Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 07, July-2020

Flow Based Logic Design Partitioning

Anupama Patil
Department of Electronics and Communication
B M S College of Engineering
Bengaluru, India

***Abstract*— The major challenges encountered during the design of IC include battery lifetime and energy constraints especially for Internet of Things and mobile applications. Many techniques in the domain of low power and power management have gained widespread attention because of the significance of power efficiency. Due to the increasing complexity of systems on chips (SoCs), multiple power domains on a single integrated circuit (IC) are becoming more common as process nodes are getting smaller. Stacked power domain approach connects voltage domains in series, which effectively improves battery lifetime and efficiency of power delivery. Stacked domain implementation requires partitioning the logic design into two power domains. But this approach requires balancing the currents between power domains. Level shifter insertion between power domain regions, can result in remarkable area penalties. In the proposed paper, a flow based partitioning algorithm is introduced. Considering the solution obtained after initial placement, a flow-based partitioning is applied, which is based on maximum flow and minimum cut algorithm and aware of cell placement in order to partition all cells into two different power domains while achieving balance in cross-domain currents. Algorithm is implemented on MCNC bench mark circuit. Algorithm removes the need for dummy vertices (super source and super sink) to do partitioning.**

Keywords—max flow min cut; Ford Fulkerson, Stacked power domain

I. INTRODUCTION

The partitioning is defined as dividing a chip into smaller blocks/sub blocks. Partitioning is carried out to separate different functional blocks and to ease routing and placement. Partitioning is the initial step in the physical design process. The designer breaks the larger design into various smaller functional modules/blocks and then proceeds with implementation of these smaller modules during RTL design phase. These smaller functional blocks are structurally instantiated or linked in the main module. Main module is called TOP LEVEL module. This type of partitioning is called as Logical Partitioning.

There are many techniques of multi-supply voltage, which allows operation of different blocks with different voltages. Operating at a lower voltage reduces power consumption, but with the compromise in speed. Depending upon the performance requirements, different parts of a chip can use different supply voltages. One of the solutions to reduce power is using multiple supply voltage implementations, since reduction in the voltage has a squared impact on active power consumption. Multiple supply voltage implementation requires insertion of level shifters on signals that cross different voltage levels. Without using level shifters, the signals crossing voltage levels cannot be sampled correctly.

For placement and optimization to implement whole design, the tool must know that no logic can be moved from one power domain to another power domain. The tool must be capable enough to use the correct set of timing libraries for each of the power domain. Lower voltage can result in timing issues and can increase transition time. Logic needs to be upsized or inserted, to overcome timing issues which results in more power consumption.

If there exists a misalignment in battery voltages in comparison to scaled core voltages, this can result in inefficiency which requires saving of power. A stacked power domain design normally connects power domains in series which are connected parallelly in the conventional design for the purpose of aligning the system on chip power domain voltages with battery voltages [1]. The top power domain is placed over the bottom power domain in order to reduce the current to half and double the voltage compared to that of conventional design. This technique performs implicit 2:1 down conversion of external supplies. If the supply voltage of a conventional design i.e. power supply VDD = V and ground supply VSS = 0, the power and ground supplies for the top and bottom domains in the stacked-domain design are (2V, V) and (V, 0), respectively.

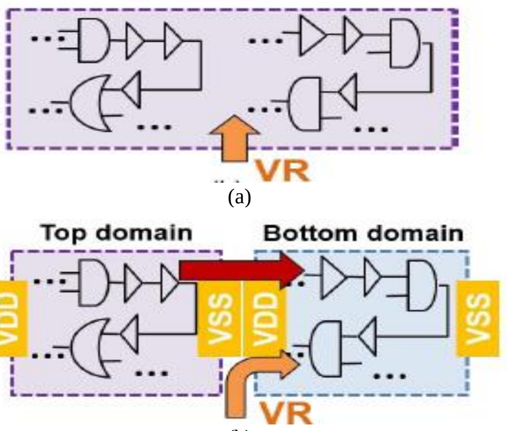


Fig 1. (a) Conventional design. (b) Stacked power-domain design

Fig 1 shows comparison between Conventional design and stacked power domain design. In the above figure, VR represents voltage regulator. The arrow in red symbolizes stacked current. The arrow in orange symbolizes current from the Voltage Regulator.

