Floorplan Design for Modern FPGAs

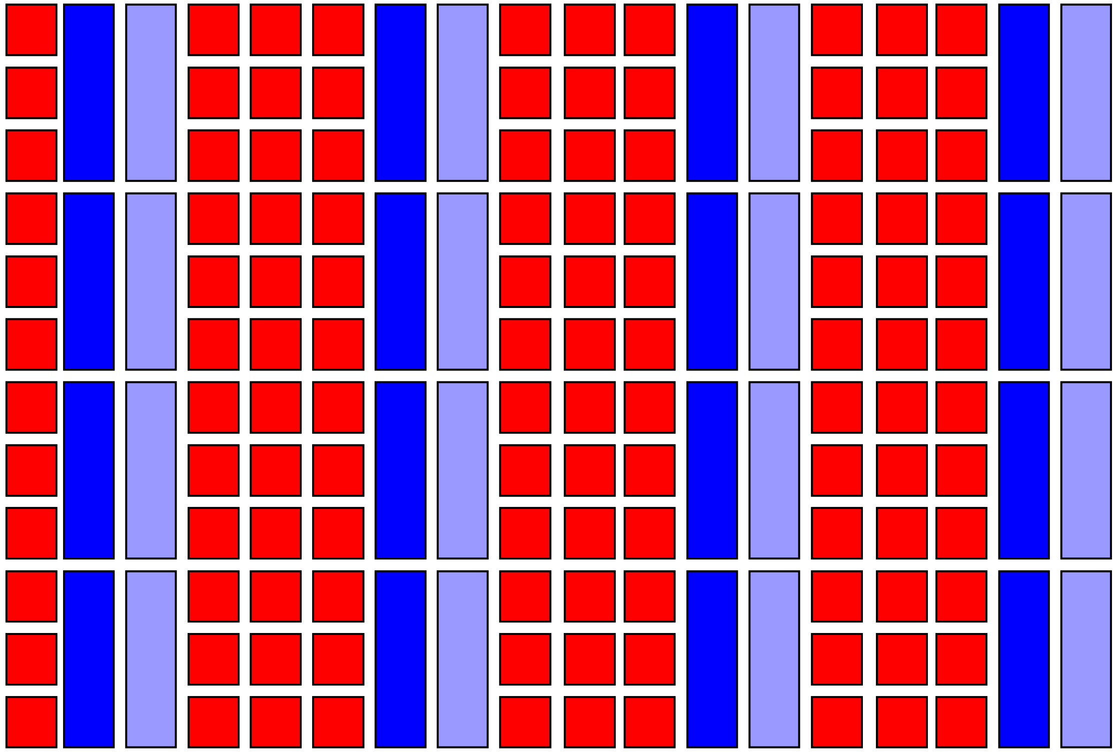
Outline

- Introduction
  - Floorplanning for FPGAs with heterogeneous resources
  - Realizations for a module
  - Irreducible realization list
- Algorithm
  - Calculate irreducible realization lists
  - Reduce complexity
- Compaction & Postprocessing
- Experimental results

Why FPGA Floorplanning?

- Floorplanning enables
  - a divide-and-conquer approach to the physical implementation of large designs
  - parallel compilation
- Floorplanning is also a required step in partial reconfiguration design flow

FPGA Architecture



🟥 CLB

🟦 RAM

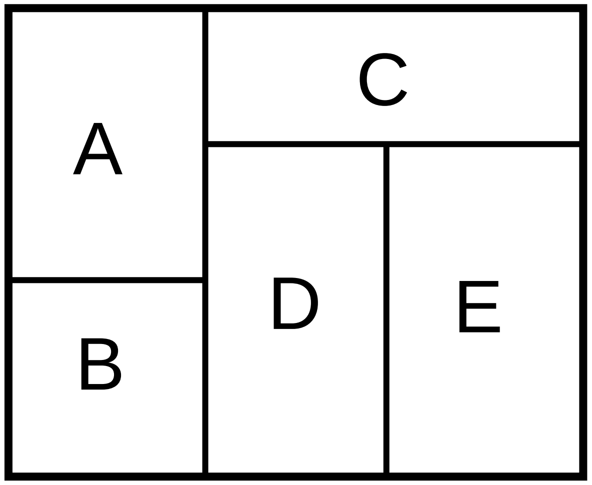
▨ Multiplier

Xilinx XC3S5000
- 8320 CLBs
- 104 RAMs
- 104 Multipliers

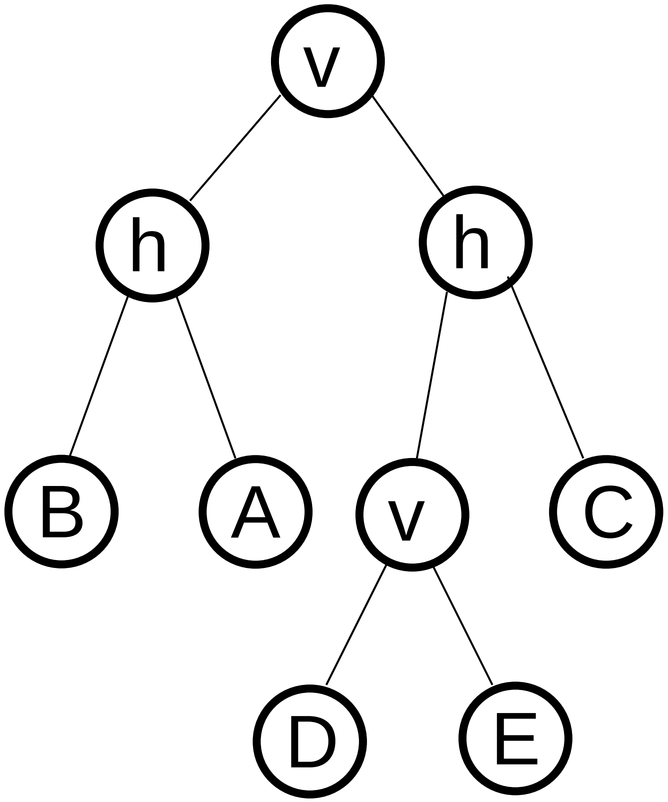
FPGA Floorplanning

- Place a set of modules onto an FPGA chip.
- Resource requirement vector of a module $<n_1,n_2,n_3>$
  - n_1 is the number of CLBs
  - n_2 is the number of RAMs
  - n_3 is the number of multipliers
- Place each module in a rectangular region satisfying its resource requirement.
- No overlapping among modules.

