Implementation of Fiduccia Mattheyses Partitioning Algorithm for gate level designs

1.Problem Statement:

To implement, and experiment the Fiduccia-Mattheyses partitioning algorithm implemented for gate-level designs. The goal of the partitioning project is to minimize the cut set size, while meeting given area constraints fixed for the partitions.

2. Introduction:

In VLSI design cycle, physical design is a major step. In this step, circuit representation is converted into geometrical representation called as layouts. Partition has an important role in physical design, and has a significant influence on the speed of the design process. The goal of the partition is to divide the circuit design in to several smaller sub components with minimum number of interconnection between them, adhering to constraints given keeping original functionality intact. The size of the vlsi circuit is growing drastically every day. Hence there is a need to optimize the automation of partition process. In this project, we have implemented Fiduccia-Mattheyses Algorithm of partitioning the circuit.

2.1 Fiduccia-Mattheyses Algorithm:

Fiduccia-Mattheyses (FM) algorithm is a hypergraph partitioning algorithm in which time-complexity is quadratic in nature [3]. It is an iterative algorithm [1] and It allows unbalanced partitions [3].

The Fiduccia-Mattheyses (FM) algorithm runs by priority, processing moves through gain. A move shifts the cell from one partition to another. When the cell has moved, the gain of connected components is also updated. In every running pass, each vertex is moved exactly once.

At the beginning, all vertices are locked, and every possible move is assigned a gain. Iteratively the move with the highest gain is chosen, and after moving the vertex is locked. Before every vertex is locked, the FM algorithm is repeated to perform selection, and execution of the best-gain move, followed by gain update. After that, the best solution seen is adopted as the starting solution of the next pass. When there is no more remaining improvement, passes are no longer applied.

