From gain-based model back to appropriate power levels
- There is an error in timing/load when 'ideal' power levels are not available.
  - Goal: Minimize this error.
  - Can be tuned to delay error or capacitance error.

\[
\begin{align*}
C_1 &= \frac{C}{g_1} \\
C_2 &= \frac{C}{g_2} \\
C_3 &= \frac{C_{out}}{g_3}
\end{align*}
\]

\[d = l \cdot g + p\]

Gain Based: Observations:

- Gain Based algorithms: A major improvement.
  - More homogeneous (global) algorithms and designs.
  - Can be better targeted for area and/or delay.

- Reveal inherent cell characteristics to optimization tools, leading to improved QOR

- Good library design is required to facilitate discretization step

- Ideally suited for operation within Physical Synthesis
Placement Driven Buffering

Rip Out all Buffers

Insert Buffers based on placement info

What to do About Long Wires?

- Add buffers
- Tune wire sizes
- Modify the placement to reduce them
Placement Driven vs Logic Driven Buffer Insertion

- Logic driven buffer insertion focuses on logic topology and buffer sizing while assuming a statistical wire load model
- Placement driven buffering uses an existing placement as the fundamental constraint

Placement Driven Buffer Insertion: Buffopt (IBM)

- Multiple buffer types
- Inverters
- Capacitance, Slew and Noise constraints
- Wire Sizing
- Simultaneous driver sizing
- High order interconnect delay and $C_{\text{effective}}$
- Blockage handling
How Do Buffers Help?

- Reduce delay
  - Wire delay quadratic in length
  - Buffers make delay essentially linear
  - Delay gate dominated, not wire dominated
- Fix other problems
  - Bad slews at sinks
  - Capacitance range violations
  - Noise induced by capacitance coupling

How Does Wire Sizing Help?

- Highly resistive lines increase delay
- Wider wires or thick metal layers reduces resistance, but can increase capacitance
- For long interconnect, resistance reduction outweighs capacitance increase
Simple Buffer Insertion Problem

Given: Source and sink locations, sink capacitances and RATs, a buffer type, source delay rules, unit wire resistance and capacitance

Find: Buffer locations and a routing tree such that slack at the source is minimized

\[ q(s_0) = \min_{1 \leq i \leq 4} \{ RAT(s_i) - delay(s_0, s_i) \} \]
Fundamental Buffer Insertion

- Van Ginneken’s dynamic programming algorithm
- Building block: candidate (Cap, slack)
  - Candidates for each node stored as a list
  - Each sink has one candidate
  - Propagate candidates up the tree
- Guarantees optimal solution
- Quadratic complexity

Assumptions for the Basic Van Ginneken algorithm:

- Given a routing tree
- Given a set of potential insertion points
- Single buffer size
- No sink or driver sizing
- Linear gate delay model
  - $R_d C_{\text{down}} + K_d$
- Elmore wire delay model
  - $R_w \left( C_w/2 + C_{\text{down}} \right)$
Van Ginneken Extensions

- Multiple buffer types
- Inverters
- Capacitance, Slew and Noise constraints
- Wire Sizing
- Simultaneous driver sizing
- High order interconnect delay and C\text{effective}
- Blockage recognition

Example

- Connect the end points of the net using a steiner route
- Add Candidate Nodes
- Final buffer solution is optimal for this route, and this set of candidate nodes.
- Other routes may produce better final solutions.
- Net routing topology is an input to Van Ginneken's algorithm
Example

How Many Candidates?

- Number of candidates seems to double with each additional node

- Prune candidate with worst slack when capacitances is greater or equal
- Linear number of candidates
Pseudo Code:

List = NULL;
For each node (bottom up traversal of graph)
{
    augment each item in list with wire segment up to node
duplicate the list
    for each element of the duplicate list
        add a buffer at node
    analyze each element in list
    analyze each element in buffered (duplicate) list
    pick best element of buffered list and delete the rest
    new list is union of list and “best” element of buffered list
}
Pick best solution;