Slicing Floorplan



Slicing Floorplan

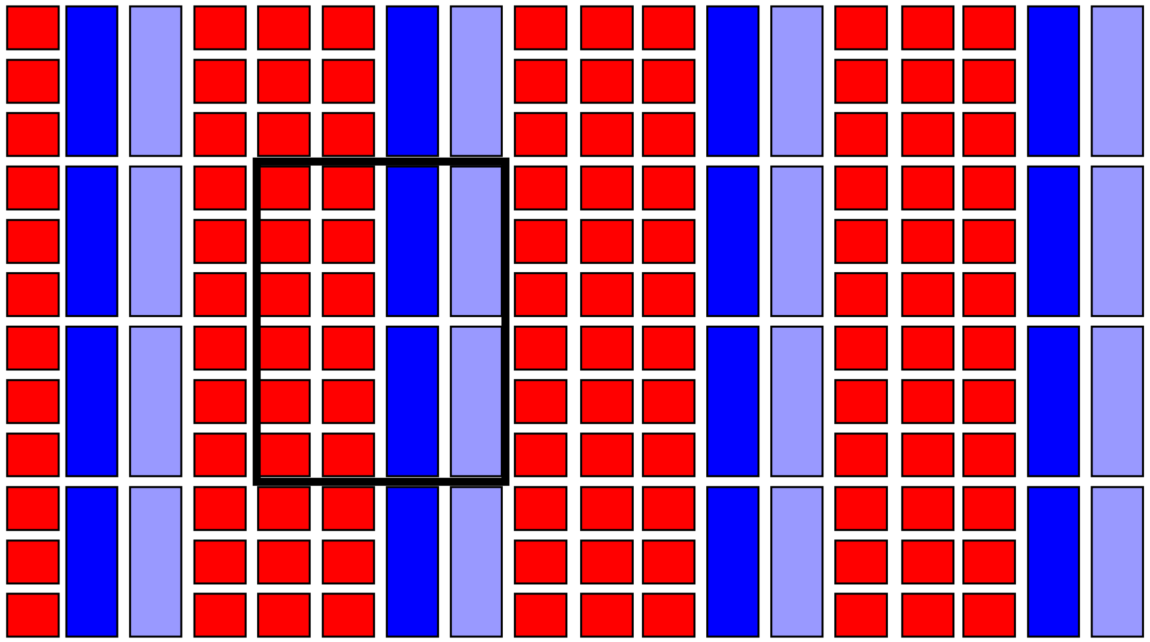


Slicing Tree

Module Realizations

- Employ a coordinate system on the chip.
- A realization *r* for a module θ is a rectangular region <x, y, w, h>
  - (x, y) is the coordinate of *r*'s lower left corner.
  - w is the width of *r*.
  - h is the height of *r*.
  - *r* satisfies the resource requirement of θ.

Example

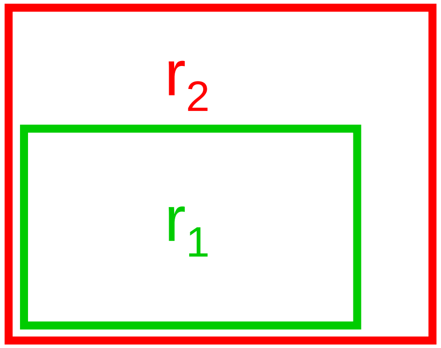


Resource requirement of module θ : <10,2,1>
(10 CLBs, 2 RAMs, and 1 Multiplier)

A realization for θ : r = <4,3,4,6>

Dominance Relation

- S_θ : set of all realizations for a module θ
- r_1, r_2 are two realizations from S_θ
  - r_1 dominates r_2 iff
    - $x(r_1) \geq x(r_2)$
    - $y(r_1) \geq y(r_2)$
    - $x(r_1) + w(r_1) \leq x(r_2) + w(r_2)$
    - $y(r_1) + h(r_1) \leq y(r_2) + h(r_2)$

