DEPARTMENT OF ELECTRONICS AND COMMUNICATION
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

EC-704
VLSI DESIGN AUTOMATION

VLSI-CAD DESIGN FLOW FOR RTL-NETLIST

BY:

SATHVIK BHAT - (181EC141)

UNDER THE GUIDANCE OF:

Dr. M S BHAT

Department of Electronics and Communication

CONTENTS

1. ABSTRACT

The objective of this project is to develop an algorithm to perform VLSI-CAD Design Flow steps on any given RTL-Netlist. The steps involved are partitioning, floorplan, placement and routing respectively. The algorithm is based on an Object Oriented Approach where every netlist acts as an object with a graph structure.

Partitioning of the netlist is carried out using 2-sets of algorithms, namely, Kernighan-Lin (KL) algorithm and Simulated Annealing process. The objective of partitioning is to divide the entire netlist gates into 2-sections. The KL algorithm follows a greedy approach providing nearly the best partitioning. The Simulated annealing process is a continuous iterative process finding the best possible partition.

Floorplanning of the netlist provides an estimate of the length, area, relative position and block size of individual blocks. This data is used by the placement algorithm to define the exact location and orientation for a block. The floorplanning is carried out using the Simulated Annealing Process to get the best floor plan in the form of a skewed-slicing tree. The objective of floorplanning is get a floor plan with minimal area and wiring length.

Placement operation is also carried out using the Simulated Annealing Process. The objective of placement is to place the blocks inside a defined placement core area with no (or minimal) overlap between the blocks and minimal wiring length.

Routing process is defining the positions of wire that connect 2-different blocks. The Maze routing algorithm is applied to get the information of routing wire with minimal wirelength.

2. INTRODUCTION

Any Application specific Integrated Circuit(ASIC) design needs to undergo a series of VLSI CAD optimization to make effective use of chip area, and the routing wire length, to minimize the latency. With the development of software technologies, this process has become more simple, which required expert designers and a lot of manufacturing time earlier.

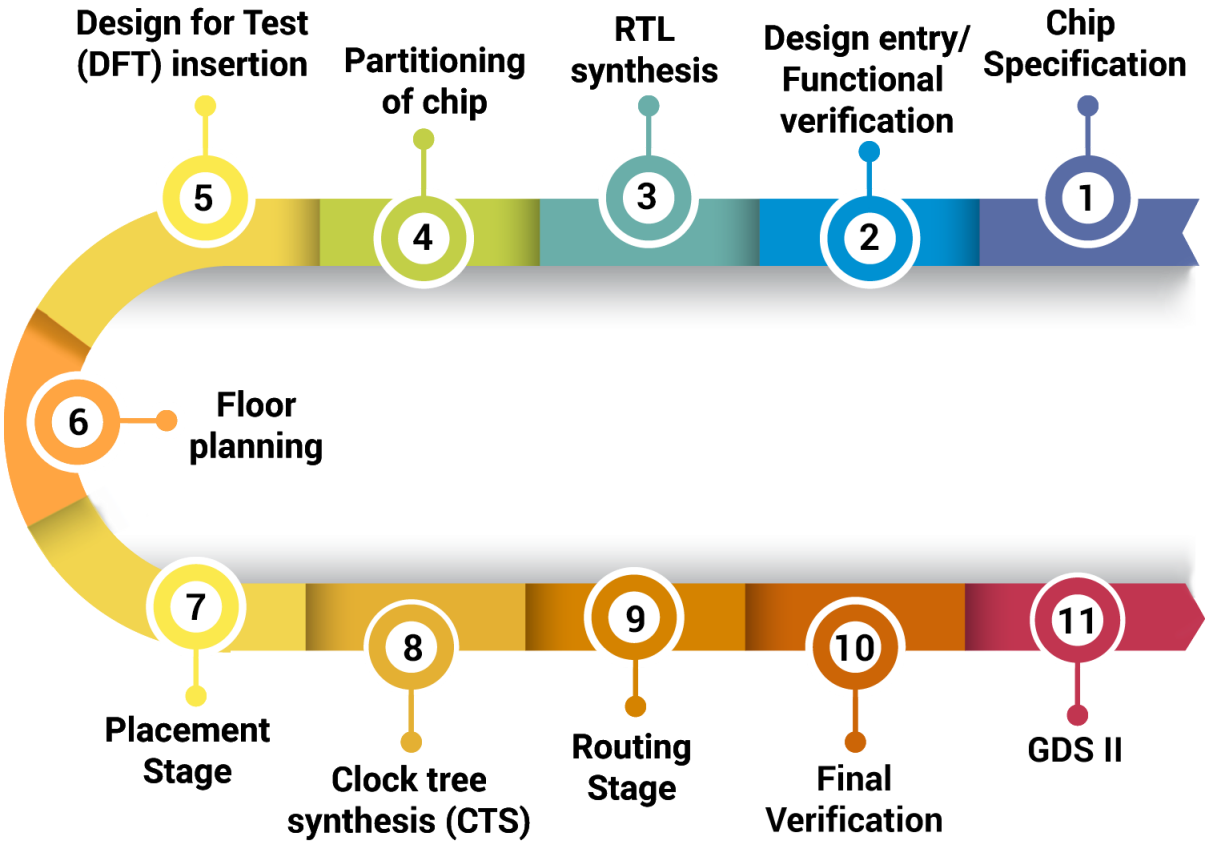
The objective of this project is to develop a basic VLSI Design Automation algorithm which generates a physical layout representation of the steps involved in VLSI Design. The project tries to build an optimized layout and routing using well-developed techniques like simulated annealing.

