CAIRO UNIVERSITY

GRADUATION PROJECT THESIS

Dijkstra aided Global Automatic Router (D-GAR) Tool

Authors:
Mohamed Hany Fayez
Mostafa Mohamed Bahaa
Sara Abd El-Latif Mohamed

Supervisors:
Dr. Hassan Mostafa
Associate Professor
Engr. Fady Atef
Layout Team Manager

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

under the academic supervision and industrial sponsorship of

ONE-Lab, Cairo University
Si-Vision, Synopsys Inc.

Department of Electronics and Electrical Communication Engineering

"Indeed, those who have believed and done righteous deeds – indeed, We will not allow to be lost the reward of any who did well in deeds."

The Holy Quran, Surat Al Kahf 30

Acknowledgements

First and foremost endless gratitude to Allah who has granted us with the blessing of time, effort, and health invested in this project. Second, our families who endured and supported us through this five-year journey. And finally, our mentors, Dr. Hassan Mostafa and Engr. Fady Atef, to whom a great dept is owed. By this project we only hope it fulfills repaying a small portion of.

Contents

List of Figures

Abstract

This thesis presents the graduation project work done in the analog layout automation tool. The team was academically sponsored by ONE Lab, Cairo University and industrially by Si-Vision, Synopsys Inc. The tool presented in this work aims to greatly cut down the time needed by the analog layout designer in finalizing his first top routing iteration. The tool employs innovative solutions adopted from literature and modified such that it fits the requirements.

Chapter 1

Introduction

1.1 Motivation

ICs are becoming more complex and highly integrated to meet the new technologies requirements. Gradually transforming to what is called SoCs. Electronic Design Automation (EDA) tools are thus required to handle the difficult design rules associated with smaller technology nodes, as well as the tight circuit performance constraints.

Many state-of-the-art EDA tools have shown success in the digital domain. However, more efforts are still needed for Analog/Radio Frequency (RF) circuits due to their high performance sensitivity to the various layout design reliability and integrity concerns. Moreover, the inherent error-prone manufacturing process representing another obstacle to overcome.

The analog layout synthesis problem has attracted considerable interest for the past several decades. Unfortunately, it has not seen widespread adoption by circuit layout designers as a feasable solution. The traditional perception has been that the these tools' results are unable to match the expert designer, both in terms of his ability to comprehend and implement specialized layout tricks, and in terms of the variety of topologies with circuit-specific constraints.

Rule-based methods is the direct approach, and it was considered as first generation approaches. However, distilling the layout designer's experience and notion of the circuits in hand into a limited set of rules can be challenging. In recent years, the analog layout design landscape has shifted tides in several ways more favorable for automation, some of which are listed as follows:

- Fixed pitch design rules and Manhattan routing approach in modern nm technology nodes has opened opportunity windows for incorporating rule-based procedures.
- More analog blocks are required in integrated systems making it more of a repetitive task with the same design considerations and reliability issues mitigation techniques.
- The advancements in the field of algorithms and the available computation powers to tackle such automation problem.

The aforementioned reasons provide opportunities for solving the analog layout automation problem in a manner that was not previously possible.

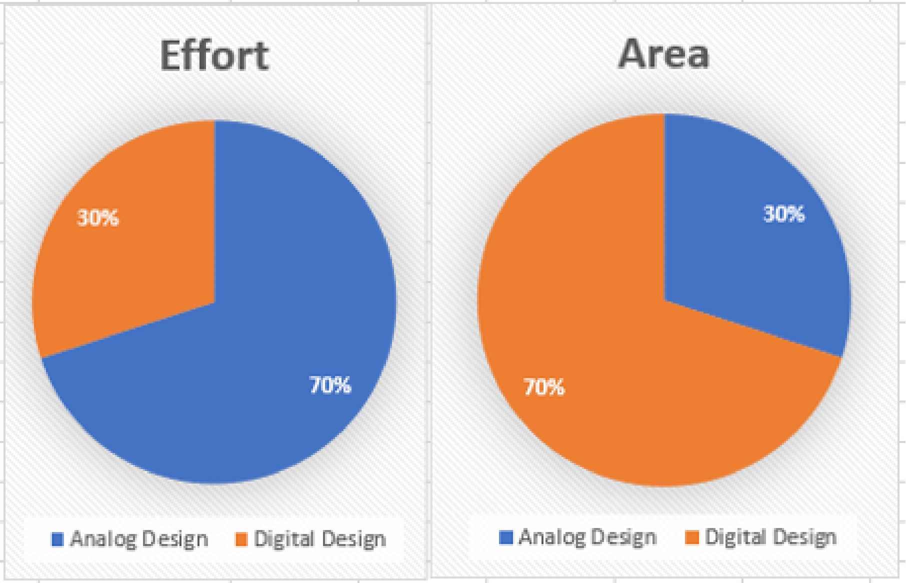


FIGURE 1.1: Area Vs. Effort of Chip

Even for high-performance blocks, although automatically generated layouts are unlikely to meet all the specifications, they will considerably reduce the iterations between circuit optimization and layout. Such cases are where layout generation is the primary bottleneck.

Fig 1.2 draws comparison between the analog and digital partitions of a typical IC. It is clear that although the digital partition comprises around 70% of the chip's area, it only requires around 30% of the exerted effort. On the other hand, the analog partition is the opposite in terms of area and effort.

This implies that the market requires an analog layout automation tool to help in cutting down the time and effort needed in the layout design process. The market naturally requires the tool to offer innovative and computationally simple solutions for the various analog layout considerations such as route matching, current capability handling, routes variable width, assigning metal layers to signal types, lack of standard cells and others.

1.2 Full Tool Overview

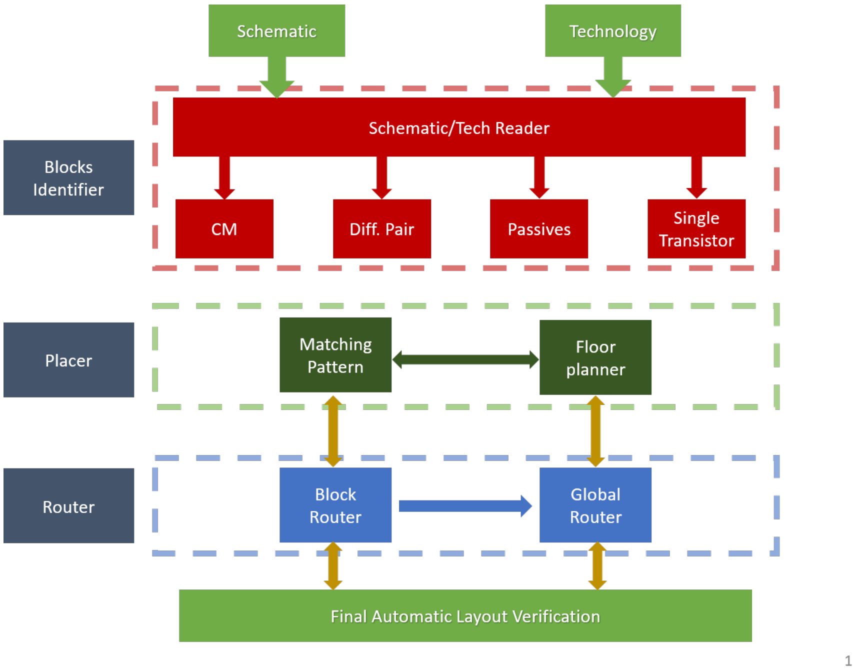


FIGURE 1.2: Full Tool Overview.