Battery lifetime (T) and external power (P_{ext}) are inversely proportional to each other. In a stacked-domain,

$$P_{ext} = P_{stk} + P_{VR,in} = P_{stk} + P_{VR,out}/\eta_{VR} \quad (1)$$

Where, P_{ext} is Total input power from external supply

P_{stk} is direct power of stacked power domain from external supply

$P_{VR,in}$ is Voltage regulator input power from external supply

$P_{VR,out}$ is Voltage regulator output power to core

η_{VR} is Voltage regulator power conversion efficiency

In a traditional design, power supply is only provided through Voltage Regulator. Total input power from the external supply for a traditional design ($P`_{ext}$) is calculated as,

$$P`_{ext}=P_{core}/\eta_{VR} \quad (2)$$

Where, P_{core} is Total core power consumption

The Ratio between the battery lifetime of stacked-domain design (T) and the battery lifetime of conventional design (T`) is,

$$T/T` = (2 \cdot I_{stk} + I_{VR}) / (2 \cdot \eta_{VR} \cdot I_{stk} + I_{VR}) \quad (3)$$

Where I_{stk} is Stacked current i.e. the current from top power domain to bottom power domain

I_{VR} is Voltage regulator output current

The lifetime of a battery increases with a smaller I_{VR} because the power efficiency of the Voltage Regulator reduces with smaller currents supply. If there is a proper balance of current between two different domains (i.e., $I_{VR}= 0$) in stacked domain implementation, the design results in improvement in lifetime of a battery over the traditional design.

There are some challenges related to stacked power domain implementation.

i. Level shifters are required for the communication between the power domains; such level shifter can convert extreme levels of signal.
ii. The improvement in power efficiency is dependent directly on balancing of current between two power domains.
iii. Partitioning optimization must take into consideration various operating scenarios, power penalties and area penalties and also impact of level shifters insertion on timing.
iv. Extending stacked domain concept to three or multiple power domains can increase number of level shifters and demand topology of power delivery which can be more complex.

The contribution of the proposed paper is stated below.

1) This paper proposes a flow based partitioning methodology that exploits MAX_FLOW_MIN_CUT algorithm, which can partition a netlist into two parts where each part belongs to a power domain. This becomes basic structure of stacked domain.
2) The optimization flow is validated on an open source benchmark circuit.
3) The proposed optimization facilitates the maximum flow of current between the stacked domains.

II. PREVIOUS WORK

This section describes the previous work done on netlist partitioning and voltage stacking. The power-island generation problem [2] describes region-based Multi Supply Voltage circuits, where each circuit is divided into "voltage islands". Here each voltage island occupies a contiguous physical space and assumes different supply voltages for power domains. But our stacked domain optimization shows balanced current across power domains. Voltage stacking of memory and logic in studied in [3], in this the series connection of corresponding power domains of the logic and memory is described. An ARM Cortex-M0+ and its peripherals are powered from supply of 0 V to VDD, and its 4-kB ROM as well as the 16-kB SRAM are powered from supply VDD to 2VDD satisfying 2:1 ratio of voltages. Stacked-domain technique applied to a complete MCU or microcontroller unit [5] is ad hoc and it has a standard design flow. But it is not applicable to wider range of designs. System designed for energy reuse between two or more stacked CPUs is explained in [7]. To double the battery life, a charge-recycling circuit is used. This architecture consists of multiple CPUs where CPUs are divided as lower load group and upper load group. Electrical charges are shared among the stacked CPUs. Temporary charge storage and reuse is done by the help of tank capacitor. But this design is not useful for complex realistic applications. A smart regulation scheme is described in [8]; this design is relatively complex, which consists of processor cores.

Different types of netlist partitioning have been studied in the previous literature. We highlight three basic netlist partitioning approaches.

A. Move-Based Netlist Partitioning

Kernighan and Lin [6] and Fiduccia and Mattheyses [4] proposed move based algorithms for network partitions. The algorithm in [4] iteratively move or swap a pair vertices but in [6] in any single move only a single vertex will be moved across the cut at reducing net-cut costs. Vertices are weighted. A gain function helps in partitioning a set of vertices into two parts while maintaining minimized number of hyper edge cuts and balanced weights. FM algorithm adds new features to K-L heuristic. Improvements to the FM and KL algorithms have been proposed, such as multilevel k-way hyper graph partitioning algorithm substantially improves the existing KL algorithm[10], multilevel extension of FM based partitioning technique, in which optimal placement and partitioning algorithms are developed based on Gray code-based enumeration and branch-and-bound [9].