The size of VLSI designs has increased to systems of hundreds of millions of transistors. The complexity of the circuit has become so high that it is very difficult to design and simulate the whole system without decomposing it into sets of smaller sub-systems. This divide and conquer strategy relies on partitioning to manipulate the whole system into hierarchical tree structure. **Partitioning** is used to tackle the design complexity with a divide and conquer strategy.

In VLSI physical design, **floorplanning** is an essential design step, as it determines the size, shape, and locations of modules in a chip and as such it estimates the total chip area, the interconnects, and delay. Hence, floorplanning makes sure of 3 things, i. every module has been assigned an appropriate area and aspect ratio. ii. Every pin of the module has a connection with other modules or periphery of the chip. iii. Modules are arranged in a way such that it consumes less area on a chip.

Placement is the process of determining the locations of circuit devices on a die surface. It is an important stage in the VLSI design flow, because it affects routabil-ity, performance, heat distribution, and to a lesser extent, power consumption of a design.

Routing is the process of creating physical connections based on logical connectivity. Signal pins are connected by routing metal interconnects. Routed metal paths must meet timing, clock skew, max trans/cap requirements and also physical DRC requirements. In a grid based routing system each metal layer has its own tracks and preferred routing direction which are defined in a unified cell in the standard cell library.



VLSI Design Flow

3. INITIALIZATION

3.1 GRAPH STRUCTURE

A netlist is a text file containing the information on all the gates that are needed to be used in the circuit along with its inputs and outputs. An example of a netlist file is shown below:

```
INPUT(G0)
INPUT(G1)
INPUT(G2)
INPUT(G3)

OUTPUT(G17)

G5 = DFF(G10)
G6 = DFF(G11)
G7 = DFF(G13)

G14 = NOT(G0)
G17 = NOT(G11)

G8 = AND(G14, G6)

G15 = OR(G12, G8)
G16 = OR(G3, G8)

G9 = NAND(G16, G15)

G10 = NOR(G14, G11)
G11 = NOR(G5, G9)
G12 = NOR(G1, G7)
G13 = NOR(G2, G12)
```

‘s27.bench’

To perform operations like positioning, flipping and routing the gates, we need to generate a data-structure which represents all the information contained in the netlist. The data-structure representation of a netlist helps it restructure the way they are arranged.

Graphs are the most commonly used data structures to represent any netlist. The nodes in the graph represent a gate in the netlist. The edges in the graph represent that the output of the source node is connected to

the input of the destination node. An-undirected graph is preferred as data in an edge is the same all-over.

```
In [85]: new_graph = undirected_graph("s27.bench")
         new_graph.print_graph()
```

```
G0 (INPUT)      Adjacent Nodes:  G14(distance=6),
G1 (INPUT)      Adjacent Nodes:  G12(distance=1),
G2 (INPUT)      Adjacent Nodes:  G13(distance=7),
G3 (INPUT)      Adjacent Nodes:  G16(distance=2),
G17 (NOT)       Adjacent Nodes:  G11(distance=8),
G5 (DFF)        Adjacent Nodes:  G10(distance=3), G11(distance=9),
G10 (NOR2)      Adjacent Nodes:  G5(distance=3), G14(distance=2), G11(distance=6),
G6 (DFF)        Adjacent Nodes:  G11(distance=1), G8(distance=9),
G11 (NOR2)      Adjacent Nodes:  G6(distance=1), G17(distance=8), G10(distance=6), G5(distance=9), G9(distance=5),
G7 (DFF)        Adjacent Nodes:  G13(distance=4), G12(distance=7),
G13 (NOR2)      Adjacent Nodes:  G7(distance=4), G2(distance=7), G12(distance=2),
G14 (NOT)       Adjacent Nodes:  G0(distance=6), G8(distance=1), G10(distance=2),
G8 (AND2)       Adjacent Nodes:  G14(distance=1), G6(distance=9), G15(distance=2), G16(distance=8),
G15 (OR2)       Adjacent Nodes:  G12(distance=9), G8(distance=2), G9(distance=7),
G12 (NOR2)      Adjacent Nodes:  G15(distance=9), G1(distance=1), G7(distance=7), G13(distance=2),
G16 (OR2)       Adjacent Nodes:  G3(distance=2), G8(distance=8), G9(distance=6),
G9 (NAND2)      Adjacent Nodes:  G16(distance=6), G15(distance=7), G11(distance=5),
```

A graph representation of the ‘s27.bench’ netlist

NOTE: The edge weights are generated using the random function.