In the last few years a lot of work in the layout synthesis task of analog integrated circuits has been performed to automate the layout ,that work emerged from universities ,some found their way into commercial EDA tools but unfortunately these tools are still far away from reality and this is due to the fact that the design of analog circuits is complex as there is countless interactions between them and going to smaller technology nodes makes the task harder as the design rules becomes more complex, and the interactions impact becomes greater. Layout generation and verification process flow consists of the following main steps given the schematic and the technology node the layout engineer does the floor planning then routing followed by physical verification ,PEX and finally post layout automation.

The methodology presented in this thesis focuses on the global router block which is part of a full layout automation tool. Before starting to talk about it the main block flow , lets take a closer look to the other blocks of the full tool that are mentioned earlier in the layout generation and verification process flow . So the first block is schematic reader , from its name all it does is identifying what is inside the block such as current mirrors, differential pairs , capacitors ,resistors and single transistors, and of course this block will perform this task given the schematic and the technology node. Next block is the placer and this block flow is as follow : Matching Pattern generator and Floor planner.
To understand the matching pattern generator ,let's take a closer look to the matching process itself.

Matching is the process of creating similar environment for a set of transistors so they exhibit the same electrical properties and that guarantee that the desired functionality are not altered . There are two types of mismatch :systematic mismatch that one is due to non uniform thermal distribution during fabrication process ,it can be solved by proper layout techniques ,best device matching so as to be as close as possible , the second type is random mismatch the error resulting from that one can't be identified or controlled while implementing the layout ,it will happen during the fabrication and the reasons behind that are the non uniform etch rate ,the doping and finally the wafer itself .Regarding the wafer problems are due to the mask misalignment problems. The matching pattern is done applying techniques such as interdigitation and common centroid where Interdigitation means that the devices are distributed in an interleaved manner it's more preferred for diff pairs as it guarantees that both transistors see the same process variations so they'd be matched under all conditions on the other hand common centroid techniques the devices are symmetrically laid out about a certain axis so as to have a common center this method is common for current mirrors and capacitors it increases matching. It is worth mentioning that dummies are used to balance the effects on the lateral transistors ,they may be used as well to shield all around the devices in smaller and more sensitive technologies, they are used as well to combat the LOD effect and the well proximity effect as well.

The LOD effect it occurs as a result of the device characteristics variations according to the distance of its gates from the diffusion edges so dummies are added to cancel this effect by making both distances the same. The well proximity effect is due to the variation of performance for transistors that are placed close to the well edge than the ideally placed ones that is caused during the ion implantation step when the ion beam is tilted causing non uniform ion distribution on the transistors that are closed to the well edge so dummy placement solves this issue as well.

The Floor planner block is very important ,analog mask designers must be involved in the floor planning not only it saves time but makes signal flow more efficient specially when the floorplan is very convoluted ,wires all over the place which introduce coupling mechanisms as well as parasitics and this will for sure affects the circuit performance. So the blocks relative positioning must be done so as to make sure the floorplan is driven by the following prominent concerns: pin-out, block placement and signal flow. A good choice of pin-out could reduce parasitics and help the mask designer produce a clean layout. Regarding the block placement part ,it will help to understand how the top level assembly will be performed so the main goal is to keep the inter-block wires as short as possible and to avoid wires running all around the chip. Once a clean and satisfying floorplan is ready ,the signal flow part becomes more easier .Now , the way of thinking about the signal flow depends on what are the concerns, so in case of worrying about how the internal blocks talk to each other then the insides will drive the pin out but if the main concern is about how the pins interact and connect with each other , then the pins will drive how the blocks are placed inside. One major thing to think about while working on the chip floorplan is the wiring , the area needed for wiring is quite important , that matter needs to be discussed with the circuit designer so as to be informed about the area needed for differentiall signals ,for special symmetry or perhaps for additional isolation .The B*trees is the algorithm used to implement an automated floorplanner.

Before directly talking about the next step which is routing , lets take a look into some of the important concepts in routing :

- Parasitic capacitance which is unavoidable and unwanted and may severely limit the performance specially at high frequencies ,they are common solutions in order to decrease capacitance such as reduce the wire length so as to reduce the overlap between the wires and substrate ,go to higher metal layer away from the substrate ,use minimum metal width and maximum oxide thickness.

- Electromigration that happens when using thin metals that can no longer bear high current density consequently atoms get displaced from their original positions causing voids in the metal layer .Ways to solve this problem : widen the wire to reduce current density ,reduce the frequency ,lower the supply voltage , keep the wirelength short.

- Antenna effect that takes place due to plasma etching charge accumulation over a thin gate metal wire which leads to oxide breakdown ,that why there is an antenna rule that defines the max metal area to gate oxide area to avoid the damage and according to that rule the voltage that the gate can put up with will be calculated .In order to solve antenna effect ,high level metal jumpers and antenna diodes are used.

$$\text{AntennaRule} = \frac{\text{Metal area}}{\text{Gate oxide area}} \alpha \frac{Q}{C} \tag{1.1}$$

$$V = \frac{Q}{C} \tag{1.2}$$

- Supply noise which might ruin your IC ,decoupling capacitors between the power and ground rails are used , using separate supplies for each domain is a solution as well.

- Parasitic resistance which is determined by calculating the sheet resistance of each metal layer ,and it must be suitable with the required voltage drops , in order to reduce it one of the tips is using wider metals ,higher metals levels and metal stack as the parallel equivalent is the smallest resistance.

- Coupling noise between blocks and here shielding plays a major role and this shielding can be done using guard rings ,it could lateral or all around depending on the available area.

