EE 5301 – VLSI Design Automation I

Part IV: Floorplanning

Kia Bazargan

University of Minnesota

References and Copyright

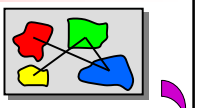
- Textbooks referred (none required)
  - [Mic94] G. De Micheli
    "Synthesis and Optimization of Digital Circuits"
    McGraw-Hill, 1994.
  - [CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest
    "Introduction to Algorithms"
    MIT Press, 1990.
  - [Sar96] M. Sarrafzadeh, C. K. Wong
    "An Introduction to VLSI Physical Design"
    McGraw-Hill, 1996.
  - [She99] N. Sherwani
    "Algorithms For VLSI Physical Design Automation"
    Kluwer Academic Publishers, 3rd edition, 1999.

References and Copyright (cont.)

- Slides used: (*Modified by Kia when necessary*)
  - [©Sarrafzadeh] © Majid Sarrafzadeh, 2001;
    Department of Computer Science, UCLA
  - [©Sherwani] © Naveed A. Sherwani, 1992
    (companion slides to [She99])
  - [©Keutzer] © Kurt Keutzer, Dept. of EECS,
    UC-Berekeley
    http://www-cad.eecs.berkeley.edu/~niraj/ee244/index.htm
  - [©Gupta] © Rajesh Gupta
    UC-Irvine
    http://www.ics.uci.edu/~rgupta/ics280.html
  - [©Kang] © Steve Kang
    UIUC
    http://www.ece.uiuc.edu/ece482/

Floorplanning

- Problem
  - Given circuit modules (or cells) and their connections, determine the *approximate* location of circuit elements
  - Consistent with a hierarchical / building block design methodology
  - Modules (result of partitioning):
    - Fixed area, generally rectangular
    - Fixed aspect ratio → hard macro (aka fixed-shaped blocks)
      fixed / floating terminals (pins)
      Rotation might be allowed / denied
    - Flexible shape → soft macro (aka soft modules)

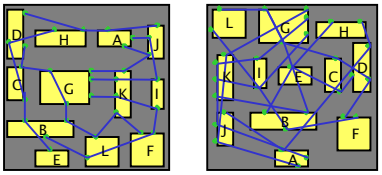


Floorplanning (cont.)

- Objectives:
  - Minimize area
  - Determine best shape of soft modules
  - Minimize total wire length
    - to make subsequent routing phase easy
      (short wire length roughly translates into routability)
  - Additional cost components:
    - Wire congestion (exact routability measure)
    - Wire delays
    - Power consumption
- Possible additional constraints:
  - Fixed location for some modules
  - Fixed die, or range of die aspect ratio
- NP-hard (what did you expect? ☺)

Floorplanning: Why Important?

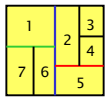
- Early stage of physical design
  - Determines the location of large blocks
    → detailed placement easier (divide and conquer!)
  - Estimates of area, delay, power
    → important design decisions
  - Impact on subsequent design steps (e.g., routing, heat dissipation analysis and optimization)



Figs: [©Sherwani]

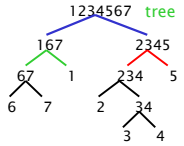
Floorplan Classes

- Slicing, recursively defined as:
  - A module OR
  - A floorplan that can be partitioned into two slicing floorplans with a horizontal or vertical cut line

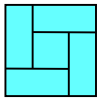


Slicing floorplan

Corresp. Slicing tree



- Non-slicing
  - Superset of slicing floorplans
  - Contains the "wheel" shape too.

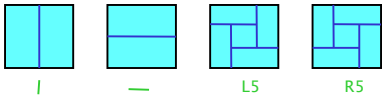


Non-Slicing floorplan

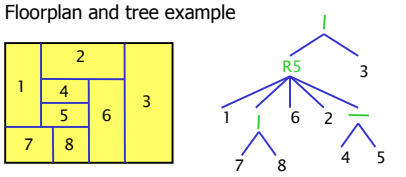
[©Sarrafzadeh]

Non-slicing Floorplan Example

- Hierarchical floorplan of order 5
  - Templates



  - Floorplan and tree example



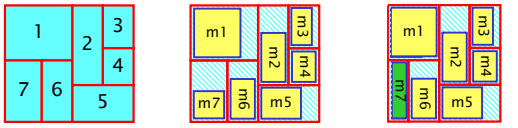
[©Sarrafzadeh]

Floorplanning Algorithms

- Components
  - "Placeholder" representation
    - Usually in the form of a tree
    - Slicing class: Polish expression [Otten]
    - Non-slicing class: O-tree, Sequence Pair, BSG, etc.
    - Just defines the *relative position* of modules
  - Perturbation
    - Going from one floorplan to another
    - Usually done using Simulated Annealing
  - Floorplan sizing
    - Definition: Given a floorplan tree, choose the best shape for each module to minimize area
    - Slicing: polynomial, bottom-up algorithm
    - Non-slicing: NP! Use mathematical programming (exact solution)
  - Cost function
    - Area, wire-length, ...