B. Clustering Based Netlist Partitioning

A circuit clustering technique for minimizing the delay under general delay model is explained in [12]. Algorithm has two phases, labelling phase and clustering phase. Delay minimization is from PIs (primary input) to POs (primary output) where area constraints are taken into consideration.

*C. **Flow-Based Netlist Partitioning***

An iterative max flow and min cut calculation and clustering process to obtain a balanced bi partitioning solution is explained in [11]. It provides comparatively better quality solution of flow-based partitioning, meeting an objective of obtaining min-cut. Using the flow based partitioning method; stacked domain implementation has been performed in [1]. The implementation flow of [1] is as shown below.

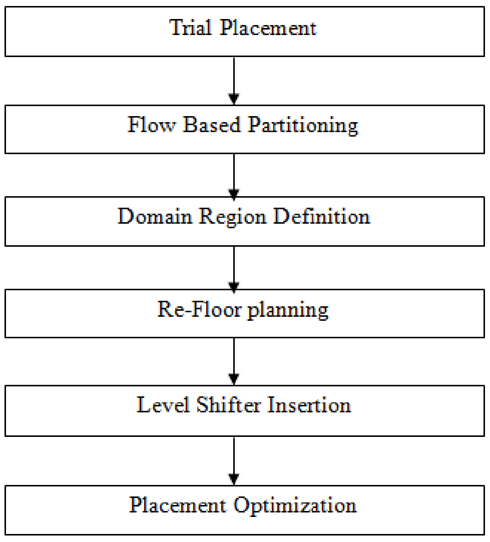


Fig 2. Stacked domain implementation flow

Fig 2 demonstrates stacked domain implementation flow. The stacked-domain implementation explained in [1], the netlist is partitioned before the actual floor-planning stage in order to define the power domain of each block or instance. A trial placement is performed before actual placement because the optimization process during placement stage upsize the cells and inserts buffers. This could result in change in the current profile of each power domain. This results in current imbalance where balance had happened during the partitioning stage. In order to assign cells or instances to corresponding power domains a technique called layout-aware partitioning is performed. A particular layout region is defined for every power domain where each one of these power domain has a continuous region. There is a need for minimizing the boundary length between two power domains which is performed using an optimization technique referred as dynamic programming. Commercial APR tool legalizes the placement of instances within the region defined for each power domain. Then floor plan is updated by shifting the power domains and inserting level shifters between power domains. Level shifter insertion done with the aim of minimizing wire length using a matching-based optimization.

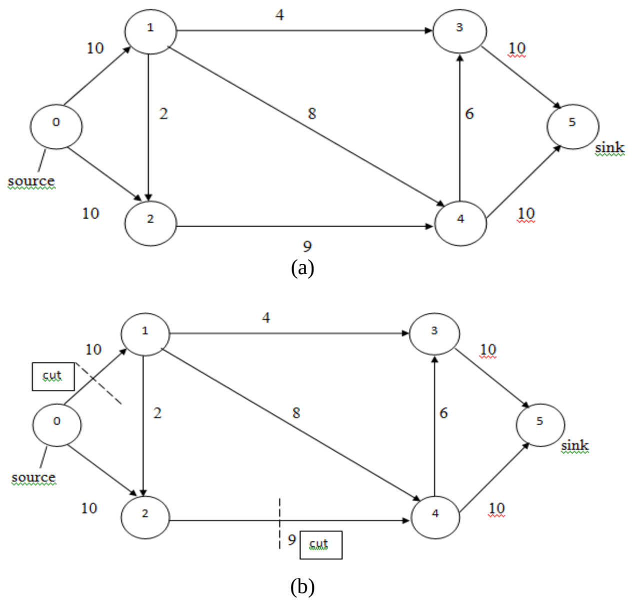
Although partitioning flow will be aware of critical timing paths as a result of trial or initial placement, insertion of level shifter as well as legalization of instant placement incur timing violations. An incremental optimization of the placement is performed including VT-swapping and gate sizing in order to fix timing violations.