All the previous problems has to be taken into consideration , it has to be mentioned that a good layout designer has to go back and forth from floor planning to routing as they are tightly connected and they affect one another . Now comes the router block turn, this one is divided into block level router , global router. Lets take a closer look to the block level router , after finishing the matching patterns and the floorplan of the block itself , it's time to do the internal block routing that targets minimum route sizing , symmetry specially for the differential signals and minimum parasitics as well so as to have a satisfying performance. To be able to automate the interconnections between the devices a netlist is given to be able to find the path between different nets using pathfinder algorithms which are part of the maze algorithms that is used in routing automation. The basic concept of the approach is to make a connection between two terminals avoiding any obstacles which in that case are other devices and wires .This is implemented usually by using a grid like to avoid DRC violation . Another approach can be used such as template based techniques that uses a pre-defined template that has the relative positions and interconnection of devices. It has to be mentioned that wiring symmetry is very critical specially in the routing of differential circuits. So to achieve complete symmetry for differential circuits ,symmetrical placement and symmetrical routing should be satisfied . The global router which is the thesis scope its goal is to place routes between different blocks taking into our considerations the same constraints mentioned earlier in the block router. A quick look to block flow diagram that will be discussed later on in details ,the block flow starting from signal identifier and classifier which identifies pin pairs having the same net name ,it classifies the signal according to its type and each signal is prioritized so supply signal comes first followed by current signals then voltage signals and finally the digital signals . The second block is the channel estimator its role is to detect the vertical and the horizontal channels and the intersections between them ,the following block is the node builder that virtually allocate pins to the channel intersections called nodes by means of proximity, the next block is the shortest path finder its goal is to virtually allocate the routing paths to nets using Dijkstra routing algorithms and finally the detailed router its role is to find the routes placement coordinates and its output is an executable TCL script that will place the physical routes .

1.2.1 Schematic/Tech. File Reader (Parser)

1.2.2 Placer

An automatic placement tool should produce solutions similar to manual layouts in density and performance In order to reduce parasitics have better process variations on different conditions, placement of analog devices must be made taking simultaneously many requirements into consideration. Which appears in the form of symmetry, matching and proximity constraints, allied to the multitude of possible device implementations, with different sizes and aspect ratios, make the analog placement task hard to automate.

Furthermore, these constraints must be strictly satisfied while attempting to minimize several objectives, such as chip area, interconnect length or parasitic impact. This is fundamental as the attainable routing quality and, most of the parasitic effects and consequent post-layout circuit's performance degradation are set once a placement solution is fixed.

1.2.3 Router

Routing is one of the final steps in analog layout synthesis. Net-list doesn't provide information about how the device's terminal should be connected, because analog circuits performance dependent on layout parasitics and needs more attention than digital the quality of routing is very critical in the circuit performance.

Input:

- Routing region: multi-layer rectangle
- Obstacles: size/location
- Pins: location
- Net-list

Output:

- Routed paths for all nets

Constraints:

- Routing resources
- Connection rules
- Design rules

routing problem is usually solved by using a two-stage approach of global routing followed by detailed routing due to its complexity. Global routing first partitions the routing region into channels then decides for each net while optimizing a certain cost function the whole path given the resources ex. (total channel width and signal type), while the detailed router assigns the actual wires and vias in these channels.

The detailed routing algorithms are classified into:

- Grid-based routing
- Gridless routing

For grid-based routing, a routing grid is superimposed on the routing region, and then the detailed router finds routing paths in the grid and the gridlines are called wire pitch which is defined as larger than or equal minimum wire width and spacings sum,while the The gridless detailed routing model does not follow the grid-based model.so we can use different wire widths and spacing

Chapter 2

Literature Review

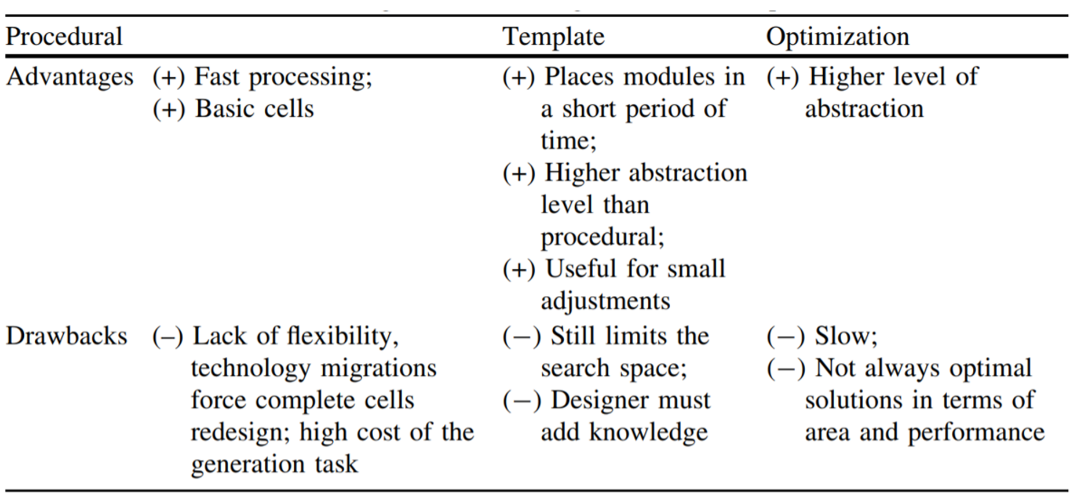
In this section, some of the milestones in the analog layout generation, along with some recent tools, will be reviewed. In the earliest approaches, procedural module generation techniques coded the entire layout of a circuit in a software tool, which would generate the target layout for the parameters attained during sizing. This parametric representation of the layout is fully developed by the designer, either by a procedural language or a graphical user interface (GUI). ALSYN employs fast procedural algorithms that are controlled through a database of structures and attributes. Although fast, these methods lack the flexibility to accommodate wide changes, making the cost of introducing a new design task relatively high and technology migrations may force complete cells redesign.

The use of template approaches, which define the relative position and interconnection of devices, is a common practice. A template-based generation is used by Intellectual Property Reuse-based Analog IC Layout (IPRAIL) to automatically extract the knowledge embedded in an already made layout, and use it for retargeting. Layout retargeting is the process of generating a layout from an existing layout. The main target is to conserve most of the design choices and knowledge of the source design, while migrating it another given technology, update specifications or attempt to optimize the old design.

In order to retain the knowledge of the designer but without forcing an implicit definition, LAYGEN [1] tool for example, uses a template-based approach to guide the layout generation. ALADIN also allow designers to integrate their knowledge into the synthesis process. While ALG uses the same knowledge-based principle, allowing the designer to interact with the tool in different phases, leaving to the discretion of the designer if the final layout is obtained almost full automatically or by designer directives.

Zhang et al. developed a tool that automatically conducts performance constrained parasitic-aware retargeting and optimization of analog layouts. Performance sensitivities with respect to layout parasitics are first determined, and then the algorithm applies a sensitivity model to control parasitic-related layout geometries, by directly constructing a set of performance constraints subject to maximum performance deviation due to parasitics.