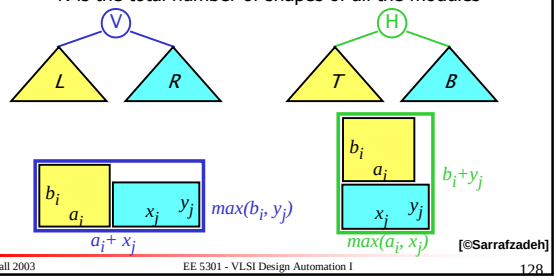
Area Utilization, Hard and Soft Modules

- The hierarchy tree and floorplan define "place holders" for modules
- Area utilization
  - Depends on how nicely the rigid modules' shapes are matched
  - Soft modules can take different shapes to "fill in" empty slots ➔ floorplan sizing



Floorplan Sizing for Slicing Floorplans

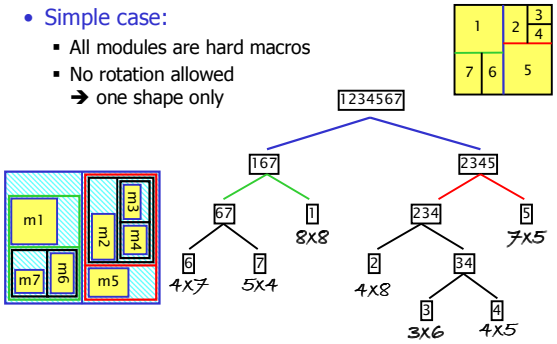
- Bottom-up process
- Has to be done per floorplan perturbation
- Requires O(n) time.
  - N is the total number of shapes of all the modules



[©Sarrafzadeh]

Sizing Slicing Floorplans

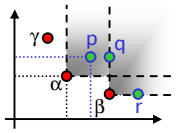
- Simple case:
  - All modules are hard macros
  - No rotation allowed ➔ one shape only



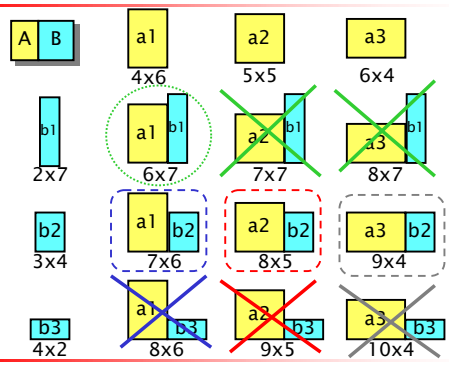
Sizing Slicing Floorplans (cont.)

- What if modules have more than one shape?
- If area only concern:
  - Module A has shapes 4x6, 7x8, 5x6, 6x4, 7x4, which ones should we pick?
  - Module A has shapes 4x6, 5x5, 6x4, which ones should we pick? (what if B is 3x5?!)

- Dominant points
  - Shape (x_1, y_1) dominates (x_2, y_2) if $x_1 \le x_2$ and $y_1 \le y_2$.



Sizing Slicing Floorplans: Example



Slicing Floorplan Sizing Algorithm

Procedure Vertical_Node_Sizing

Input: Two sorted lists L = { $(a_1, b_1), \dots, (a_s, b_s)$ },
R = { $(x_1, y_1), \dots, (x_t, y_t)$ }
where $a_i < a_j$, $b_i > b_j$, $x_i < x_j$, $y_i > y_j$ for all $i < j$
Output: A sorted list H = { $(c_1, d_1), \dots, (c_u, d_u)$ }
where $u \le s + t - 1$, $c_i < c_j$, $d_i > d_j$ for all $i < j$

```
begin-1
        H := ϕ
        i := 1, j := 1, k = 1
        while (i ≤ s) and (j ≤ t) do
        begin-2
                (c_k, d_k) := (a_i + x_j, max(b_i, y_j))
                H := H ∪ { (c_k, d_k) }
                k := k + 1
                if max(b_i, y_j) = b_i then i := i + 1
                if max(b_i, y_j) = y_j then j := j + 1
        end-2
end-1
```

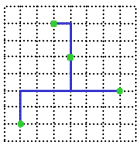
[©Sarrafzadeh]