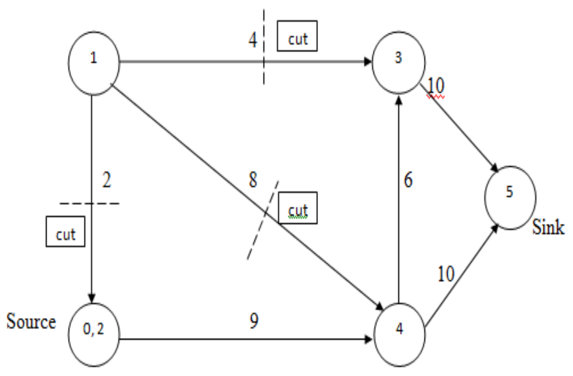
In this paper, the improved version of flow-based netlist partitioning approach is applied which is demonstrated taking into consideration the basics of [11] for partitioning the circuit into two different power domains.

III. PROPOSED WORK

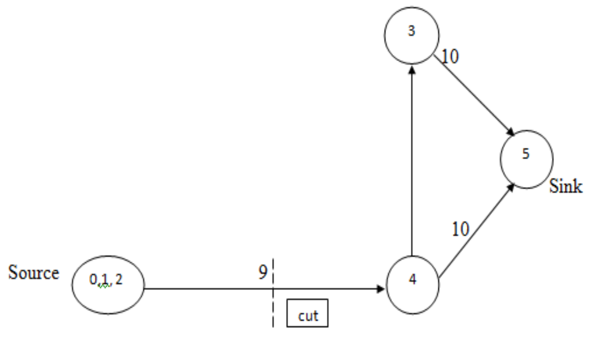
The proposed netlist partitioning algorithm partition the netlist into two parts based on current flow between the cells/instances. Partitioning is done on the basis of iterative max flow, min cut process. The Ford-Fulkerson algorithm is used as a method that resolves the max flow, min cut problem. Flow-based method divides instances into two power domains with the objective of 1) minimizing the number of cuts, 2) balancing the current between two partitions or clusters (where two partitions represent two power domains in stacked domain design).

Max flow min cut theorem, finds solution for partitioning with minimized number of cuts for the given netlist through the method of max flow optimization. But this cannot ensure balancing constraints are met. To meet balancing constraint after every max flow process, need to cluster cut edge node/ vertex with one neighbor vertex, forming a super vertex. This is done to avoid getting same solution for partitioning. Once an updated flow network is obtained, another set of max flow computation and optimization is performed. This approach will perform max flow computation as well as clustering iteratively unless and until the balancing criteria is obtained. In this process, process stops when current iteration flow is greater than previous flow meeting max flow objective and min number of cuts are obtained. Fig. 3 demonstrates basic idea of the flow based partitioning.





(c)



(d)

Fig 3. Flow-based net list partitioning. 0th and 5th nodes are Source and Sink, respectively. Dotted lines denote cuts. Number on edges represents actual flow value which is current flow in this algorithm. (a) Initial graph representing flow network. (b) First iteration with max flow min- cut computation. (c) First clustering operation and second iteration of max flow min- cut computation (d) Second clustering operation

Algorithm 1 explains proposed Flow based netlist partitioning technique.

Algorithm 1: Flow based netlist partitioning

For a given netlist, numbers of cells/instances are represented by 'V' or 'n' which represents vertices / nodes in graph representation of netlist. Connection between cells/instances is represented by 'e' which represents edges in graph representation of netlist.

Step 1. Pre_placement of instances: Initial placement of cells based on the position of cells defined in netlist file.

Step 2. Define source and sink node: The flow between vertices selected as source and sink may not cover the complete flow network, if two nodes of the netlist are selected randomly as source and sink nodes, which results in unbalanced partitioning solutions. To address this problem instances which are located near core boundary will be considered as source and sink.

Step 3. Convert given netlist into a graph, in which a graph has nodes representing cells, edges representing connection between cells with edge capacity defining current flow between nodes.