An **Adjacency Matrix** is another way of representing a graph data structure. If an element A[i][j] of a graph is not 0, then nodes ‘i’ and ‘j’ are connected with an edge weight represented by A[i][j].

```
In [86]: new_graph.print_adjacency_matrix()
```

```
Order:
G0 - G1 - G2 - G3 - G17 - G5 - G10 - G6 - G11 - G7 - G13 - G14 - G8 - G15 - G12 - G16 - G9 -
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 3, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 3, 0, 0, 6, 0, 0, 2, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 9, 0, 0, 0, 0]
[0, 0, 0, 0, 8, 9, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 5]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 7, 0, 0]
[0, 0, 7, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 2, 0, 0]
[6, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 0, 1, 0, 2, 0, 8, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 9, 0, 7]
[0, 1, 0, 0, 0, 0, 0, 0, 7, 2, 0, 0, 9, 0, 0, 0, 0]
[0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 6]
[0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 7, 0, 6, 0]
```

Adjacency Matrix of the generated graph

4. PARTITIONING

4.1 PARTITIONING THEORY

As the size of each transistor is decreasing by the day, the amount of transistors on a chip are increasing rapidly. As the amount of transistors on the chip increases, it becomes difficult to optimize area and routing considering all the modules present on the chip.

Hence, the netlist of the circuit is divided into smaller netlists so that each of the sub-netlists can be designed and developed individually and later integrated into a single chip. This process of dividing the larger netlist into smaller sub-netlists is called partitioning.

The objective of partitioning is to reduce the cut-size between any 2 sub-netlists(modules). Cut-size denotes the edges that connect between 2 sub-modules. On minimizing the cut-size, we are grouping the set of modules such that a minimal number of edges connect between sub-modules.

Objective: To generate partitions among netlist modules that minimizes the cut-size(interconnections between sub-modules).

4.2 K-L Algorithm

Kernighan-Lin Algorithm proposes a heuristics approach to generate 2 partitions from the given netlist gates. It is an iterative approach, which is carried out until the cut-size between the partitions keeps decreasing.

The objective of the KL Algorithm is to partition the graph G(V, E) into disjoint subsets A and B such that the cut-size(sum of the edge weights from A to B) is minimum. Reducing the cut-size reduces the overall cost of the system and increases the independence of the subset.

The procedure is as follows:

1. A dummy node is added to the graph if the number of nodes in the graph is an odd number.
2. The algorithm initializes 2 subsets by randomly dividing the nodes of the graph. The difference cost(difference between internal and external costs) is calculated for allnodes.
3. Then the algorithm finds out the reduction in costs by swapping any 2 pairs of nodes. The nodes that produce maximum reduction or minimal increase in costs are locked.This process is iterated until all nodes are locked.
4. The nodes are swapped until the overall costs are reduced, else the iteration is stopped.

```
In [8]: new_graph.kl_partition()

Initial Partitions:
First Partition:         G0 G1 G2 G3 G17 G5 G10 G6 dummy
Second Partition:        G11 G7 G13 G14 G8 G15 G12 G16 G9
D values for First partition:  [6, 6, 8, 5, 1, 7, 10, 7, 0]
D values for Second Partition:  [13, -4, -3, 8, -11, -20, -12, -1, -14]

Initial Cut size =  52
```

Initial Partition on ‘s27.bench’ netlist file

```
Final Partitions:
First Partition:         G0 G14 G11 G9 G17 G5 G10 G6 dummy
Second Partition:        G2 G7 G13 G1 G8 G15 G12 G16 G3
Final Cut Size =  12
```

Final Partition on ‘s27.bench’ netlist on applying KL Algorithm

On applying KL-Algorithm, we can observe a decrease in cut-size from 52 to 12 .

4.2 Partition Using Simulated Annealing

Simulated Annealing is a probabilistic search technique that maps the chemical annealing process to an algorithmic domain. The annealing process initiates with a high temperature and drastically cooled down to form crystals. Identically, an entity from the solution space of the partitioning problem is taken as an initial state and some amount of iterations are carried through.