3.Flow Chart of Fiduccia-Mattheyses partitioning algorithm

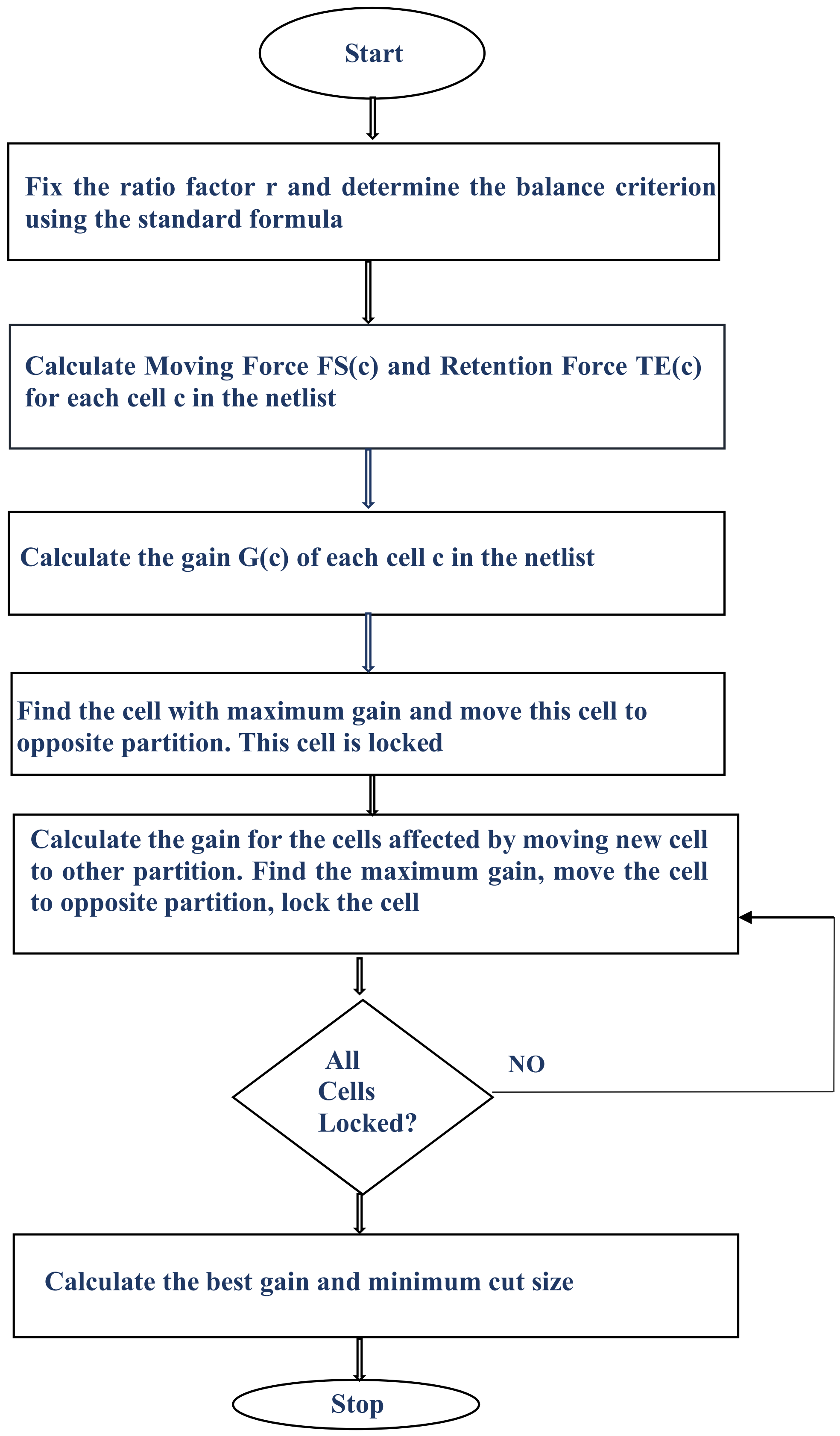

Start
Fix the ratio factor r and determine the balance criterion using the standard formula
Calculate Moving Force FS(c) and Retention Force TE(c) for each cell c in the netlist
Calculate the gain G(c) of each cell c in the netlist
Find the cell with maximum gain and move this cell to opposite partition. This cell is locked
Calculate the gain for the cells affected by moving new cell to other partition. Find the maximum gain, move the cell to opposite partition, lock the cell
All Cells Locked?
NO
Calculate the best gain and minimum cut size
Stop

Fig 1. Flow Chart

4. Implementation Details:

Step 1: Created a Cell class. Each cell class holds following information as public data members. Cell Id, Partition, locked status, Area and a linked list to store all nets in which cell is present. Initially all cells are unlocked.

Step 2: Read .net file and convert it in to .hgr file using perl script. The .hgr file contains the information regarding all netlists.

Step 3: Read the .hgr file, and create a Hash map to add the Net list mapped using an integer Id.

Step 4: Traverse the netlist, and update each cell object with information in for each nets that the selected cell was part of. This helps to reduce the number of traversal required by eliminating unnecessary traversal to nets in which selected cell is not present.

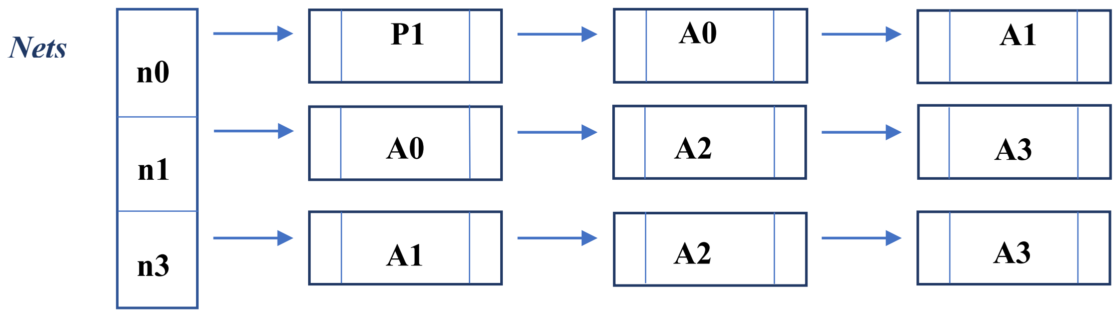


Fig 3. Implementation of Net list in Hash map

Step 5: Read the .are file and get the total area max area of a cell. The area of each cell is updated in the cell objects created earlier. Even the cell ids are taken from .are file and updated in cell object.

Step 6: Created a gain bucket. In which key values are the values of gain. The middle point is zero and bucket keeps updating both in positive and negative sides each time a new gain is calculated. The cells with same gain are mapped to same key value and connected via a vector. Below figure shows the bucket structure created.