Step 4. Set initial flow to infinity. I.e. Present flow $\leftarrow \infty$

Step 5. Apply Max flow Min cut algorithm in order to partition netlist into two parts : The max flow min-cut theorem is defined as in any flow network, maximum amount of flow that is being passed from source node to sink node is equal to sum of weight of the edges in the minimum cut, i.e. the smallest total weight of the edges which if removed would disconnect the source from the sink. The following steps are performed to find max flow and min cut for flow network.

i. Initialize the variables V=100,n=0,e=0, capacity[][], flow[][]; (V= number of vertices, n= number of nodes, e= number of edges, capacity[][] = residual capacity matrix, flow[][]= actual flow matrix).
ii. Read graph values from keyboard.
iii. Print read values.
iv. Loop for n-2 iterations (n represents number of nodes) //Ford Fulkerson algorithm

A. Call function minCut and max_flow.
   a. Define a residual graph; store the capacities in the original graph into residual graph.
   b. Augment the flow if a path exists from source node to sink node. An augmenting path is a path from source to sink, which is a simple path and does not include any cycles. Augmenting path passes only through positive weighted edges.
   c. BFS (breadth first search) function returns true, only if there exists a path from source node to sink node in residual graph. (s→ source, t→ sink)
   d. Create an array which represent visited array. All vertices are marked as not visited.
   e. Create a queue of visited vertices, check source vertex and source vertex is marked as visited.
   f. If sink is reached in BFS process if started from source, then return true, else return false.
   g. Find the minimum residual capacity of the edges along the path which is filled by BFS. In other words through the path which is found, find the maximum flow through it. Residual capacity indicates how much flow is still allowed in each edge of the network graph. All edges have strictly positive residual capacity.
   h. Reverse edges along the path found. Update residual capacities of each edge.
   i. Max flow is obtained.
   j. Find the vertices reachable from source using dfs (depth first search) function.
   k. The dfs function marks elements of array, visited[i] as true, if i can be reachable from source. Initially values in array visited [] must be false. Where i is any node in graph.
   l. Print all the edges those are from a reachable vertex to non-reachable vertex in the original graph. Min cut is obtained.

B. Print min cut edges in ith iteration.
C. Print max flow that is obtained ith iteration.
D. If (present flow is not equal to zero and greater than past flow) then
   Print max flow attained and terminate program.

```
            Else
            Assign present flow to past flow.

        E.End If

    v.  End loop
```

Step 6. Merge cut edge node with neighbor node. Cut edge node is merged with a node which has minimum flow difference between actual capacity and residual capacity

Step 7. Update flow values of graph.
Step 8. Print flow graph (represented by capacity flow graph)
Step 9. End

IV. EXPERIMENTAL RESULTS AND ANALYSIS

Proposed netlist partitioning algorithm is implemented using C++ and simulated using open source cross-platform IDE that is Code Blocks. Proposed algorithm was implemented on MCNC benchmark circuit. A Block netlist "hp" is used for implementation, which is in "yal" text format. The MCNC benchmark was chosen because netlist has information like cell dimension which was helpful to perform initial placement of cells, network section gives information of interconnection between cells/modules that was useful to get edges in graph representation. Random current flow values are assigned to each edge in the graph, as MCNC benchmark suite doesn't have current information between cells. Netlist is converted into graph format using python scripting language and simulated using ANACONDA which is an open source distribution for Python. Proposed algorithm took less than 5 seconds to execute for these benchmark circuits on a PC with 4GB RAM. Table 1 gives details of MCNC benchmark circuit HP.

TABLE 1: MCNC BENCH MARK CIRCUIT

Benchmark	Modules/ Cells	Total number of nets	Number of Pins	No of iterations	Run Time (in seconds)
HP	11	83	309	4	1.97

Table 1 describes the No of cells, No of pins and total nets under benchmark. Along with that the no of iterations performed to partition netlist and run time are specified in table. For the benchmark circuit "HP" netlist has 11 modules where each module represents a cell/instance, 309 pins and 83 nets. Among 83 nets, only nets with forward direction are taken to form directed graph since implemented algorithm only considers forward flow. "HP" circuit takes 4 iterations for netlist partitioning and it takes 1.97 seconds to execute the code.

Following section describes the partitioning process using bench mark circuits "HP".

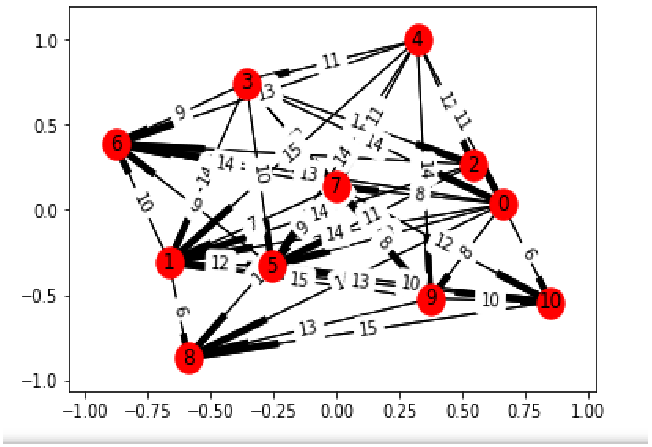


Fig 4. Graph representation of netlist with edges assigned with random current values for "HP" benchmark circuit.