The optimization-based layout generation approaches consist of synthesizing the layout solution using optimization techniques according to some cost functions, with a higher level of abstraction. The simulated annealing and genetic algorithms are the most common choice for solving analog



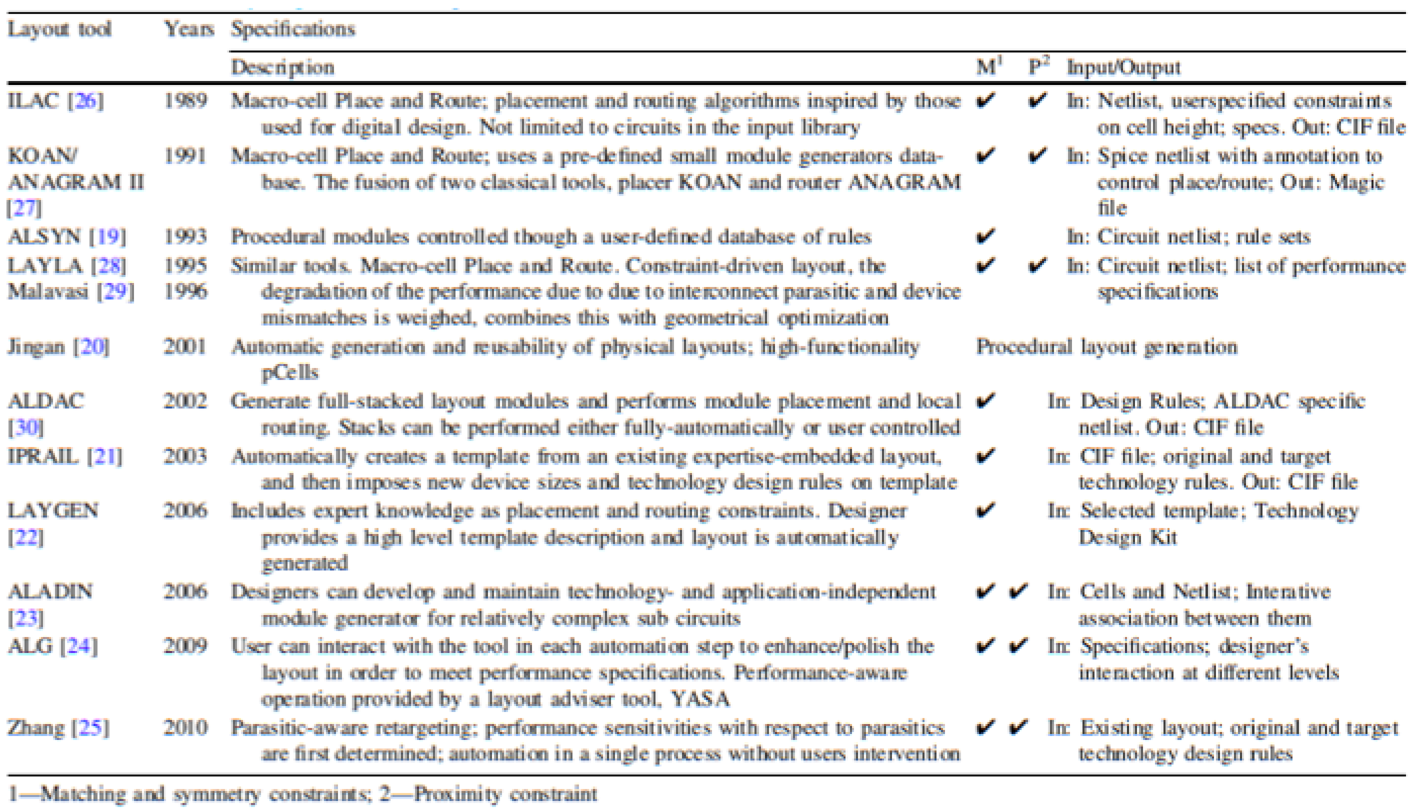
Procedural		Template	Optimization
Advantages	(+) Fast processing; (+) Basic cells	(+) Places modules in a short period of time; (+) Higher abstraction level than procedural; (+) Useful for small adjustments	(+) Higher level of abstraction
Drawbacks	(–) Lack of flexibility, technology migrations force complete cells redesign; high cost of the generation task	(−) Still limits the search space; (−) Designer must add knowledge	(−) Slow; (−) Not always optimal solutions in terms of area and performance

FIGURE 2.1: Classification of analog tools based on generation techniques

device-level placement problems, beyond their flexibility in terms of incremental addition of new functionalities; they are relatively easy to implement.

Several publications were reviewed to conduct the literature review. Papers in [2] - [3] - [4] - [5] - [6] - [7]. Analog Layout Synthesis [8] book was particularly helpful. Also books in [9] and [10] were reviewed. Moreover, publications in [11] - [12] - [13] - [1] - [14] - [15] were reviewed.

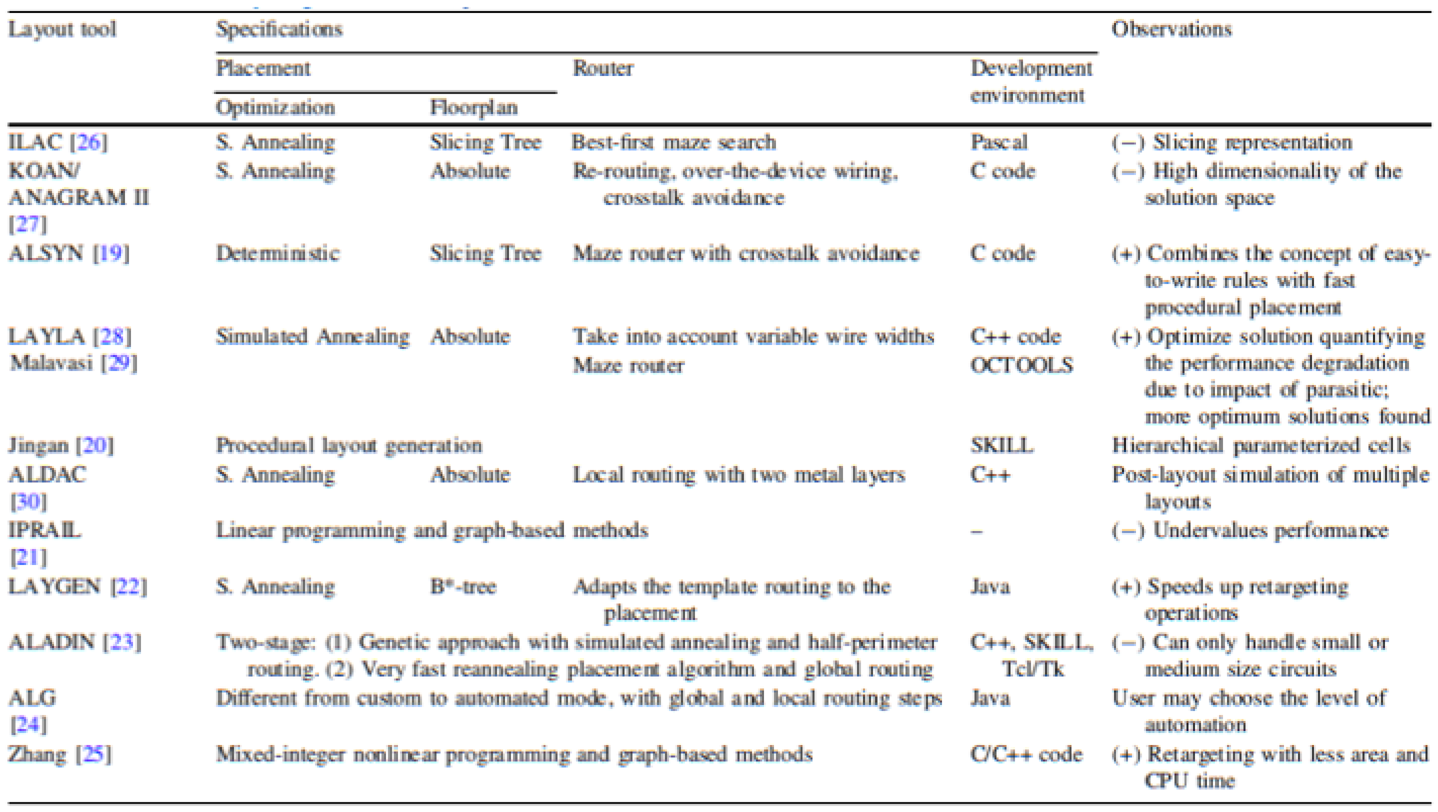
In Fig. 2.1, a classification of the analog tools presented in this section based on generation techniques is presented, and a summary of the advantages of each technique is also highlighted. A summary of the description and functional specifications of the referred tools is presented on Fig. 3.1, while on Fig. 2.3 technical specifications and few observations are reported