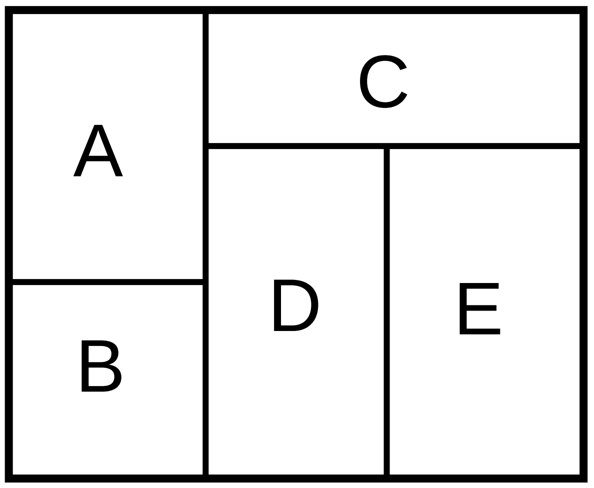


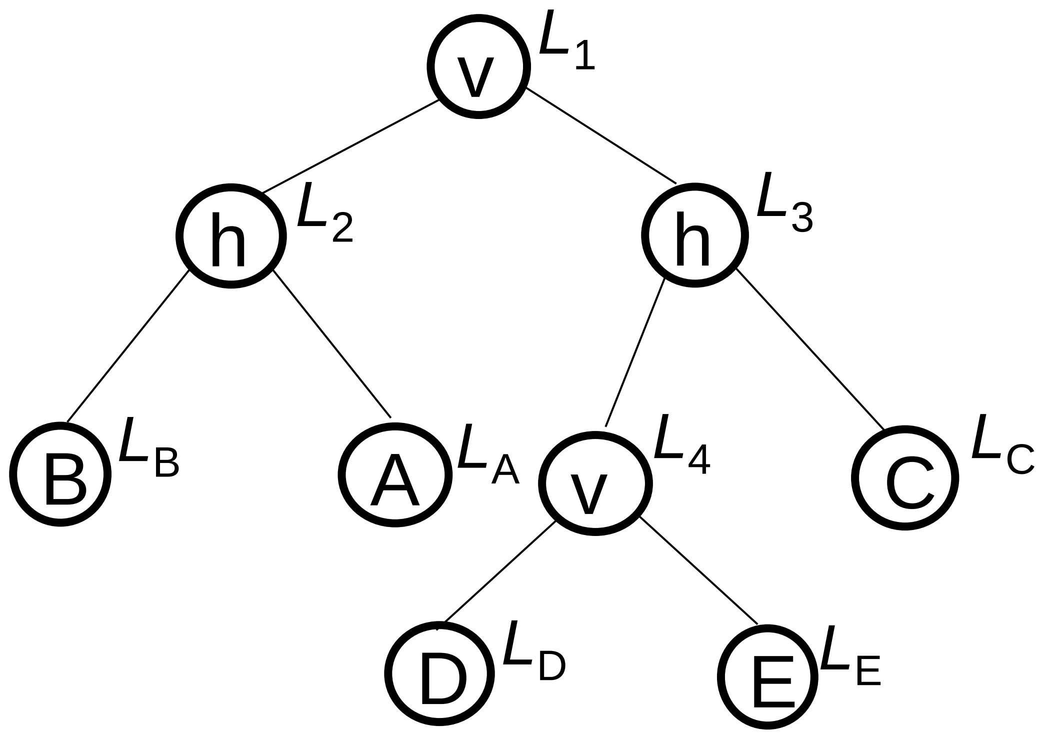
Irreducible Realization List (IRL)

- An irreducible realization list for module θ
  - L_θ(x, y) : a list of realizations
    - The starting point of each realization is (x,y).
    - No other realizations starting from (x,y) dominate any of these realizations.
- Realizations of L_θ(x, y) are sorted in decreasing height.

Irreducible Realization List (IRL)

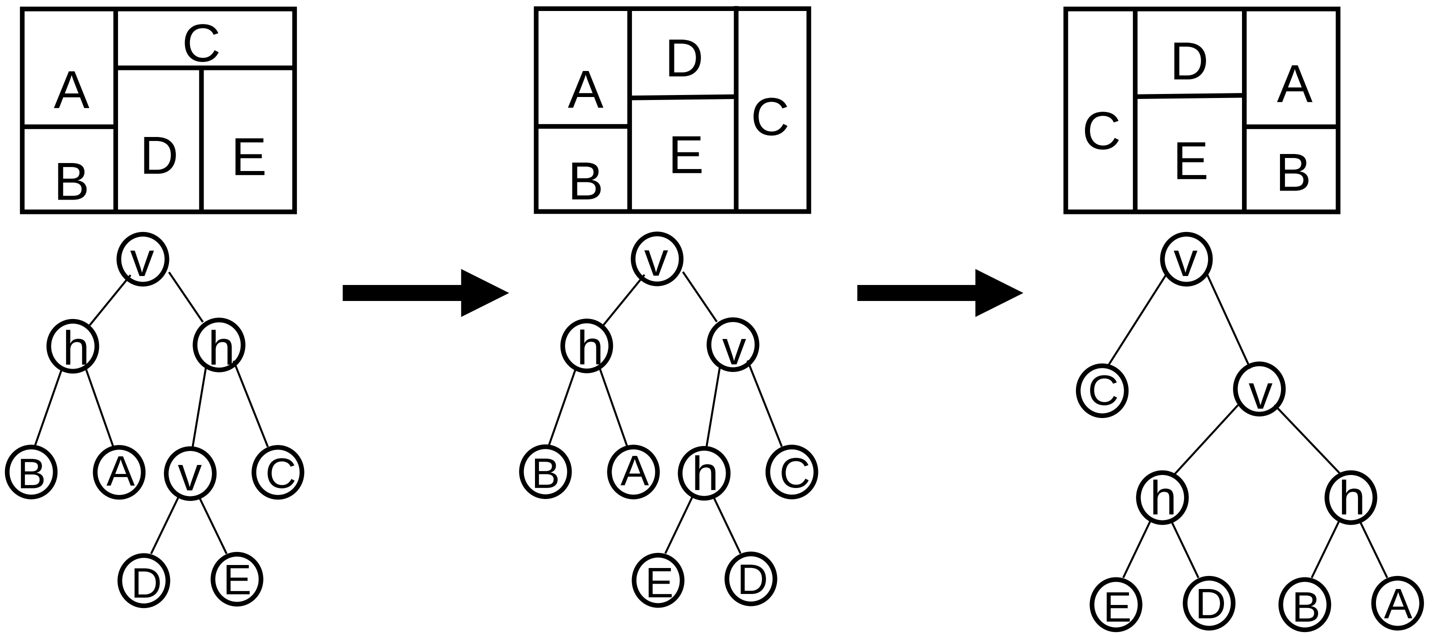
- IRLs for internal nodes (sub-floorplans)





Algorithm

- Simulated annealing (SA) searches on slicing floorplans.