Fig 4 shows initial flow graph with random current values assigned to edges for "HP" netlist. For the MCNC benchmark circuit "hp" there are 11 vertices and 41 nets in graphical representation. 0th node is considered as "Source" node and 10th node is considered as "sink" node. So V= [0 1 2 3 4 5 6 7 8 9 10] a list of nodes. Random current values are assigned in the range of 5 to 15. Table 2 shows different iterations and flow values and cut edges at each iteration. Here program stops after 4 iterations when stopping condition is satisfied. Programs prints the max flow attainted, cut edges and which node be merged in each iteration. After each iteration node is merged with neighboring node based on criteria that is a node which has minimum flow difference between actual capacity and residual capacity. At the final iteration max flow obtained is 40 and cut edges are 3-4, 3-5, 3-10, 6-10, 9-10.

TABLE 2: PARTITION OUTPUT AFTER EACH ITERATION FOR "HP"

Iteration	Max flow	Cut edges	Nodes to be merged
1	80	0-1, 0-2, 0-3, 0-10, 5-10, 6-10, 9-10	0,9 (forms super vertex 9)
2	67	1-2, 1-10, 3-10, 5-10, 6-10, 9-10	1,9 (forms a super vertex 9
3	33	2-10, 6-10, 9-10	2, 9 (forms a super vertex 9)
4	40	3-4, 3-5, 3-10, 6-10, 9-10	3,9 (forms super vertex 9)

Finally we get 2 clusters in which cluster 1 has nodes 0,1,2, 3,9 and cluster 2 has nodes 4,5,6,7,8,10. Each cluster corresponds to a power domain which becomes a basic structure for stacked domain implementation. Clustering is observed visually after seeing output of each iteration. Fig 5 shows the visual representation of two clusters.

Cluster1
0, 1, 2,
3, 9
4, 5, 6, 7,
8, 10
Cluster 2

Fig 5. Final partition with two clusters for benchmark circuit "HP"

The flow based partitioning is better compared to KL and FM algorithms, where latter ones perform move based partitioning.

The reasons are listed below:

i. KL and FM algorithms require arbitrary initial partition of vertex set but a global partitioning approach relies on entire graph rather than initial partition otherwise it will affect final solution quality. Proposed algorithm will not take into consideration the initial partition; it works on entire graph providing better quality solution.

ii. Initial edge swapping or vertex movement is done randomly. Whereas flow based partitioning selects source and sink nodes which are at core boundary or in certain situation adds dummy vertices to handle unbalanced partition.

iii. FM and KL algorithms can only form two partitions. Whereas extending flow based algorithm, more than two partitions can be formed which can result in formation of multiple power domains of stacked implementation.

iv. Time complexity of KL algorithm is $O(n^3)$ per pass [6], where N represents number of vertices. For FM time complexity is $O(P)$ per pass [4] , where P is total number of pins. But for flow based algorithm which follows max flow min cut process, total time complexity is $O(|V||E|)$ for a connected circuit N = (V, E), which is same as one max-flow computation [11].

Taking all the above points into consideration, our flow based algorithm is best suited for flow network as well as for stacked domain implementation compared to KL and FM partitioning methods.

V. CONCLUSION

In this paper, the flow based logic design partitioning is demonstrated, which provides basic structure for optimization framework of stacked power domain implementation. Algorithm partition the netlist into two parts based on current flow between the instances. Where each part or cluster corresponds to a power domain. Iterative max flow min cut process partition the netlist. Few improvements are implemented compared base flow based algorithm [1] which includes: 1) V shaped vertices were removed in base algorithm during clustering phase , where in proposed algorithm V shaped vertices are not considered instead clustering is done based on flow difference. 2) Need for Super Source and Super Sink is removed in proposed algorithm where nodes located near core boundary are taken as Source and Sink. The algorithm is implemented in C++ and netlist to graph conversion is implemented using python. Algorithm tested successfully using MCNC bench mark circuit. Partitioning provides nearly balanced currents for one of bench mark circuit. Our future work includes, a. Extending the partitioning approach in obtaining 3 or multiple power domains. b. Reduction in number of level shifter which should be inserted between power domains by using control switch.