Layout tool	Years	Specifications			
		Description	M[1]	P[2]	Input/Output
ILAC [26]	1989	Macro-cell Place and Route; placement and routing algorithms inspired by those used for digital design. Not limited to circuits in the input library	✔	✔	In: Netlist, userspecified constraints on cell height; specs. Out: CIF file
KOAN/ ANAGRAM II [27]	1991	Macro-cell Place and Route; uses a pre-defined small module generators database. The fusion of two classical tools, placer KOAN and router ANAGRAM	✔	✔	In: Spice netlist with annotation to control place/route; Out: Magic file
ALSYN [19]	1993	Procedural modules controlled though a user-defined database of rules	✔		In: Circuit netlist; rule sets
LAYLA [28] Malavasi [29]	1995 1996	Similar tools. Macro-cell Place and Route. Constraint-driven layout, the degradation of the performance due to due to interconnect parasitic and device mismatches is weighed, combines this with geometrical optimization	✔	✔	In: Circuit netlist; list of performance specifications
Jingan [20]	2001	Automatic generation and reusability of physical layouts; high-functionality pCells	Procedural layout generation		
ALDAC [30]	2002	Generate full-stacked layout modules and performs module placement and local routing. Stacks can be performed either fully-automatically or user controlled	✔		In: Design Rules; ALDAC specific netlist. Out: CIF file
IPRAIL [21]	2003	Automatically creates a template from an existing expertise-embedded layout, and then imposes new device sizes and technology design rules on template	✔		In: CIF file; original and target technology rules. Out: CIF file
LAYGEN [22]	2006	Includes expert knowledge as placement and routing constraints. Designer provides a high level template description and layout is automatically generated	✔		In: Selected template; Technology Design Kit
ALADIN [23]	2006	Designers can develop and maintain technology- and application-independent module generator for relatively complex sub circuits	✔	✔	In: Cells and Netlist; Interative association between them
ALG [24]	2009	User can interact with the tool in each automation step to enhance/polish the layout in order to meet performance specifications. Performance-aware operation provided by a layout adviser tool, YASA	✔	✔	In: Specifications; designer's interaction at different levels
Zhang [25]	2010	Parasitic-aware retargeting; performance sensitivities with respect to parasitics are first determined; automation in a single process without users intervention	✔	✔	In: Existing layout; original and target technology design rules

1—Matching and symmetry constraints; 2—Proximity constraint

FIGURE 2.2: Overview of layout generation tools, part I



Layout tool	Specifications				Observations
	Placement		Router	Development environment	
	Optimization	Floorplan			
ILAC [26]	S. Annealing	Slicing Tree	Best-first maze search	Pascal	(−) Slicing representation
KOAN/ ANAGRAM II [27]	S. Annealing	Absolute	Re-routing, over-the-device wiring, crosstalk avoidance	C code	(−) High dimensionality of the solution space
ALSYN [19]	Deterministic	Slicing Tree	Maze router with crosstalk avoidance	C code	(+) Combines the concept of easy-to-write rules with fast procedural placement
LAYLA [28]	Simulated Annealing	Absolute	Take into account variable wire widths	C++ code	(+) Optimize solution quantifying the performance degradation due to impact of parasitic; more optimum solutions found
Malavasi [29]			Maze router	OCTOOLS	
Jingan [20]	Procedural layout generation			SKILL	Hierarchical parameterized cells
ALDAC [30]	S. Annealing	Absolute	Local routing with two metal layers	C++	Post-layout simulation of multiple layouts
IPRAIL [21]	Linear programming and graph-based methods			–	(−) Undervalues performance
LAYGEN [22]	S. Annealing	B*-tree	Adapts the template routing to the placement	Java	(+) Speeds up retargeting operations
ALADIN [23]	Two-stage: (1) Genetic approach with simulated annealing and half-perimeter routing. (2) Very fast reannealing placement algorithm and global routing			C++, SKILL, Tcl/Tk	(−) Can only handle small or medium size circuits
ALG [24]	Different from custom to automated mode, with global and local routing steps			Java	User may choose the level of automation
Zhang [25]	Mixed-integer nonlinear programming and graph-based methods			C/C++ code	(+) Retargeting with less area and CPU time

FIGURE 2.3: Overview of layout generation tools, part II

Chapter 3

Proposed Global Router Flow

3.1 Introduction

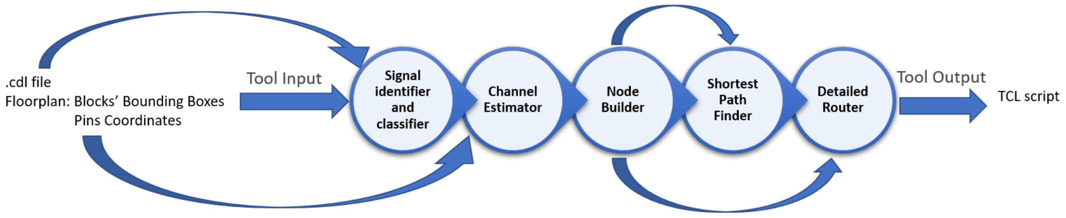


FIGURE 3.1: Proposed Global Router Flow Block Diagram

The proposed flow aims at presenting a novel approach to the analog layout automation problem. Previous work reviewed in Chapter 2 proved to lack the essence of how the analog layout designer thinks and resorted to purely, brute--force mathematical solutions instead.

This work's solution revolves around that missing essence. Two main characteristics of the proposed solution are:

- The tool partitions the problem into smaller, more clear sub problems.
- The tool then finds consistent procedures for the individual sub problems.

These characteristics are in conformity with how a typical layout designer would approach such a problem.