Example

Node 1 processing: 2 evaluations, at most 2 candidates kept
Node 2 processing: 4 evaluations, at most 3 candidates kept
Node 3 processing: 6 evaluations, at most 4 candidates kept
Node 4 processing: 8 evaluations, at most 5 candidates kept
Node 5 processing: 10 evaluations, at most 6 candidates kept
Node 6 processing: 12 evaluations, at most 7 candidates kept
    ◆ Now pick the best one: Optimal solution

\[
    \text{Num\_evaluations} = 2 \sum_{i=1}^{N} i = (N)(N+1)
\]

\[
    \text{Num\_candidates} \leq N + 1
\]
Merging Branches

Critical

Merge is additive

Van Ginneken Algorithm
Summary

- Good
  - Clever pruning controls # of candidates
  - Finds an optimal solution in quadratic time
  - Easily extended to cover a variety of important considerations (like multiple buffer types, wire sizing, polarity, slew, & capacitance constraints, etc.)

- Bad
  - Results depend on quality of route provided
Example Route:

Critical: cannot offload due to route

Different route leads to better solution

Physical Synthesis Flow

Wire-load Models \(\rightarrow\) Synthesized Netlist

Cleanup: Remove buffers, nominal power levels on gates

Initial "basic" placement

Logical + Placement optimizations

Timing-driven placement

Timing improvement?

Yes \(\rightarrow\) Placed Netlist

No more

For minimal wire-length, min-cut, Steiner tree estimates, physically aware timing

Physically aware logic optimizations

For minimal net weights, based on the timing of the net

Jan. 2003 ASPDAC03 - Physical Chip Implementation 217

Jan. 2003 ASPDAC03 - Physical Chip Implementation 218
Example Route:

If still critical, add net weight
Multiple Buffer Types

- Instead of one buffer type, can choose from \( m \) power levels
- Generate \( m \) candidates instead of one
- Still optimal
- Complexity increase quadratic in \( m \)

Inverters

- Store candidates in “+” and “-” lists
  - + implies polarity preserved
  - - implies polarity reversed
- Adding inverter
  - Switches candidate in + list to - list
  - Switches candidate in - to + list
- Final result only chosen from + list
Capacitance Constraints

- Each gate $g$ can drive at most $C(g)$ capacitance
- When inserting buffer $g$, check downstream capacitance.
- If it is bigger than $C(g)$, throw out candidate
- Increases efficiency

Slew Constraints

- Similar to capacitance constraints
- When inserting buffer, compute slews to gates driven by buffer
- If any slew exceeds its target, throw out candidate
- Potential difficulty: computing slew accurately in bottom-up fashion
Noise Constraints

- Each gate has acceptable noise threshold
- Compute cumulative noise for each wire via Devgan noise metric
- Throw out candidates that violate noise

Can avoid noise while optimizing timing!

Wire Sizing:

For each node (bottom up traversal of graph)
{
    for each Wire Size
    {
        augment each item in list with Sized wire segment
        duplicate the list
        for each element of the duplicate list
            add a buffer at node
        analyze each element in list
        analyze each element in buffered (duplicate) list
        pick best element of buffered list and delete the rest
        new list is union of list and “best” element of buffered list
    }
}

Do Final pruning & Pick best solution;
Blockage Recognition

Delete insertion points that run over blockages

Route Around Blockage
Buffer Bays

Routing Into Buffer Bays
“Buffer Site”
- Similar to buffer bays, only exact buffer locations are pre-specified, not just areas
- Useful as a mechanism for IP blocks and microprocessor design
- Dummy cell that holds a buffer
- Not connected to any net
- Becomes buffer when assigned to a net
- Extra sites → decoupling caps
- Sprinkle sites throughout design
- Allocate percentage within macros

Routing Into Buffer Sites
Generate Steiner Tree

Reduce Congestion and Coupling
Reduce Congestion and Coupling

Assign Buffers
Comments about Buffering and Wire Sizing:

- Extremely critical: One of the highest leverage timing closure items
- There are extended provably correct algorithms for dealing with the problem.
- Steiner route & Blockage avoidance are mostly heuristic: Hot research area!