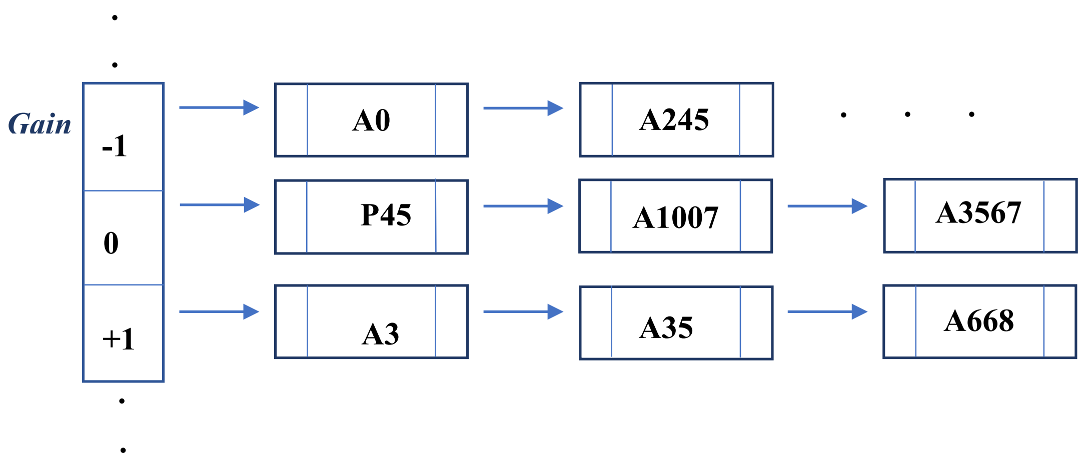
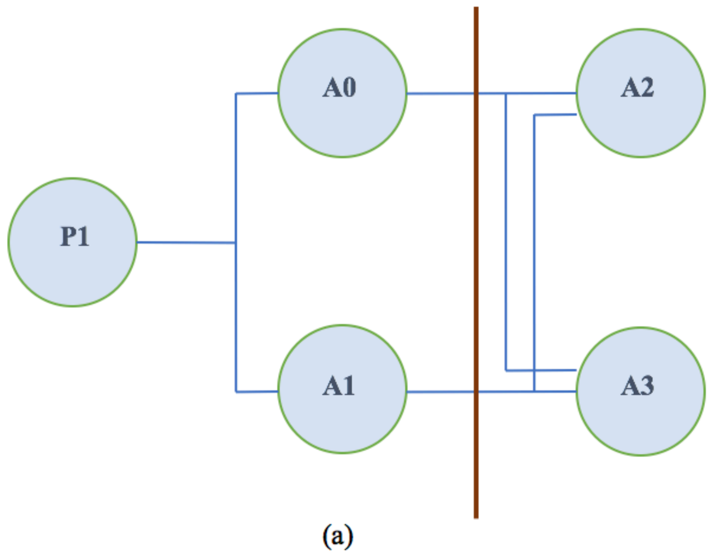


Fig 4. Gain Bucket Structure

In Fig 3. Cells A0, A245 are the cells with same gain -1.and respectively other cells with same gain are also connect via a linked list data structure.

Step 7: Expected area of first partition is calculated using total area, and the fixed the ratio cut r. Then area of each cell is added cumulatively. When the cumulative sum of areas is equal to expected area of the first partition, the index of the cell is noted, and is partitioned at this point. Each cell object is updated with partition details.

Step 8: In first pass consider the first cell and traverse only the nets to which that cell belongs and calculate its FS(x). Traverse again to calculate TE(x) and then calculate gain G(x). Continue the same for all the cells. While calculating FS(x) and TE(x), we have used the concept of hyper graph. So, number of cuts are not over estimated. Below is the figure which shows the hyper graph and normal graph representation details.