At every temperature, the process continues to reach equilibrium at that very temperature. In the course of the process, new solutions are generated and checked if a better solution has arrived or not, if the better solution is found, it is kept, otherwise a selection procedure takes place.

The simulated annealing process for partitioning is applied as follows:

1. The algorithm initializes 2 subsets by randomly dividing the nodes of the graph. The cost is calculated based on the cut size for the partition and the balance between partition sizes.
2. Cost = cut_size + lambda(size(A) - size(B))
3. The algorithm then picks a random node and moves it to the other partition. The decrease or increase in cost is calculated.If the cost is decreased, the move is surely accepted,if not, it is accepted based on the value of increase in cost and current temperature (represents time). As the temperature increases the probability of picking an uphill move decreases.
4. The above process is continued either until there are 10 continuous uphill moves or until an average of 10 moves is performed on each node.
5. After this the temperature is further reduced by factor ‘r’, and step2 and step3 are followed again.
6. Even after 5 successive changes in temperature, if the bestcost doesn’t decrease further, the algorithm is stopped.

```
In [9]: new_graph.simulated_annealing_partition(init_temp = 20, r = 0.9, cost_lambda = 1)
```

Simulated Annealing Process:

```
First Partition:          G0 G1 G2 G3 G17 G5 G10 G6
Second Partition:         G11 G7 G13 G14 G8 G15 G12 G16 G9
Initial Cost 53


Current Temperature 20

Iteration:  1
Selected Node: G5       Transfer Cost 48
Move Accepted
First Partition:          G0 G1 G2 G3 G17 G10 G6
Second Partition:         G11 G7 G13 G14 G8 G15 G12 G16 G9 G5
Cost:  48

Iteration:  2
Selected Node: G6       Transfer Cost 43
Move Accepted
First Partition:          G0 G1 G2 G3 G17 G10
Second Partition:         G11 G7 G13 G14 G8 G15 G12 G16 G9 G5 G6
Cost:  43
```

The process is carried out until the final temperature is reached or if there is no improvement on the cost of partition at successive temperatures.

```
Best First Partition:
         G12 G2 G17 G13 G1 G7
Best Second Partition:
         G11 G10 G16 G8 G6 G9 G14 G0 G15 G3 G5
Cost:  15
Cut Size:  10
Balance Size for First Partition:  0.35294117647058826
```

Final Partitions after simulated annealing

We can observe there that simulated annealing process provides better cut-size compared to KL Algorithm. This is mainly due to the fact that the simulated annealing process accepts partitions of various sizes, and only a penalty is added.

5. FLOORPLANNING

5.1 FLOORPLAN THEORY:

Floorplanning is an essential design step for hierarchical, building-module design methodology. Floorplanning provides early feedback that evaluates archi-tectural decisions, estimates chip areas, and estimates delay and congestion caused by wiring.

We can classify floorplans into two categories for discussions: (i)slicing floorplan and (ii)non-slicing floorplan.

Slicing tree is a binary tree with modules at the leaves and cut types at the internal nodes. There are two cut types, H and V. The H cut divides the floorplan horizontally, and the left(right) child represents the bottom (top) sub-floor plan. Similarly, the V cut divides the floorplan vertically, and the left (right) child represents the left(right) sub-floor plan.

Since a slicing tree can represent more than one floorplan, it is desirable to prune such redundancies to facilitate floor plan design. As such, we refer to a slicing tree as a **skewed slicing tree** if it does not contain a node of the same cut type as its right child.

5.2 SIMULATED ANNEALING PROCESS

Simulated Annealing process is used to generate the optimal skewed-slicing tree. Simulated annealing-based floorplanning relies on the representation of the geometric relationship among modules. The floorplan is then generated using the slicing tree.

The simulated annealing process for floorplan is as follows:

1. The algorithm initializes a skewed-slicing tree where all the cut-lines are vertical and all the blocks are placed in a horizontal

line. The cost is calculated based on the area used by the floorplan area and the half-perimeter wire length between wall blocks.

2. At each iteration at each temperature, one of the following moves are performed:
    a. M1: Swap 2 adjacent operands
    b. M2: Invert a series of consecutive operators
    c. M3: Swap adjacent operator and operand, given it doesn't violate ballooning property.
3. Cost is calculated for each of the following swap and swap is legalised only if a criterion is met.
4. The slicing tree with the best cost is generated once the final temperature of the simulated annealing is reached.