Slicing Floorplan Sizing

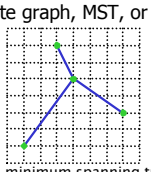
- Input: floorplan tree, modules shapes
- Start with sorted shapes lists of modules
- In a bottom-up fashion, perform:
  - Vertical_Node_Sizing
    AND
    Horizontal_Node_Sizing
- When get to the root node, we have a list of shapes. Select the one that is best in terms of area
- In a top-down fashion, traverse the floorplan tree and set module locations

Wire Length

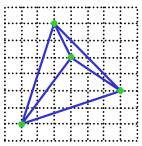
- For hyperedges:
  - Either of complete graph, MST, or Steiner tree



(a) Steiner tree (length = 13)



(b) minimum spanning tree (length = 11)



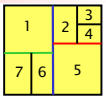
(c) complete graph (length = 32)

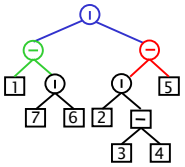
- For each edge:
  - Euclidian distance sqrt($(x_1-x_2)^2 + (y_1-y_2)^2$).
    - Direct lines
  - Manhattan distance $|x_1 - x_2| + |y_1 - y_2|$
    - Manhattan: Only horizontal / vertical lines

[©Sherwani]

Polish Expression

- Tree representation of the floorplan
  - Left child of a V-cut in the tree represents the left slice in the floorplan
  - Left child of an H-cut in the tree represents the top slice in the floorplan
- Polish expression representation
  - A string of symbols obtained by traversing a binary tree in post-order.

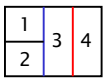


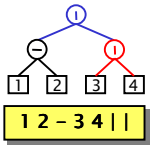


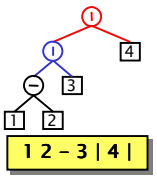
1 7 6 | - 2 3 4 - | 5 - |

Normalized Polish Expression

- Problem with Polish expressions?
  - Multiple representations for some slicing trees
    - When more than one cut in one direction cut a floorplan
  - Larger solution space
  - A stochastic algorithm (e.g., Simulated Annealing) will be more biased towards floorplans with multiple representations
    - (More likely to be visited)



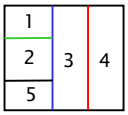


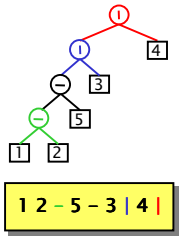


[©Sarrafzadeh]

Normalized Polish Expression (cont.)

- Solution?
  - Assign priorities to the cuts
  - In a top-down tree construction,
    - Pick the right-most cut
    - Pick the lowest cut
  - Result: no two same operators adjacent in the Polish expression (i.e., no "| |" or "— —")





Simulated Annealing

- Idea originated from observations of crystal formations (e.g., in lava)
  - A crystal is in a low energy state
  - Materials tend to form crystals (global minimum)
  - If at the right temperature (i.e., right speed), a molecule will adhere to a crystal formation
- Very slowly decrease temperature
  - When very hot, molecules move freely
    - When a molecule gets to a chunk of crystal, it *might* move away due to its high speed
  - When colder, molecules slow down
    - The probability of moving away from a local optimum decreases
  - When the material "freezes", all molecules are fixed and the material is in minimum energy state

Simulated Annealing Algorithm

- Components:
  - Solution space (e.g., slicing floorplans)
  - Cost function (e.g., the area of a floorplan)
    - Determines how "good" a particular solution is
  - Perturbation rules
    (e.g., transforming a floorplan to a new one)
  - Simulated annealing engine
    - A variable T, analogous to temperature
    - An initial temperature T_0 (e.g., T_0 = 40,000)
    - A freezing temperature T_{freez} (e.g., T_{freez}=0.1)
    - A cooling schedule (e.g., T = 0.95 * T)

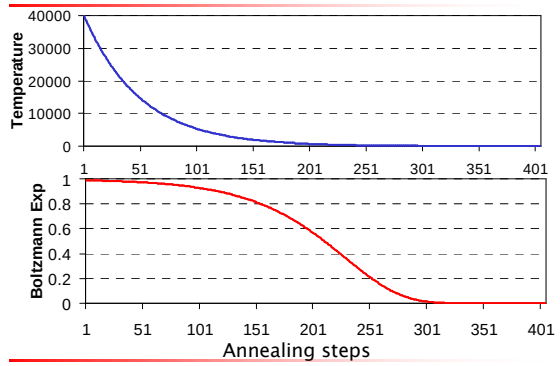
Simulated Annealing Algorithm

```
Procedure SimulatedAnnealing
    curSolution = random initial solution
    T = T0                          // initial temperature
    while (T > Tfreez) do
        for i=1 to NUM_MOVES_PER_TEMP_STEP do
            nextSol = perturb (curSolution)
            Δcost = cost(nextSol) - cost(curSolution)
            if acceptMove (Δcost, T) then
                curSolution = nextSol      // accept the move
        T = coolDown (T )
Procedure acceptMove (Δcost, T)
    if Δcost < 0 then return TRUE      // always accept a good move
    else
        boltz = e^(-Δcost / k T)       // Boltzmann probability function
        r = random(0,1)                // uniform rand # between 0&1
        if r < boltz then return TRUE
        else return FALSE
```