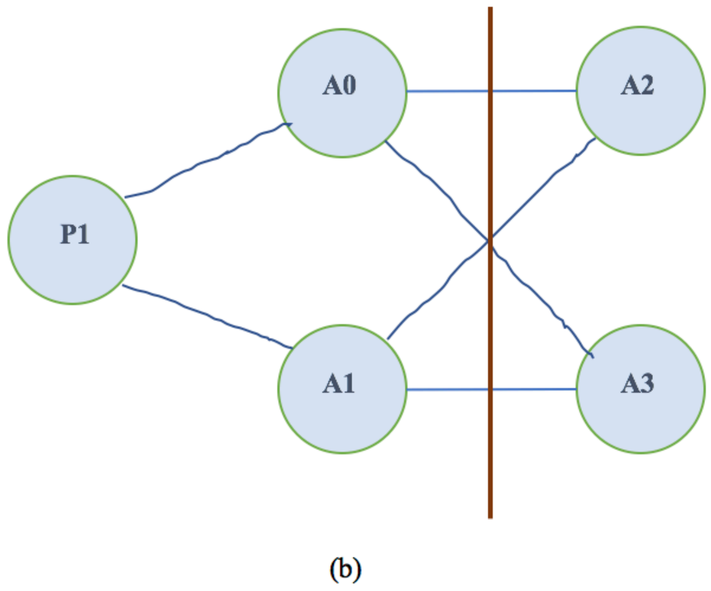


Fig 5. (a) Hyper graph (b)Normal graph

In Fiq 4. Shows the cut size reduction in Hyper graph (cut size =2) compared to normal graph (cut size=4) So, in our implementation we have optimized the way, cut size is calculated, by implementing a hyper graph.

Step 9: Calculate the best gain in the first pass by traversing in the gain bucket list, and select a cell which has maximum gain, while satisfying the area condition if moved to other partitions. This cell is locked, and partition detail is changed as well.

If there is a cell which has maximum gain, but doesn't meet the area balance criteria, then the next cell in the same list with maximum area is considered. This continues until the cell to be locked is found. Once the cell is locked, its removed from bucket structure. Gains of locked cell will not be updated as per the algorithm [1].

Step 10: Repeat step 8 and step 9 until all the cells are locked. Once all cells are locked, the execution terminates.

Step 11: Get the best gain from bucket structure, and get the cut size at that point. This is the reduced cut set and the best gain obtained though this reduced minimal cut size value.

5.Experiment Results:

The results are shown in the table below. We have experimented with several IBM benchmark files as input and with two different r values.

Circuit	**# Modules**	**# Nets**	**Ratio factor [r]**	**Initial cut size**	**Final cut size**	**Max Gain**
ibm01	12752	14111	0.5	8993	3089	5904
ibm01	12752	14111	0.6	8589	3524	5065
ibm02	19601	19584	0.5	13139	6160	6979
ibm02	19601	19584	0.6	13305	5862	7443
ibm03	23136	27401	0.5	17118	7057	10061
ibm04	27507	31970	0.5	20600	7696	12904
ibm05	29347	28446	0.5	19023	17707	1316
ibm06	32498	34826	0.5	19496	9388	10108

System specifications used run the code: 8 Core AMD Ryzen CPU, 16 GB Ram, GTX 1070 (GPU).
IDE Used: Sublime Text Editor.

6.References:

[1] C.M. Fiduccia and R.M. Mattheyses, ―A Linear Time Heuristic for improving Network Partition‖ 19th Design Automation Conference, 1982, pp. 175-181.

[2] B.W Kernighan and S. Lin, ―An Efficient Heuristic procedure for partitioning Graphs‖, The Bell system technical journal, Vol.49 ,Feb. 1970, pp. 291-307.

[3] N. A. Sherwani, ―Algorithms for VLSI Physical Design Automation‖, Springer, 3rd Ed., 1999.

[4] P. Eles, Z. Peng, K. Kuchcinski and A. Doboli, "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search," Design Automation for Embedded Systems,
1997, pp. 5 - 32.

[5] C.J. Alpert, J.H. Huang, and A.B. Kahng, ―Multi-level circuit partitioning‖ IEEE Transaction on Computer-Aided Design of Integrated Circuit and Systems, Vol. 17, No. 8,1998.

[6] G. Karypis, V. Kumar, ―Multi-level K-way hyper graph partitioning‖, VLSI Design Overseas Publishers Association, 2000, pp. 1 – 16.

[7] S. Dutt, ―A New Faster Kernighan- Lin Type Graph Partitioning Algorithm‖, In proc. IEEE Intel Conf. computer Aided Design, 1993, pp. 370-377.

[8] W. E. Donath and A. J. Hoffman, "Algorithms for Partitioning Graphs and Computer Logic Based on Eigenvectors of Connection Matrices," IBM Technical Disclosure Bulletin, 15(3), 1972, pp.938-944.

[9] L.W. Hagen , D.J..Haung ,A.B Kahng, ―On Implementation choices for iterative improvement partitioning algorithms‖, European Design Automation Conference, 1995, pp. 144-149.

[10] L. A. Sanchis, ―Multi-way Network Partitioning‖, IEEE Transaction on Computers, Vol. 38, No. 1, January 198

Appendix:

cell.h - Contains a cell class defined
main.cpp – algorithm implemented
conv.pl – pearl file to conver .net file to .hgr