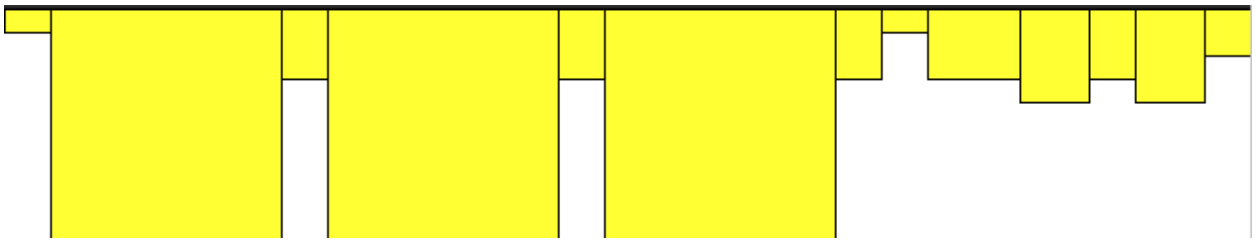
Simulated Annealing Process:

```
In [13]: new_graph.floorplan_simulated_annealing(GATE_SIZES, T0 = 270, Tf = 10, r = 0.85, k=10, lambda_cost = 0.5)

         Slicing Tree Traversal:
         G17 [2, 1] -> G5 [10, 10] -> V [12, 10] -> G10 [2, 3] -> V [14, 10] -> G6 [10, 10] -> V [24, 10] -> G11 [2, 3] ->
         V [26, 10] -> G7 [10, 10] -> V [36, 10] -> G13 [2, 3] -> V [38, 10] -> G14 [2, 1] -> V [40, 10] -> G8 [4, 3] -> V
         [44, 10] -> G15 [3, 4] -> V [47, 10] -> G12 [2, 3] -> V [49, 10] -> G16 [3, 4] -> V [52, 10] -> G9 [2, 2] -> V [5
         4, 10] ->
         Initial Cost =  2141.75
         Slicing Tree Expression:
         G17-G5-V-G10-V-G6-V-G11-V-G7-V-G13-V-G14-V-G8-V-G15-V-G12-V-G16-V-G9-V-


         Simulated Annealing Process




         Current Temperature =  270
         Current Expression:
         G17-G5-V-G10-V-G6-V-G11-V-G7-V-G13-V-G14-V-G8-V-G15-V-G12-V-G16-V-G9-V-


         Iteration:  1

         M3 Move
         M3 Not possible
         Transfer Slicing Tree
         G17 G5 V G10 V G6 V G11 V G7 V G13 V G14 V G8 V G15 V G12 V G16 V G9 V
         Delta Cost =  0.0
         Uphill Move Accepted
```

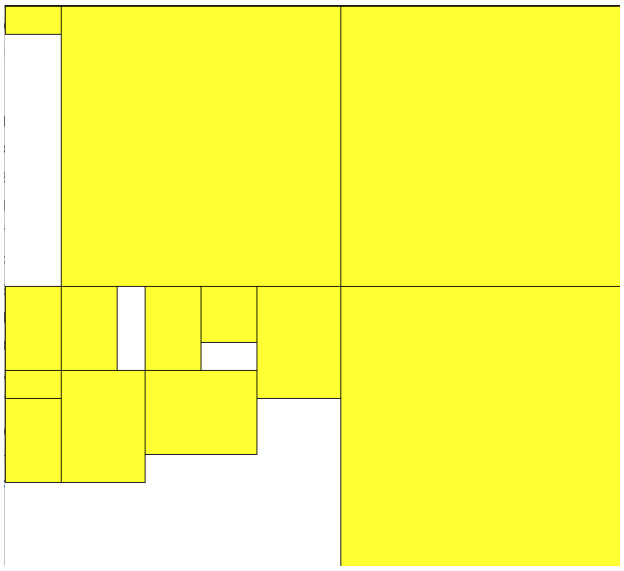
Initiating simulated annealing process

```
Best Slicing Tree
G17 [2, 1] -- > G5 [10, 10] -- > V [12, 10] -- > G6 [10, 10] -- > V [22, 10] -- > G12 [2, 3] -- > G14 [2, 1] -- >
H [2, 4] -- > G13 [2, 3] -- > H [2, 7] -- > G10 [2, 3] -- > G16 [3, 4] -- > H [3, 7] -- > V [5, 7] -- > G11 [2, 3]
-- > G9 [2, 2] -- > V [4, 3] -- > G8 [4, 3] -- > H [4, 6] -- > V [9, 7] -- > G15 [3, 4] -- > V [12, 7] -- > G7 [1
0, 10] -- > V [22, 10] -- > H [22, 20] -- >
Best cost =  753.25
```

Best floorplan slicing tree postorder traversal



Initial Floorplan



Final floorplan after simulated annealing process

6. PLACEMENT

6.1 PLACEMENT THEORY

Placement is the process of determining the locations of circuit devices on a die surface. The locations of the circuit modules in the netlist are determined in the process. The main objective of placement is to confine the floorplan of the netlist to the given chip area, by removing all the overlap between modules.