Section Outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing Optimization
Congestion Mitigation

Sources of Congestion

- Placement Quality: Do we have a good relative ordering of cells?
- Placement Density: Do we have appropriate cell spreading?
- Preplacement of large cells: Is there a better location for these cells?
- Floorplan quality: Is this a good floorplan / hierarchy?
- Netlist complexity: Are some logic groupings inherently difficult to route
- Library characteristics: Do some cells block too much metal internally?
**Congestion Mitigation**

- **Constructive Avoidance**
  - control global placement pin density: fewer pins per unit area means fewer wires per unit area
  - monitor congestion during placement and perform dynamic spreading
- **Post placement fix up**
  - remove problems from an already placed netlist

**Constructive Avoidance:**

Characteristics:
- as placement is formed, take action to avoid problems
- between each step of the placement progression there is the potential to evaluate congestion and take action
Constructive Avoidance Deficiencies:

- Depends on early estimates of congestion that may not be accurate enough to avoid all problems
- Post placement actions such as clock tree insertion, repowering, buffering, etc may add congestion to the design
- Guard banding with conservative "constructive avoidance" causes lose of performance and density

Post Placement Congestion Mitigation

- Use production global router, not internal placement based global router
- Translate congestion values into density targets for placement regions
- Perform flow based circuit spreading
- Preserve relative logic ordering of cells
Network Flow Based Spreading

Min-cost max-flow formulation, similar to any “fix-up spreader”: “thermal” placement, Bonn’s top-down placer (Vygen), etc.

\[ \text{Supply Nodes } \quad b(i) > 0 \quad \text{Demand Nodes } \quad b(j) < 0 \]

- \( \forall i \) if \( b(i) > 0 \), \( \text{Cap}(eSi) = b(i) \)
- \( \text{Cost}(eSi) = 0 \)
- \( \forall j \) if \( b(j) < 0 \), \( \text{Cap}(ejt) = -b(j) \)
- \( \text{Cost}(ejt) = 0 \)

Jan. 2003 ASPDAC03 - Physical Chip Implementation

Congestion Driven Circuit Spreading

Initial Placement
- Calculate bin level congestion
- Translate bin score to bin target density
- Network flow based circuit spreading

Is Congestion below threshold?
- Yes → Final Placement
- No → Congestion Driven Circuit Spreading

Jan. 2003 ASPDAC03 - Physical Chip Implementation
We’ve Talked About

- Placement algorithms
- Placement / Synthesis interaction
- Placement aware synthesis techniques
- The Constant Delay paradigm
- Physical Buffer insertion / Wire sizing
- Congestion Mitigation

Let’s Look at some Examples:
Pure MLP

Quadratic

Optimization Results

Jan. 2003 ASPDAC03 - Physical Chip Implementation 253

Optimization Results

Jan. 2003 ASPDAC03 - Physical Chip Implementation 254
Optimization Results

Jan. 2003
ASPDAC03 - Physical Chip Implementation 255

Optimization Results

Jan. 2003
ASPDAC03 - Physical Chip Implementation 256
Section outline

- Introduction
- Review material (timing and synthesis)
- Introduction to placement
- Placement algorithms
- Paradigms for placement-synthesis integration
- Placement aware synthesis techniques
- Congestion avoidance / mitigation techniques
- Routing optimization

Routing Based Optimization: RBO (IBM)
Routing based Timing Closure Issues

- Post Routing timing problems can be significant
  - affect design schedule
  - may be too numerous to fix manually
- Increasing design density can reduce cost, but it also increases wiring congestion
  - timing and signal integrity become more significant
  - available resource for manual fixup is limited
  - without automation may not be doable
- Rerouting with constraints may resolve some of the problems, but this process is slow

Solution:

- Integrate global routing, detailed routing and timing correction
- Global routing is efficient enough to be run in an iterative timing closure loop
- Timing critical nets avoid scenic routes
- Non-critical nets that go scenic can be repowered and buffered prior to detailed routing
### Example Problem:

- **PDS Timing**
  - Uses Steiner wires - fast
  - Ideal "Steiner" routes

- **Post PD Timing**
  - Catches this problem: Slow!

- **RBO Timing Driven**
  - Routing sees this during global route stage: Fast!

- **Timing driven wiring solution**

- **Timing deficient wiring solution**

- **Force optimal use of wiring resource (e.g. critical paths get direct route)**

### Global Routing

Divides the entire chip into localized rectangular regions called tiles.