It is worth mentioning that the purpose of this tool is **not** a click-to-layout automation. Rather, it is an aiding tool for the layout designer in his tedious, time-consuming task.

A key objective of the tool is cutting down required time to laying out the top design. The designer can in turn tune the individual modules' input, allowing him to explore the feasible solution space, and create a wider room for himself to optimize it. Finally, the designer chooses a solution that best fits his needs. The tool will help accelerate the sign-off process by boosting the layout design-circuit design feedback loop.

Python scripting language was used for prototyping the tool. Its huge open-source community as well as its easy syntax made it the perfect choice.

Synopsys Custom Compiler design tool custom TCL scripting language was also used for interfacing the main Python script with the tool for implementing the final output.

The proposed flow in Fig.3.1 can be summarized as follows:

- Tool Inputs:
  - Floorplanner output .txt file describing the top floorplan blocks' bounding boxes relative to the top hierarchy origin.
  - Floorplanner output .txt file describing the pins locations relative to the top origin.
  - .cdl file. A Custom Compiler file describing the top floorplan electrically. i.e. net names associated to which blocks.
- Tool Modules:
  1. Signal Identifier and Classifier (.cdl Reader)
     - Inputs:
       - .cdl File
     - Outputs:
       - Categorized list of:
         - Net names.
         - Net type (Supply, Voltage, Current, Digital Control).
         - Net width, pre-assigned by circuit designer.
  2. Channel Estimator
     - Inputs:
       - Floorplanner's output describing block's bounding boxes relative to the top.
     - Outputs:
       - Vertical and horizontal channels bounding boxes.
       - Nodes bounding boxes. Nodes being defined as: The intersection between two or more horizontal and vertical channels.
  3. Node Builder
     - Inputs:
       - Floorplanner's output describing the pins locations relative to the top.
       - Channel Estimator's output.
     - Output:
       - Associating pins to nodes by means of proximity.
  4. Shortest Path Finder
     - Inputs:
       - Channel Estimator's output.

        - Node Builder's output.
    - Outputs:
        - Virtual allocation of channels to nets on a shortest path basis.
        - The procedure adopts the Dijkstra routing algorithm to the analog layout global routing problem.
        - Nets are allocated to channels in the following order:
            - Supply nets.
            - Sensitive voltage nets.
            - Sensitive current nets.
            - Digital control nets.

5. Detailed Router
    - Inputs:
        - Node Builder's output
        - Shortest Path Finder's output.
    - Outputs:
        - TCL script executed from within Synopsys Custom Compiler tool to physically place the routes and their associated vias.

- Tool output:
    - A feasible routed top layout solution that is based on typical analog layout designers insights.

The tool concept is promising, and naturally is still in its early stages. Nevertheless, this work can be perceived as a successful proof of concept. This work further needs optimization to the code architecture on both the implementation and data structures fronts. In addition to that, for a more realistic representation of the layout designer insight, the tools needs several features addition.

3.2 Signal Identifier and Classifier (.cdl Reader)

The role of that block is as Follow:

- Identifies pin pairs having the same net name.
- Classifies and prioritize the identified pin pairs by their signal type according to the following criteria:[Supply, Voltage, Current, Digital Control].

  The input to that block is a Cdl file : The Required format is as follow:

```
[Xblockints  signalType _ signalName _ widthInMicroMeter]
```

Example:

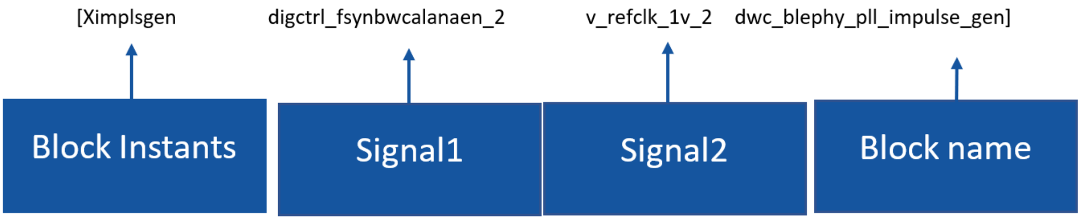


FIGURE 3.2: Cdl Example

By using Python File handling to extract the needed information from the file and then the following output is obtained: Output Text Format:

```
[signal_name  _  netWidth  _  signalPriorty]
```

The signal priority can be easily determined by knowing the signal type ,supply takes the highest priority followed voltage then current and finally digital control. Digital control signal is a noisy signal , voltage signal is noise sensitive and the current signal acts like as a shield that protects the voltage signal from digital control signal. So the supply takes a weight that corresponds to its priority which is the highest its priority weight is :4 followed by the voltage with priority equals :3 then the current will be :2 and finally the digital control priority which is:1 The cdl reader along with the floor planner output are the input to the next block which is the channel estimator.

3.3 Channel Estimator

Adhering to the basis upon which the tool was implemented, this module represents a corner stone. The Channel Estimator module is responsible for identifying and reporting the routing channels' bounding boxes given the blocks' bounding boxes. Bounding boxes being defined as the bottom left and top right corners coordinates. The module is independent of the previous one, Signal Identifier and Classifier module, and thus both can run simultaneously.

The Channel Estimator algorithm represents the first contribution in this work. Owing to the fact that no prior work, as far as the conducted literature review went, adopted the same approach of partitioning the problem as this work does. Moreover, no prior work introduced clear definitions of routing channels, not to mention utilized it.

The algorithm employs Python's Numpy library. Not only does Numpy arrays offer useful functions, but also offer strong visual representation, making the debugging process effortless.

In the current implementation, the algorithm assumes that all blocks in the top layout, prohibit the whole metal stack from crossing over it. This is a simplification to ease the coding process, but far from reality. Typically some non-noise sensitive blocks allow high metal layers i.e. above Metal 5, to cross over.

This simplification allows for single definition of the routing channels. The typical case is the existence of several routing channels definitions for each metal layer group sharing the same properties as to which blocks allow them to cross over.

The algorithm can identify routing channels' bounding boxes relative to the top origin, either with zero or non-zero offset. Moreover, it can only deal with both integer and floating-point values of blocks' bounding boxes.

A core step of the algorithm relies on computing a grid resolution for the layout, defined as the GCD of two other GCD values, one along the blocks' x-coordinates and the other along the y-coordinates.

The algorithm generally prefers a GCD value greater than one along both axis so that array sizes are small and can be processed easily by the machine running the tool.

Flowcharts depicted in Figures [3.3, 3.4, 3.5, 3.6] present an illustrative view of the algorithm. Fig. 3.3 shows a bird's eye view of the algorithm. The reader surely notices that the algorithm repeatedly calls several procedures, namely: getLocations, handleCrossOverlaps, handleOverlaps in which Figures [3.4, 3.5, 3.6] provide flowcharts summarizing the procedure core respectively.