6.2 SIMULATED ANNEALING PROCESS

Simulated annealing is an iterative heuristic for solving combinatorial optimization problems. The basic idea of simulated annealing is to search for a configuration with low cost by iteratively moving from the current configuration to a neighbor configuration. The probabilistic move helps the search procedure to get out of a local minimum.

The simulated annealing process for placement is as follows:

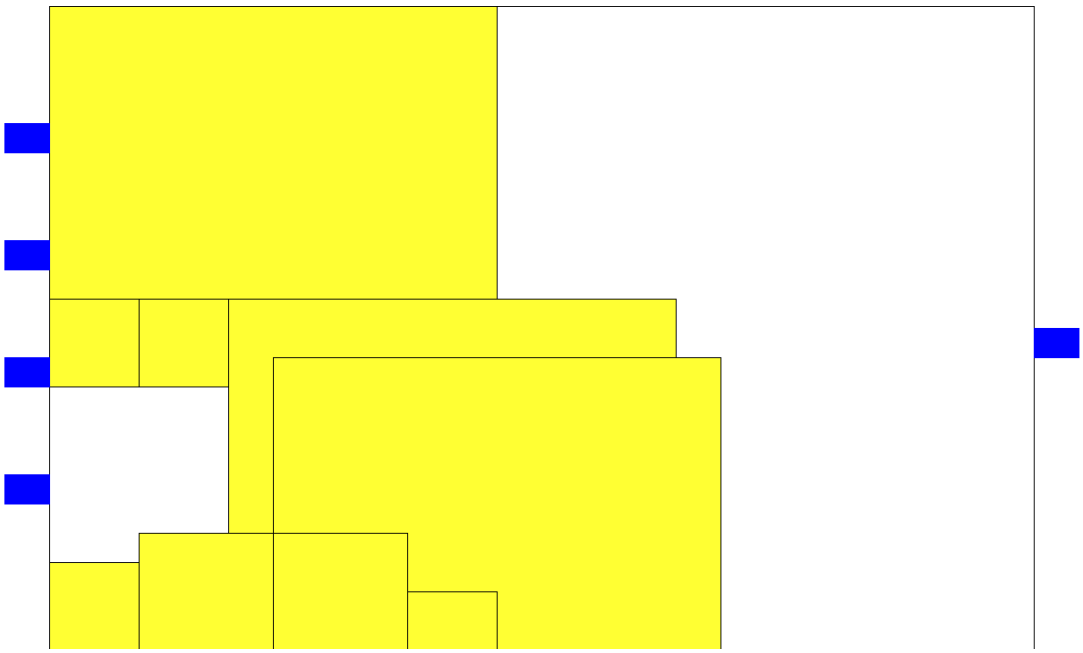
1. The core area and dimensions are calculated based on the placement density, aspect ratio and area of all the modules.
2. An initial placement is performed based on the floorplan coordinates. The entire layout is confined within the core area, which causes overlaps between the modules.
3. The cost of placement is chosen is a combination of wiring cost and overlap area. The best lambda_overlap_cost is chosen based on user's requirement for overlaps.
4. During the annealing process, the program picks up a gate and displaces it to another location to find the change in cost. The new placement is chosen based on the probability approach of simulated annealing.
5. The process stops on reaching the final temperature and the coordinates of the best placement are frozen as module locations.

```
In [16]: new_graph.placement_simulated_annealing(GATE_SIZES, target_density = 0.7, aspect_ratio = 1, routing_per_cell=4,
                                                T0 = 273, Tf = 10, k=10, r=0.95, lambda_overlap_cost = 100)

         Initial Core Area:  22 X 22
         Area Occupied:  368.0
         Overlap Area:  112.0
         Cost:  12501.5
         Placement Density =  0.7603305785123967
         No of Gates: 13
```