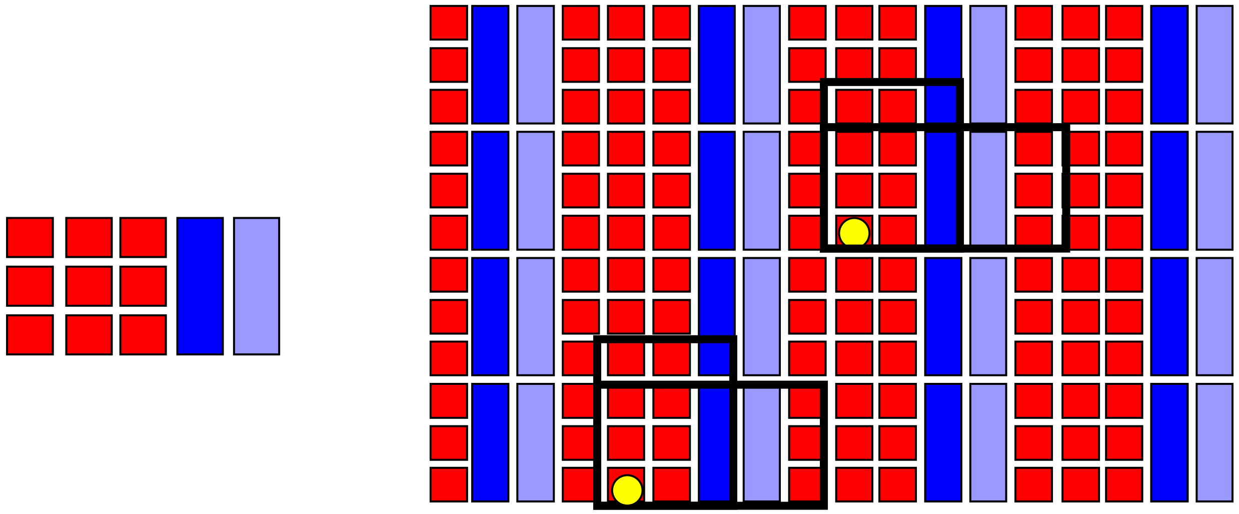


- The cost function :

$$\alpha Area + \beta WireLength$$

Reduce Space Complexity

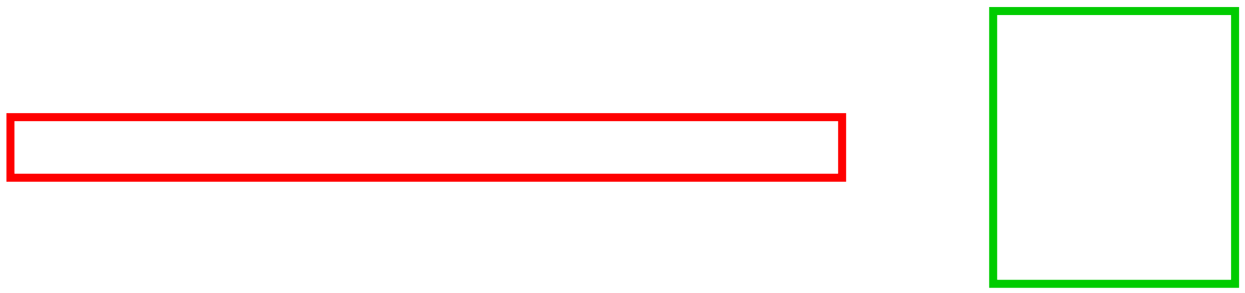
- Each FPGA chip is a two dimensional array of a basic pattern.
- We can get L_θ(x, y) for all (x, y) on a chip by computing IRLs for every point on the basic pattern only.



Resource requirement of module θ : <8,1,0>

Reduce Space Complexity

- It is practical to impose aspect ratio bounds.
- For performance consideration, we prefer each module to have aspect ratio close to 1.
  - Short internal wire length.



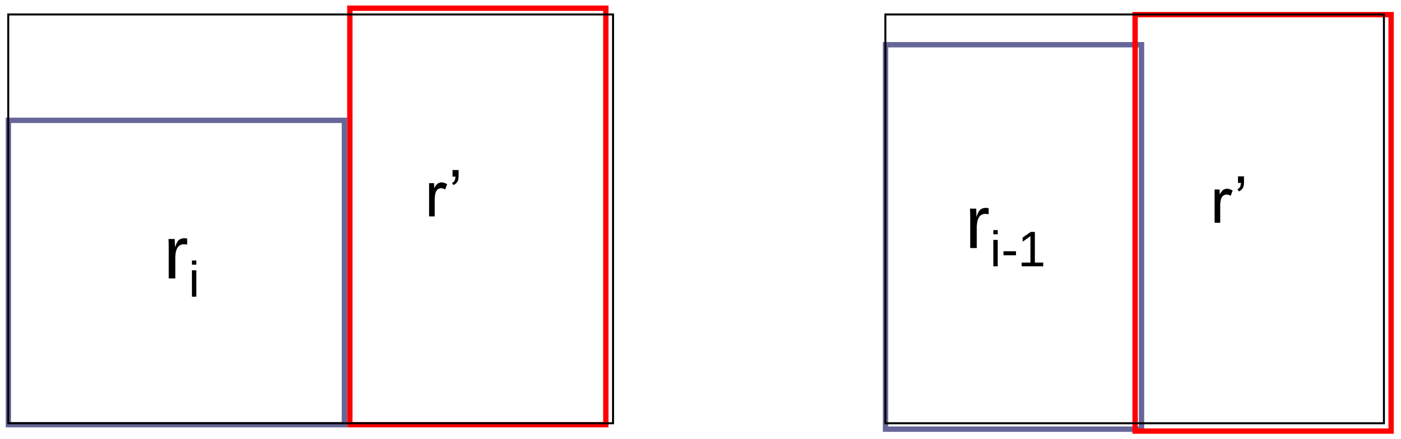
[Dr. Salil, *Multi-million gate FPGA physical design challenges,* ICCAD03]

Compute IRLs at Internal Nodes

- Consider computing IRLs for all points on the pattern for an internal node *u* that has a vertical cut.
- To compute IRL $L_u(x, y)$
  - Let $(r_1, r_2, \ldots , r_k)$ be the IRL of *u*'s left child *p* with starting point (x,y).
  - Combine r_i with a set of irreducible realizations of *u*'s right child *q*, which have heights between $h(r_i)$ and $h(r_{i-1})$.
  - Combine r_i with the irreducible realization of *q* with height less than but closest to $h(r_i)$.