The algorithm defines several entities to deal with:

- Vertical Channels: defined as rectangles along the y-axis.
- Horizontal Channels: defined as rectangles along the x-axis.
- Nodes: defined as intersection between two or more channels of opposinte orientation.

The target is to correctly identify and handle overlaps between those entities. The algorithm also identifies two overlap types:

- Cross-Overlaps: defined as channels reported in either orientation, at the same time it completely lies within a reported channel of the opposite orientation.
- Overlaps: defined as portions of channels getting reported more than once.

Cross-Overlaps are handled differently, depending on the input type, if the input is channels or nodes. As for channels, "handleCrossOverlap" procedure require a channel of a certain orientation to lie completely within another of the opposite orientation. The procedure then discards the shorter of the two cross-overlapping channels.

On the other hand, the procedure doesn't require nodes to lie within each other and suffice with just the cross-overlapping nodes being intersected. The procedure then decides which node spans more length in its orientation(taller fit criteria), and trims the overlapping one.

The Channel Estimator algrithm handles self overlaps on a wider fit basis. During the identification, if portions of channels get identified more than once, it is considered an overlap. The overlapping portion takes one of two forms:

- The overlap portion lies completely within another wider channel.
- The overlap portion lies inside another wider channel, but protrudes from it.

Either case results in that the wider channel stays intact while the overlapping channel gets trimmed.

This work used the provided Fig. 3.7 as the reference example and implemented the algorithm to correctly identify it's channels and nodes. The example was particularly helpful as it covered almost all overlapping cases. Figures 3.8 through 3.9 visually presents the algorithm in work. The figures uses the following color code:

- Orange: Analog blocks.
- Green : Vertical Channels.
- Blue : Horizontal Channels.
- White/Violet : Nodes.

To further verify and test the algorithm, three more test cases were evaluated depicted in Figures 3.10 through 3.12. Fig. 3.10 represents a block level layout, i.e. an Opamp or an LNA circuit. Implying that the proposed global routing flow can be applied on both an inter-level and/or intra-level layout seamlessly. Naturally, the proposed flow requires that in case of intra-level routing, devices must be matched, placed and internally routed.

This work considers Fig. 3.11 the main test case for the rest of the flow. Having simple enough geometry, as clear from the figure, it is easy to debug in succeeding modules. Making it a perfect candidate for the proof of concept test case.

Until now, computing grid resolution for floating point coordinates results in huge array sizes, exceeding 1 GB even with boolean representation of elements, corresponding to prolonged run time (about 2 min in worst test case scenario, Fig. 3.12). Therefore, all coordinates are trimmed off their floating point part, sacrificing room for about 3 to 6 routes based on calculations from advanced nodes PDKs.

Late stage testing revealed the runtime bottleneck becomes apparent for array sizes exceeding 500 × 500 elements. However, further inspection of the algorithm identified the single code block resulting in the bottleneck, caused by two nested loops on the generated numpy array representing the layout (see Fig. 3.4). This work leaves optimization and/or re-implementation for the coming years.

In Fig 3.14 the algorithm's output format is shown. It's structure is as follows column wise:

1. Channel orientation.
2. Bottom left X coordinate.
3. Bottom left Y coordinate.
4. Top right X coordinate.
5. Top right Y coordinate.

To conclude, the Channel Estimator algorithm represents the first contribution to this work, which is inherently a contribution by itself. It can handle various geometries of layout, both on top and intra-block level as evident from the presented test cases. Suffering from one, clearly identified, bottleneck that takes the algorithm about 2 to 3 mins. in certain special cases. When compared to counter digital automation tools, some scripts can take hours to finish. So, although the algorithm suffers from a bottleneck, it offers a decent performance.