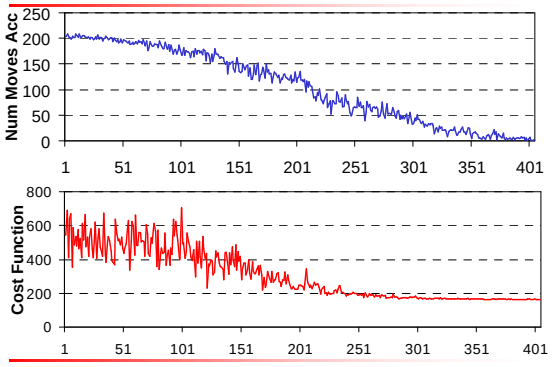

Simulated Annealing: Move Acceptance

- Good moves are always accepted
- Accepting bad moves:
  - When $T = T_0$, bad move acceptance probability ≈ 1
  - When $T = T_{freez}$, Bad move acceptance probability = 0
- Boltzmann probability function?!?
  - $boltz = e^{-\Delta cost / k T}$.
  - k is the Boltzmann constant,
    chosen so that all moves at the initial temperature
    are accepted

Simulated Annealing: More Insight...



Simulated Annealing: More Insight...

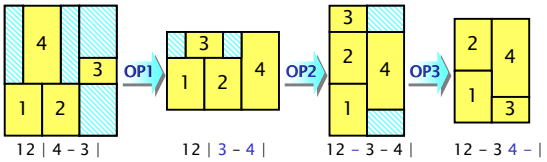


Wong-Liu Floorplanning Algorithm

- Uses simulated annealing
- Normalized Polish expressions represent floorplans
- Cost function:
  - cost = area + λ totalWireLength
  - Floorplan sizing is used to determine area
  - After floorplan sizing, the exact location of each module is known, hence wire-length can be calculated
- Perturbation?....

Wong-Liu Floorplanning Algorithm (cont.)

- Moves:
  - OP1: Exchange two operands that have no other operands in between
  - OP2: Complement a series of operators between two operands
  - OP3: Exchange adjacent operand and operator if the resulting expression still a normalized Polish exp.



12 | 4 – 3 | 12 | 3 – 4 | 12 – 3 – 4 | 12 – 3 4 – |

[©Sarrafzadeh]

Other Floorplanning Methods

- Rectangular dual graph
- Linear programming (floorplan sizing)
- Non-slicing methods
  - Sequence-pair
  - Bounded slice line grid
  - O-tree
  - Corner block list

To Probe Further...

- Andrew B. Kahng, "Classical Floorplanning Harmful?", Int'l Symposium on Physical Design (ISPD), pp. 207-213, 2000. (survey + future directions)
- F. Y. Young and D. F. Wong, "Slicing Floorplans With Range Constraint", Int'l Symposium on Physical Design (ISPD), pp. 97-102, 1999. (extension of Wong-Liu, some modules restrained to some regions)
- K. Bazargan, S. Kim and M. Sarrafzadeh, "Nostradamus: A Floorplanner of Uncertain Designs", Int'l Symposium on Physical Design (ISPD), pp. 18-23, 1998. (extension of Wong-Liu, notion of flexibility of a floorplan)

To Probe Further...

- H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani,
  "Rectangle-Packing-Based Module Placement",
  Conference on Computer Aided Design (ICCAD), pp. 472-479, 1995.
  ("Sequence Pair" non-slicing floorplan representation)
- X. Hong, G. Huang, Y. Cai, *et. al.*,
  "Corner Block List: an Effective and Efficient Topological Representation of Non-slicing Floorplan",
  Int'l Conference on Computer Aided Design (ICCAD), pp. 8-12, 2000.
  (non-slicing floorplan representation)
- Yingxin Pang , Chung-Kuan Cheng , Koen Lampaert , Weize Xie,
  "Rectilinear block packing using O-tree representation",
  Int'l Symposium on Physical Design (ISPD), pp. 156-161, 2001.
  (extension of the non-slicing floorplan representation "O-tree" to handle L-shaped modules)