Compute IRLs at Internal Nodes

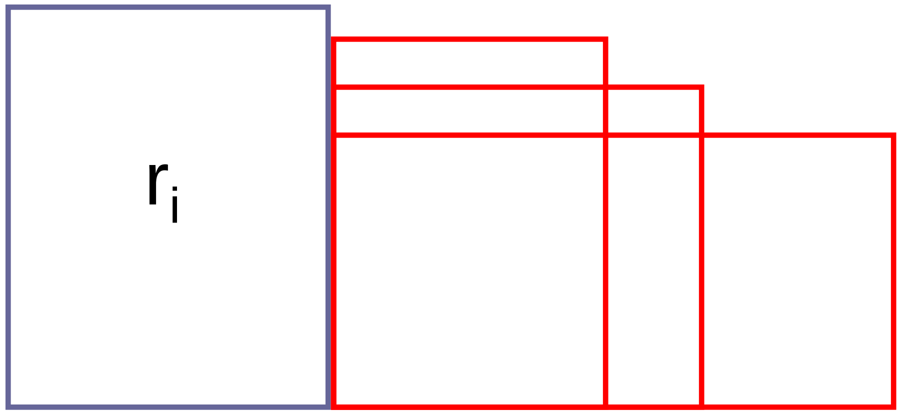
- Why do we combine r_i with realizations r' of q with heights between $h(r_i)$ and $h(r_{i-1})$?



If $h(r') \geq h(r_{i-1})$, the combination of r_i and r' is inferior to that of r_{i-1} and r'.

Compute IRLs at Internal Nodes

Why do we combine r_i with the realization of q with the highest height less than $h(r_i)$?

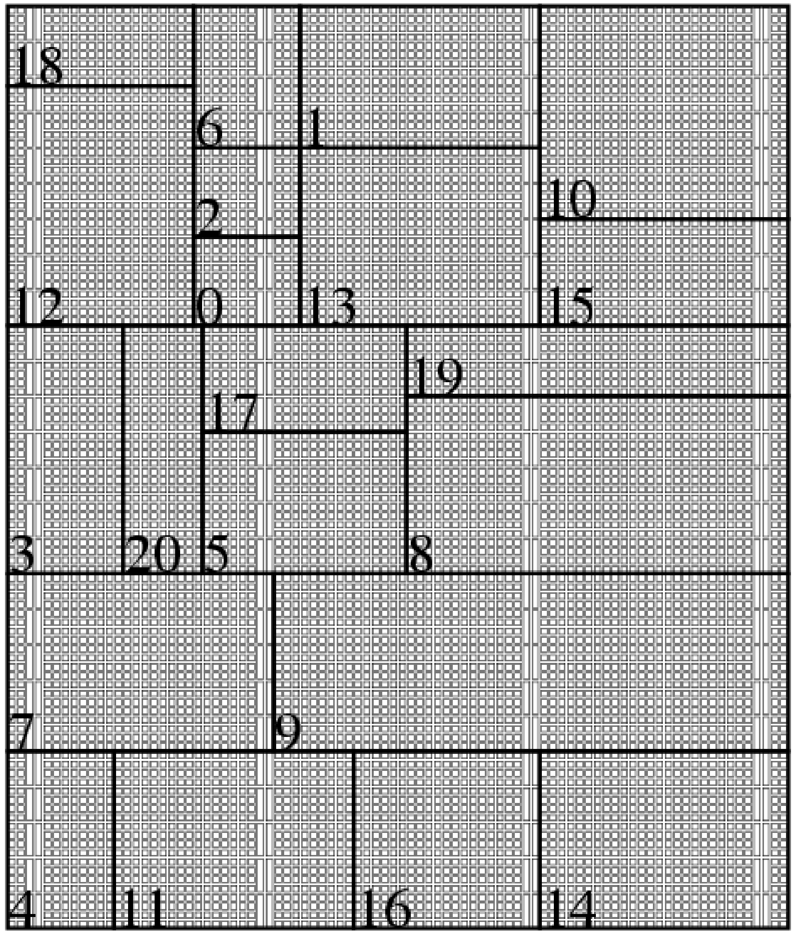


Time Complexity

- The complexity of computing IRL for one (x,y) for an internal node is O(l log l)
    - l is the minimum of the width and height of the chip.
- The complexity of evaluating a slicing tree is O($ml\,p$ log l)
    - p is the number of points on the pattern.
    - m is the number of modules.

Experimental Result

- Xilinx XC3S5000 FPGA
  - 8320 CLBs
  - 104 RAMs
  - 104 multipliers
  - l = 104
  - p = 352
- The result
  - m = 21
  - 72% CLBs
  - 88% RAMs
  - 86% multipliers
- Runtime : 59s