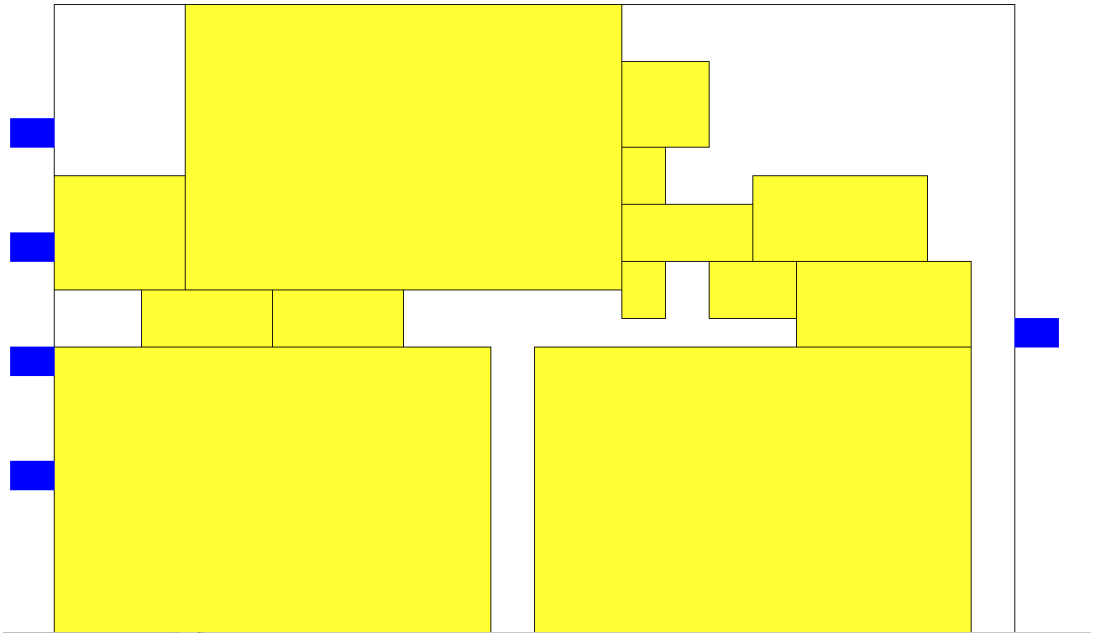
Initial Placement Properties

```
Best Placement
Placement Area:  368.0
Placement Density:  0.7603305785123967
Overlap Area:  0.0
Cost:  1323.5
```

Best placement properties



Initial Placement



Final Placement after simulated annealing process

7. ROUTING

7.1 ROUTING THEORY

The routing process determines the precise paths for nets on the chip layout to interconnect the pins on the circuit blocks or pads at the chip boundary. These precise paths of nets must satisfy the design rules provided by chip foundries to ensure that the designs can be correctly manufactured.

The most important objective of routing is to complete all the required connections(i.e., to achieve 100% routability). Since one net cannot cross another, it is generally tough to achieve 100% routability of all nets on a single layer. Also the quality of placement defines the routability of the chip.

7.2 MAZE ROUTING

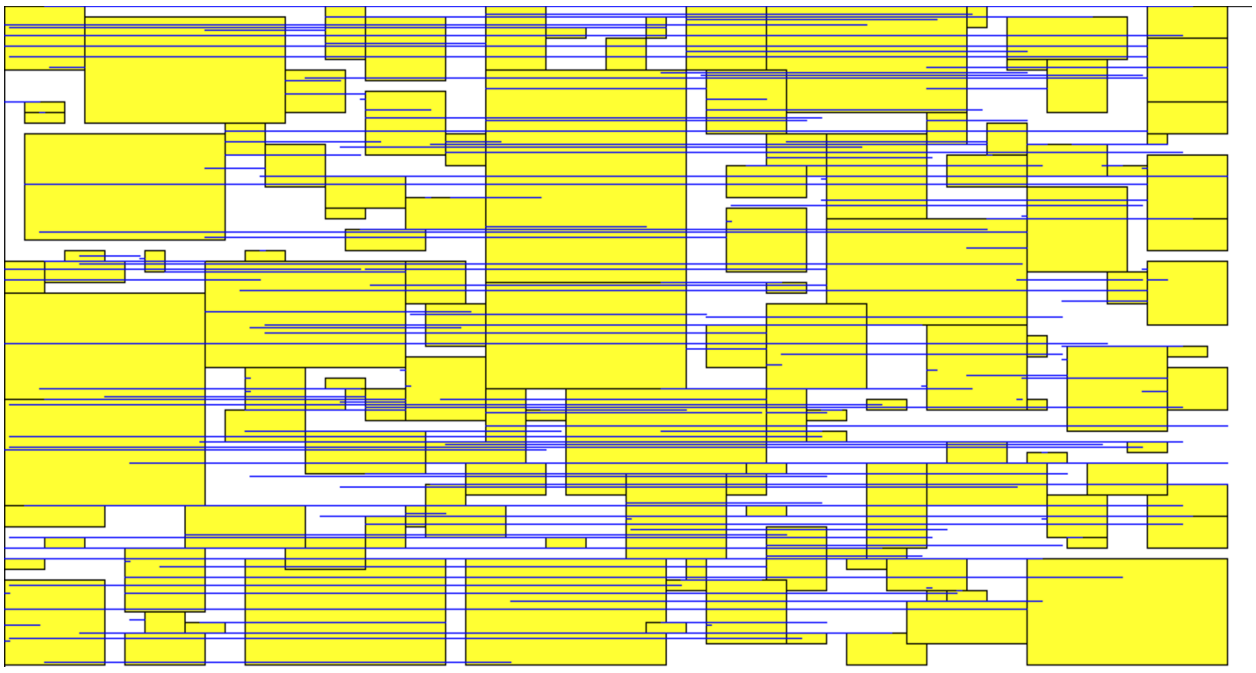
The maze-routing algorithm is a low overhead method to find the way between any two locations of the maze. The algorithm is initially proposed for chip multiprocessors (CMPs) domain and guarantees to work for any grid-based maze. In addition to finding paths between two locations of the grid (maze), the algorithm can detect when there is no path between the source and destination.