- Compress several pin locations in each tile to a single pin location

- All the shapes, wires, and opens are represented in terms of global track capacity and usage.
Global Routing

- Two step approach
  - Create the initial steiner routes
  - Compute the edge congestion's on the grid
  - Perform a rip-up reroute using shortest path algorithm to reduce the overall congestion of the design
- Advantages
  - Can communicate with detail router
  - Good correlation with final detail routing solution
RBO Extraction Process

- Very fast
  - Excellent correlation with final 3D extraction
- Uses global routes for extraction
- Neighbor information probabilistically determined based on the global routing congestion information
- Based on extraction tables

Probability of having a neighbor = (#OccupiedTracks)/(#Capacity) = 2/4 = 0.5

Capacity of All Edges = 5
Timing Critical Nets: Without RBO

Nets Routed with RBO flow
### RBO Results - Example 1

<table>
<thead>
<tr>
<th></th>
<th>Worst Slack</th>
<th>Slack Violations</th>
<th>Cap Violations</th>
<th>Slew Violations</th>
<th>Opens</th>
<th>Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steiner Estimates</td>
<td>-0.47</td>
<td>17</td>
<td>1</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLocal without RBO</td>
<td>-1.57</td>
<td>4687</td>
<td>14</td>
<td>128</td>
<td>50</td>
<td>87020</td>
</tr>
<tr>
<td>RBO Timing Closure (Global Routes)</td>
<td>-0.48</td>
<td>209</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed routing with RBO</td>
<td>-0.43</td>
<td>14</td>
<td>18</td>
<td>1</td>
<td>54</td>
<td></td>
</tr>
</tbody>
</table>

#### Example 2: - Critical Net Routed Without RBO

![Diagram showing example 2](image-url)
Example 2: Critical Net Routed With RBO

![Graph showing critical net routed with RBO]

Example 2 Results Summary

<table>
<thead>
<tr>
<th></th>
<th>Worst Slack</th>
<th>#Slack Violations</th>
<th>#Cap Violations</th>
<th>#Slew Violations</th>
<th>#Opens</th>
<th>#Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final routing</td>
<td>-0.54</td>
<td>1224</td>
<td>33</td>
<td>270</td>
<td>0</td>
<td>1152</td>
</tr>
<tr>
<td>without RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using RBO</td>
<td>-0.29</td>
<td>909</td>
<td>32</td>
<td>274</td>
<td>0</td>
<td>1070</td>
</tr>
</tbody>
</table>
Noise Detection and Avoidance:

- **RBO (Detection)**
  - **Length Base**
    - Initial selection includes length and slack threshold
    - Further pruning based on Worst Case Miller Timing
  - **Switching Window based refinement**
    - Pattern generation based on switching window overlaps

- **RBO (Avoidance)**
  - **Long Net Spreading**
  - **Track Reordering**
  - **Incremental Placement Changes**
  - **Layer Assignment**
Noise Detection and Avoidance:

- Wire width selection
- Physical Synthesis
  - Integration
  - Fix Cap And Slew Violations with Global Routes
  - Interface to RBO
  - Noise Alleviation Resizing
  - Noise Aware Buffering

Long Net Spreader
Wrap Up

- Timing closure today is highly dependant on integrated tools.
- Tightly integrated Placement, Timing & Synthesis tools are available today from multiple vendors.
- Placement techniques are dominated quadratic techniques and partitioning
- Next on the list for integration are Routing and Signal integrity tools (happening now)
- These tools have a high degree of complexity. It takes large well funded DA organizations to compete in this space.

Placement References

Synthesis References


Jan. 2003 ASPDAC03 - Physical Chip Implementation 279

DP Buffer Insertion References


Jan. 2003 ASPDAC03 - Physical Chip Implementation 280
Blockage Avoidance References


Interconnect Planning References

- An interconnect-centric design flow for nanometer technologies Cong, J. Proceedings of the IEEE , Volume: 89 Issue: 4 , April 2001 Page(s): 505 -528
- Planning buffer locations by network flows Tang, X.; Wong, D.F.; International Symposium on Physical Design, April 2001 Page(s): 180-185