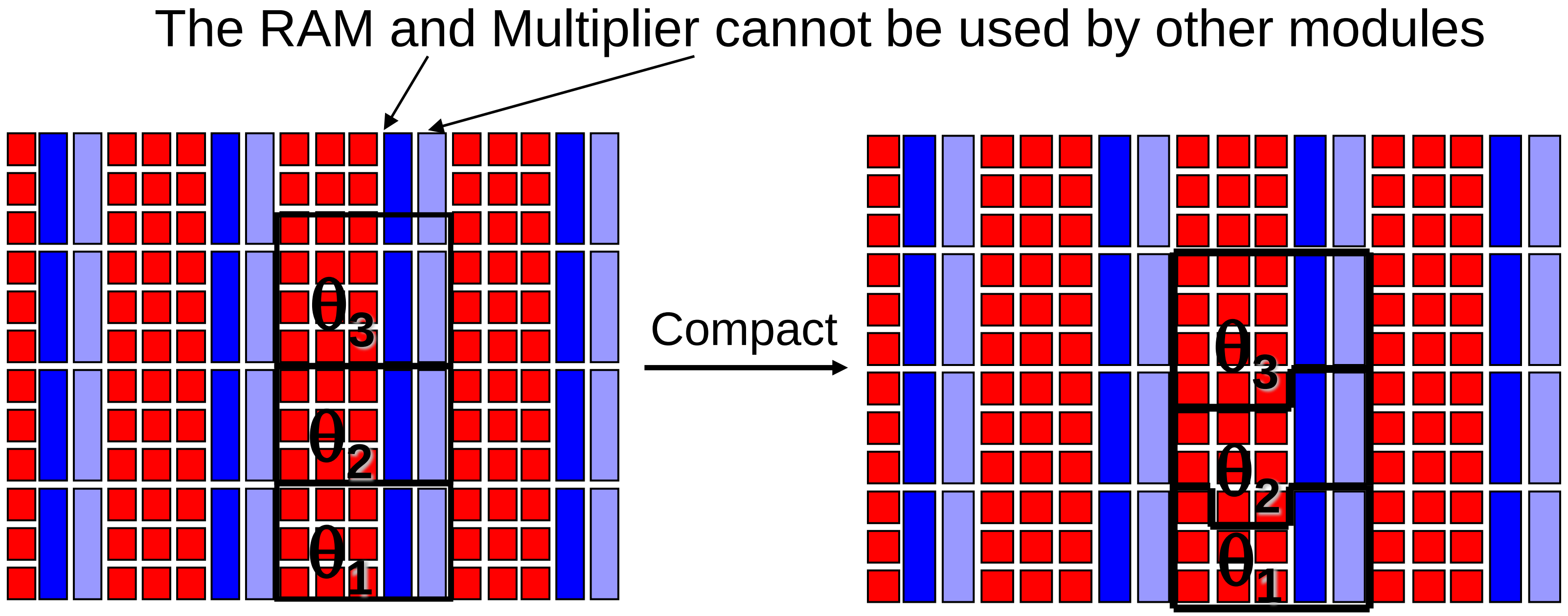


Compaction

- A rectangular realization may contain more resources than required by a module.
- Some problems cannot be solved if we only allow rectangular realizations.
- Compact vertically.

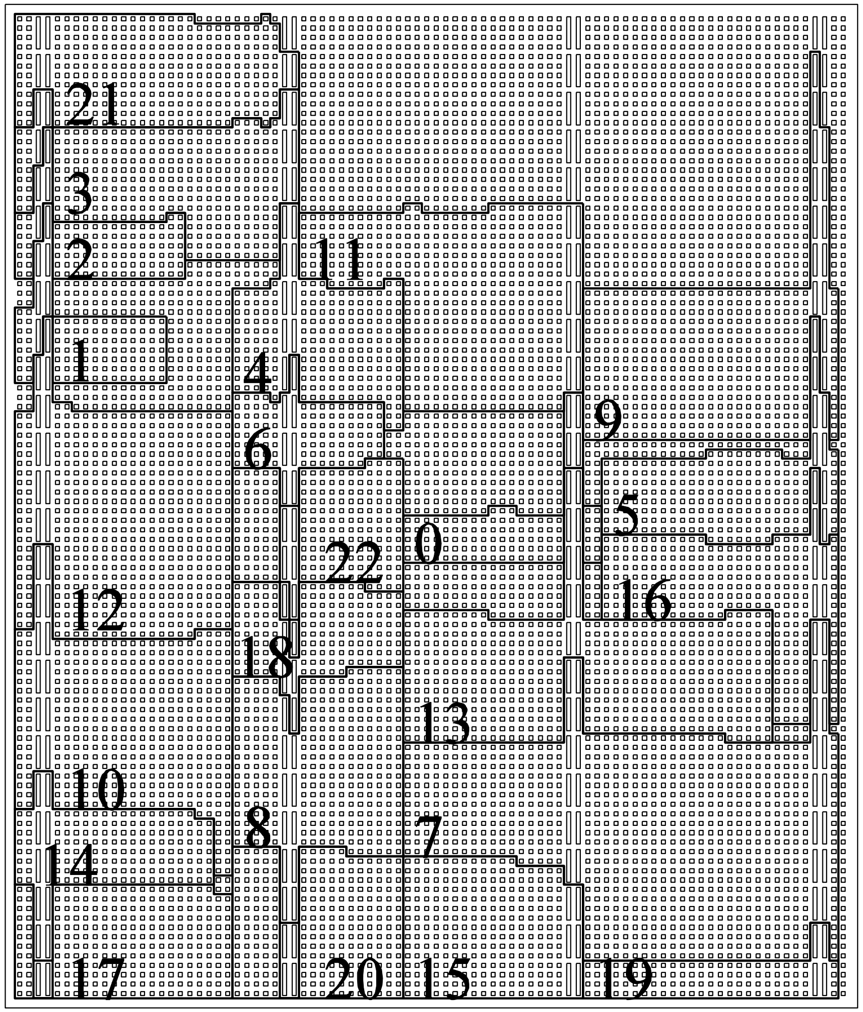
Example

- We have 3 modules θ_1 θ_2 θ_3, and their resource requirements are: <7,1,1>, <8,1,1>, <12,1,1>.



Experimental Result

- Xilinx XC3S5000 FPGA
- The result
  - $m = 23$
  - 72% CLBs
  - 84% RAMs
  - 84% multipliers
- Runtime
  - Slicing only : 27s
  - With compaction : +2s