ACKNOWLEDGMENT

I would like to extend my token of gratitude to Dr. R Jayagowri, Associate Professor, Department of Electronics and Communication Engineering, B M S College of Engineering, Bengaluru, for her immense support and constant guidance throughout this work and for her valuable suggestions which helped me to complete this work.

REFERENCES

[1] K. Blutman, H. Fatemi, A. B. Kahng, A. Kapoor, J. Li, and J. P. de Gyvez, “Logic Design Partitioning For Stacked Power Domains,” in Proc. IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Aug 2017, pp.3045-3056.

[2] R. L. S. Ching, E. F. Y. Young, K. C. K. Leung, and C. Chu,“Post-placement voltage Island generation,” in Proc. ICCAD, 2006, pp. 641–646.

[3] K. Blutman, Ajay Kapoor, Arjun Majumdar, Jacinto Garcia Martinez, Juan Echeverri, Leo Sevat, Arnoud P, Hamed Fatemi, Kofi A. A. Makinwa and J.P. de Gyvez, “A Low-Power Microcontroller in a 40-nmCMOS Using Charge Recycling,” in Proc. IEEE Journal of Solid-State Circuits, Jan 2017, pp. 950 – 960.

[4] C. M. Fiduccia and R. M. Mattheyses, “A linear time heuristic for improving network partitions,” in Proc. DAC, 1982, pp. 175–181.

[5] K. Blutman, A. Kapoor, J. G. Martinez, H. Fatemi, and J. P. de Gyvez, “Lower power by voltage stacking: A fine-grained system design approach,” in Proc. DAC, 2016, pp.78-1–78-5.

[6] B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *Bell Syst. Tech. J.* vol. 49, no. 2, pp. 291–307, 1970.

[7] K. Ueda, F. Morishita, S. Okura, L. Okamura, T. Yoshihara, and K. Arimoto, “Low-power on-chip charge-recycling DC-DC conversion circuit and system,” IEEE J. Solid-State Circuits, vol. 48, no. 11, pp. 2608–2617, Nov. 2013.

[8] S. K. Lee, T. Tong, X. Zhang, D. Brooks, and G.-Y. Wei, “A 16-core voltage-stacked system with an integrated switched-capacitor DC-DC converter,” in Proc. Symp. VLSI Circuits, 2015, pp. C318–C319.

[9] G. Karypis and V. Kumar, “Multilevel K-way hypergraph partitioning,” in *Proc*. DAC, 1999, pp. 343–348.

[10] A. E. Caldwell, A. B. Kahng, and I. L. Markov, “Improved Algorithms for Hypergraph Bi partitioning,” in Proc. ASP-DAC, 2000, pp. 661–666.

[11] H. Yang and D. F. Wong, “Efficient network flow based min-cut balanced partitioning,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 15, no. 12, pp. 1533–1540, Dec. 1996.

[12] K. Blutman, H. Fatemi, A. B. Kahng, A. Kapoor, J. Li, and J. P. de Gyvez, “Floorplan and placement methodology for improved energy reduction in stacked power-domain design,” in Proc. ASP-DAC, 2017, pp.444–449.

[13] R. Rajaraman and D. F. Wong, “Optimum clustering for delay minimization,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 14, no. 12, pp. 1490–1495, Dec. 1995.

[14] K. Blutman et al., “A microcontroller with 96% power-conversion efficiency using stacked voltage domains,” in Proc. IEEE Symp. VLSI Circuits, Jun. 2016, pp.1–2.

[15] A. E. Caldwell, A. B. Kahng, and I. L.Markov, “Optimal partitioners and end-case placers for standard-cell layout,” IEEE Trans. Comput.-AidedDes. Integr. Circuits Syst., vol. 19, no. 11, pp. 1304–1313, Nov.2000.

[16] L. Guo, Y. Cai, Q. Zhou, and X. Hong, “Logic and layout aware voltage Island generation for low power design,” in Proc. ASP-DAC, 2007, pp.666–671.

[17] https://www.geeksforgeeks.org/minimum-cut-in-a-directed-graph/

[18] https://cseweb.ucsd.edu/classes/wi09/cse242a/partition/ part.pdf