The most widely used algorithm for finding a path between two points is the **Maze routing** algorithm(also calledLee's algorithm), which is based on the breadth-first-search(BFS) technique. Maze routing adopts a two-phase approach of expand followed by retracing.

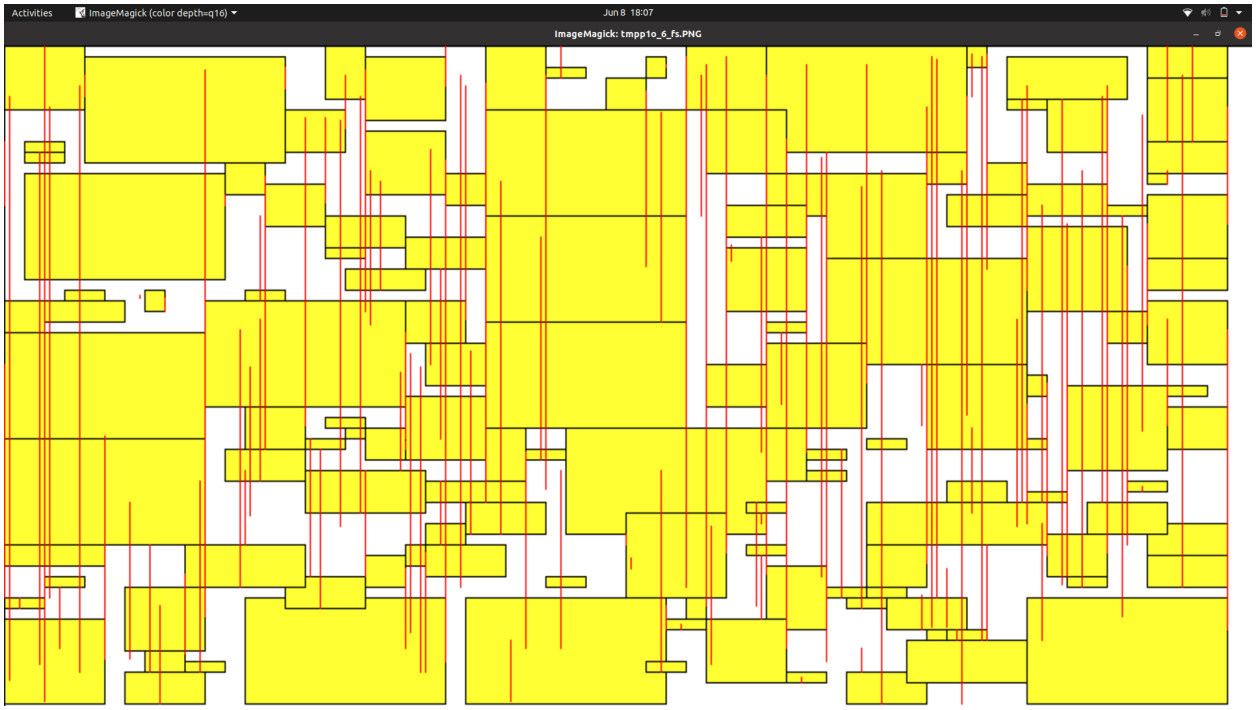
The expand phase works in the "wave propagation" manner. Starting from the source node S, the adjacent grid cells are progressively labeled one by one according to the distance of the "wavefront" from source S until the target node T is reached.

The maze routing process:

1. A routing core is initialized which denotes the exact position of routing points.
2. The algorithm uses a 2-layer routing approach where the first layer routes only horizontal routes while the second layer routes the vertical routes. The algorithm design is based on routing nets from a source block's periphery to the target block's periphery.
3. Once the core is ready, each net is chosen, and starting and ending points are initialized. A wavefront is generated containing only the source position.
4. The wavefront is expanded until the target is reached. Once the target is reached, the source node is traced based on direction stored at each location.
5. There might be cases where some nets are not routed due to non-reachability.



Horizontal Routing Layer



Vertical Routing Layer

Final Routing for ‘s298.bench’

8. CONCLUSION

The detailed steps of VLSI physical design are programmed to be performed using an automated algorithm. The project provides acceptable results for Partitioning, Floorplan and Placement. The results of routing can be improved by using multi-layered routing and by fixing the IO-Pads at the periphery of each gate.