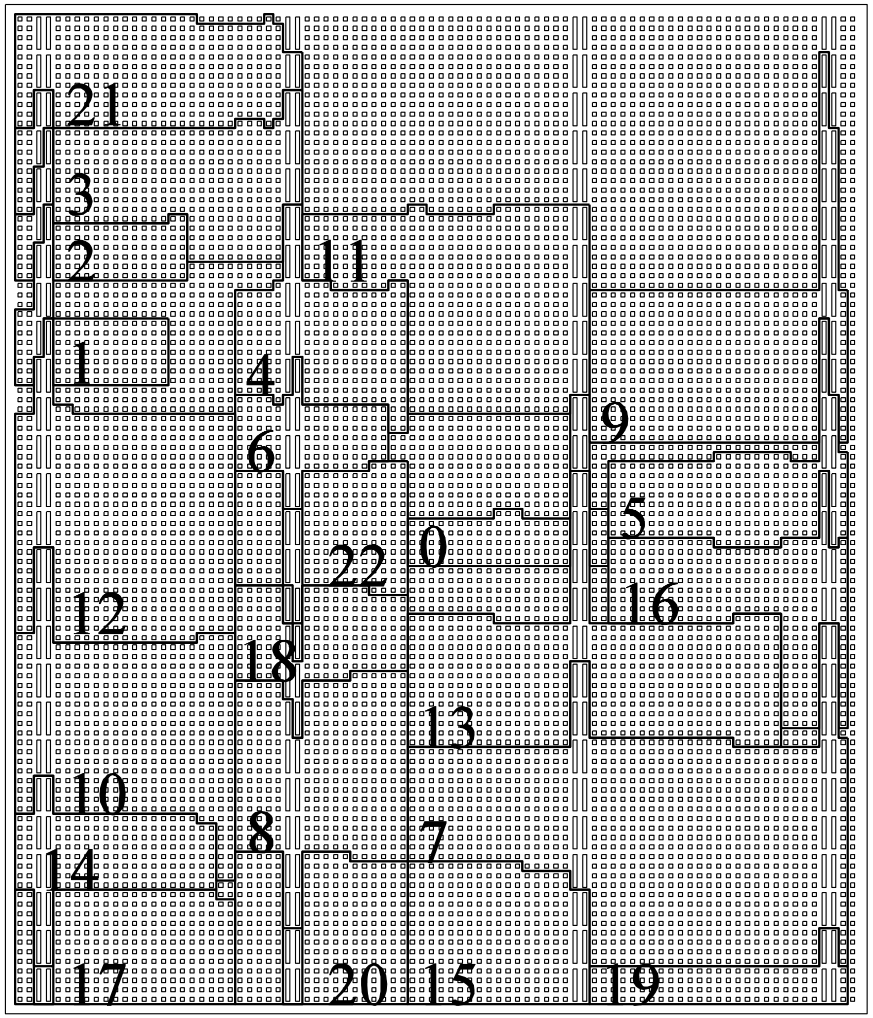


Postprocessing after Compaction

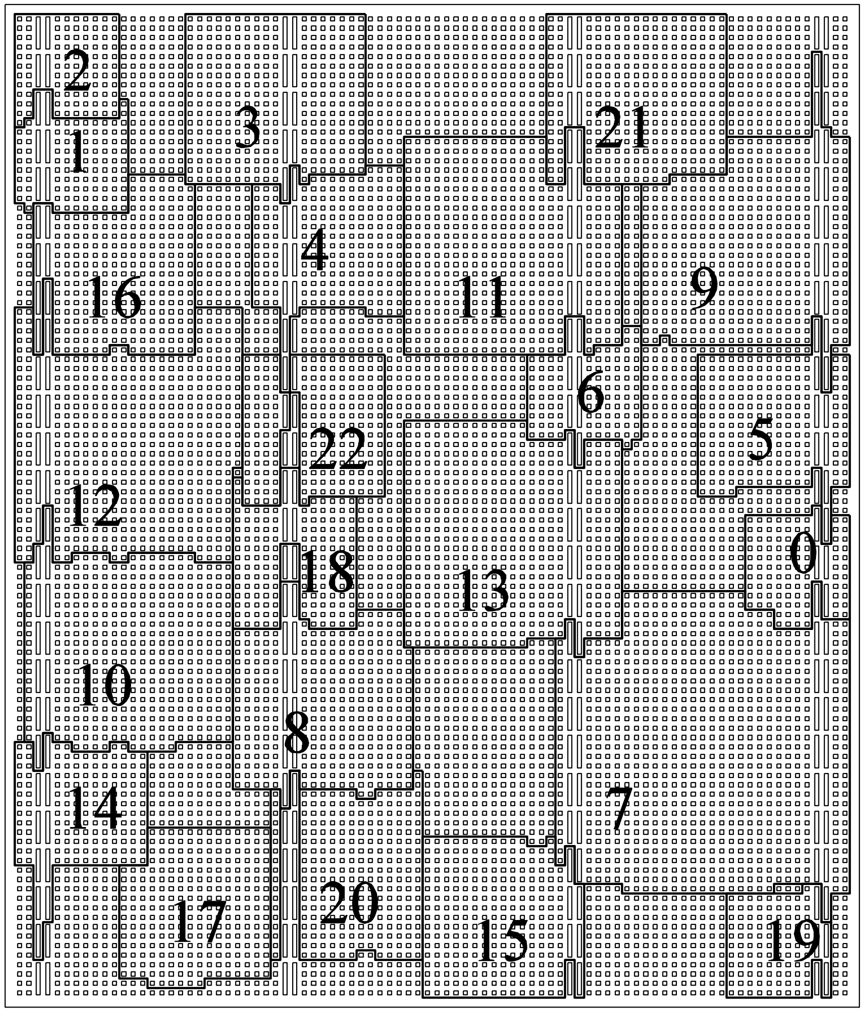
- Problems with compaction:
  - Some modules are placed in undesirable shapes.
  - Large amount of white space on top of the chip.
- Observation:

  A module can be placed freely between two lines.
  - The lower contour line records the placements of modules “below” this module.
  - The upper contour line records the placements of modules “above” this module.
- Process all modules again in reverse order of compaction with the top of the chip as the initial upper contour line.

Experimental Result



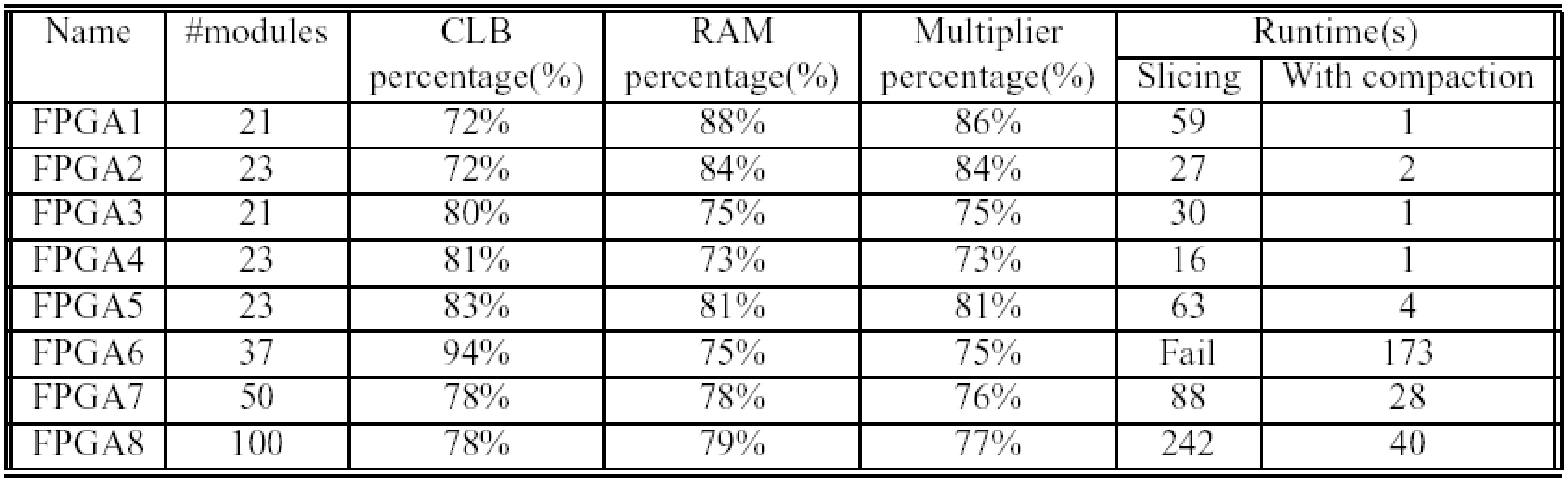
Before



After

Experimental Result

- Xilinx XC3S5000 FPGA



Name	#modules	CLB percentage(%)	RAM percentage(%)	Multiplier percentage(%)	Runtime(s)	
					Slicing	With compaction
FPGA1	21	72%	88%	86%	59	1
FPGA2	23	72%	84%	84%	27	2
FPGA3	21	80%	75%	75%	30	1
FPGA4	23	81%	73%	73%	16	1
FPGA5	23	83%	81%	81%	63	4
FPGA6	37	94%	75%	75%	Fail	173
FPGA7	50	78%	78%	76%	88	28
FPGA8	100	78%	79%	77%	242	40

Conclusions

- FPGA floorplanning algorithm targeted for FPGAs with heterogeneous resources.
- Use simulated annealing to search on slicing floorplans.
- For each slicing floorplan, compute IRLs for all the internal nodes efficiently on a basic pattern.
- Use compaction and postprocessing to solve more problems and to optimize floorplans.

References

- [1] “Floorplan Design for Multi-Million Gate FPGAs”, in *ICCAD’04.*
- [2] “Fast Unified Floorplan Topology Generation and Sizing on Heterogeneous FPGAs”, *TCAD vol.28(5), 2009*.
- [3] “HETRIS: Adaptive Floorplanning for Heterogeneous FPGAs”, in *FPT’15*.

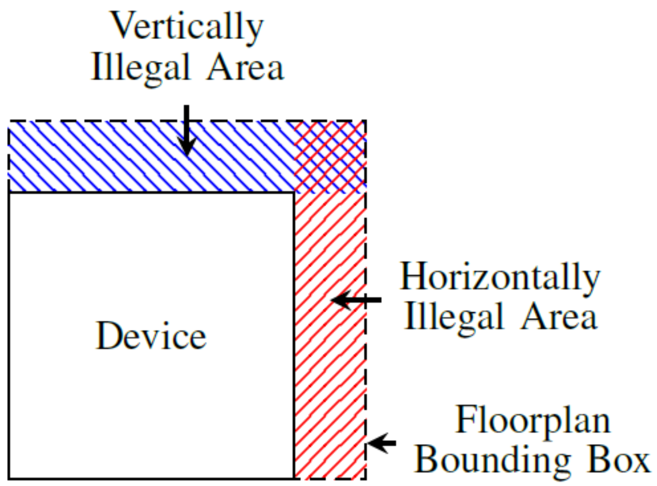
Appendix [3]

- Base cost of a solution

$$\textsc{BaseCost}(S) = A_{fac}\frac{\textsc{Area}(S)}{A_{norm}} + E_{fac}\frac{\textsc{ExtWL}(S)}{E_{norm}} + I_{fac}\frac{\textsc{IntWL}(S)}{I_{norm}}$$

where internal wirelength depends on aspect ratios of modules

- Adaptive annealing to cope with the highly non-uniform architecture to get high quality legal solutions
  - Allow transitioning through illegal parts of the solution space to escape from local minima



- Modified cost function

$$\textsc{Cost}(S) = \textsc{BaseCost}(S) + H_{fac}\frac{\textsc{HorizIll}(S)}{H_{norm}} + V_{fac}\frac{\textsc{VertIll}(S)}{V_{norm}}$$

- H_{fac}(V_{fac}) is increased if most generated solutions are illegal horizontally(vertically)

$$H_{fac} = \begin{cases} H_{fac} \cdot P_{scale}^{2} & \lambda_{legal\_horiz}(T) \leq 0.1\lambda_{legal}^{*} \\ H_{fac} \cdot P_{scale} & 0.1\lambda_{legal}^{*} < \lambda_{legal\_horiz}(T) < \lambda_{legal}^{*} \\ H_{fac} & \lambda_{legal\_horiz}(T) \geq \lambda_{legal}^{*} \end{cases}$$

where λ^{*}_{legal} is the target legal acceptance rate

- Cooling schedule is slowed down when most generated solutions are illegal