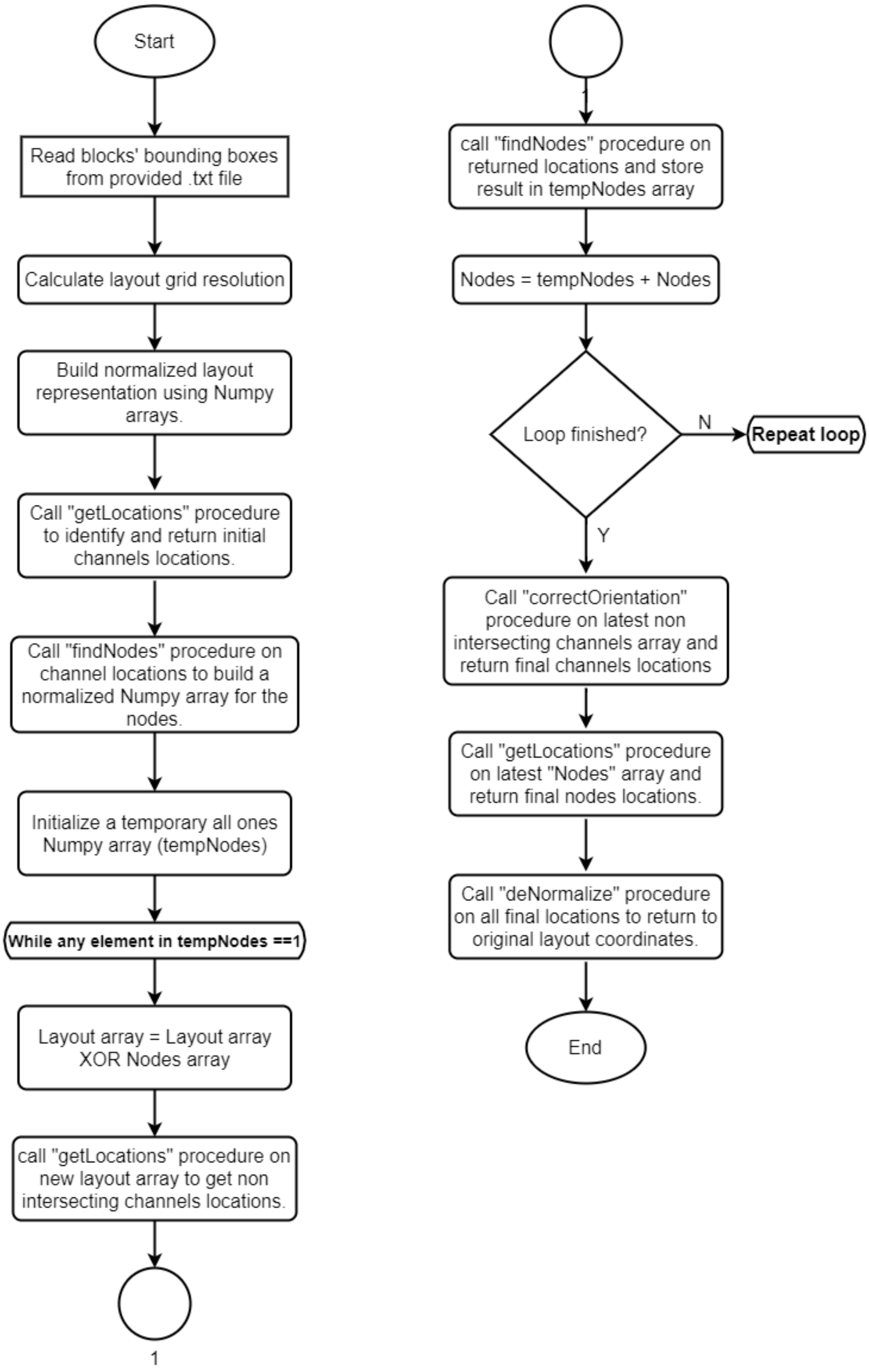

Start
Read blocks' bounding boxes from provided .txt file
Calculate layout grid resolution
Build normalized layout representation using Numpy arrays.
Call "getLocations" procedure to identify and return initial channels locations.
Call "findNodes" procedure on channel locations to build a normalized Numpy array for the nodes.
Initialize a temporary all ones Numpy array (tempNodes)
While any element in tempNodes ==1
Layout array = Layout array XOR Nodes array
call "getLocations" procedure on new layout array to get non intersecting channels locations.
1
call "findNodes" procedure on returned locations and store result in tempNodes array
Nodes = tempNodes + Nodes
Loop finished?
N
Repeat loop
Y
Call "correctOrientation" procedure on latest non intersecting channels array and return final channels locations
Call "getLocations" procedure on latest "Nodes" array and return final nodes locations.
Call "deNormalize" procedure on all final locations to return to original layout coordinates.
End

FIGURE 3.3: Channel Estimator Flowchart

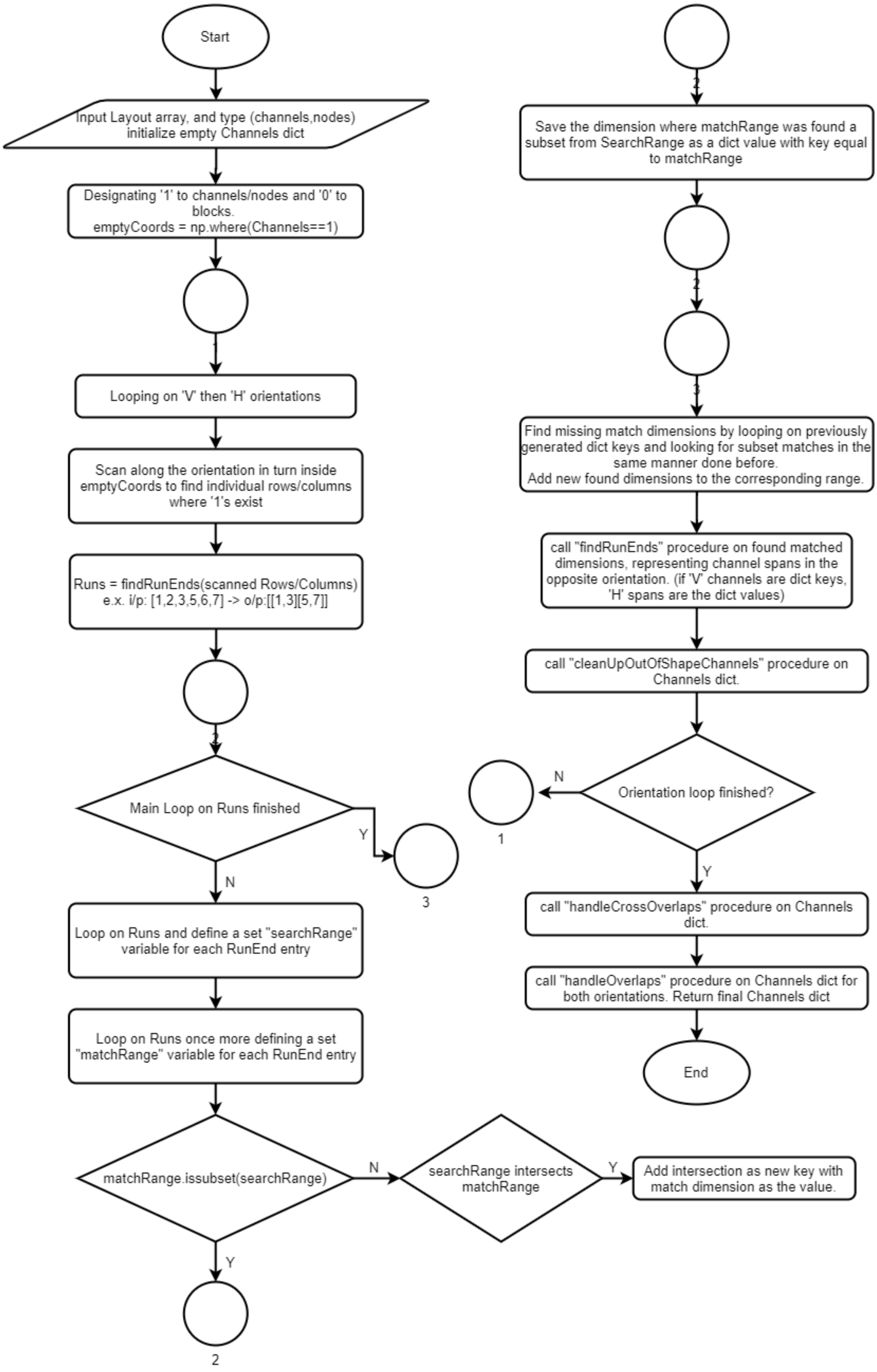

Start
Input Layout array, and type (channels,nodes) initialize empty Channels dict
Designating '1' to channels/nodes and '0' to blocks. emptyCoords = np.where(Channels==1)
1
Looping on 'V' then 'H' orientations
Scan along the orientation in turn inside emptyCoords to find individual rows/columns where '1's exist
Runs = findRunEnds(scanned Rows/Columns) e.x. i/p: [1,2,3,5,6,7] -> o/p:[[1,3][5,7]]
2
Main Loop on Runs finished
Y
3
N
Loop on Runs and define a set "searchRange" variable for each RunEnd entry
Loop on Runs once more defining a set "matchRange" variable for each RunEnd entry
matchRange.issubset(searchRange)
N
searchRange intersects matchRange
Y
Add intersection as new key with match dimension as the value.
Y
2
2
Save the dimension where matchRange was found a subset from SearchRange as a dict value with key equal to matchRange
2
3
Find missing match dimensions by looping on previously generated dict keys and looking for subset matches in the same manner done before. Add new found dimensions to the corresponding range.
call "findRunEnds" procedure on found matched dimensions, representing channel spans in the opposite orientation. (if 'V' channels are dict keys, 'H' spans are the dict values)
call "cleanUpOutOfShapeChannels" procedure on Channels dict.
Orientation loop finished?
N
1
Y
call "handleCrossOverlaps" procedure on Channels dict.
call "handleOverlaps" procedure on Channels dict for both orientations. Return final Channels dict
End

FIGURE 3.4: getLocations Procedure Flowchart

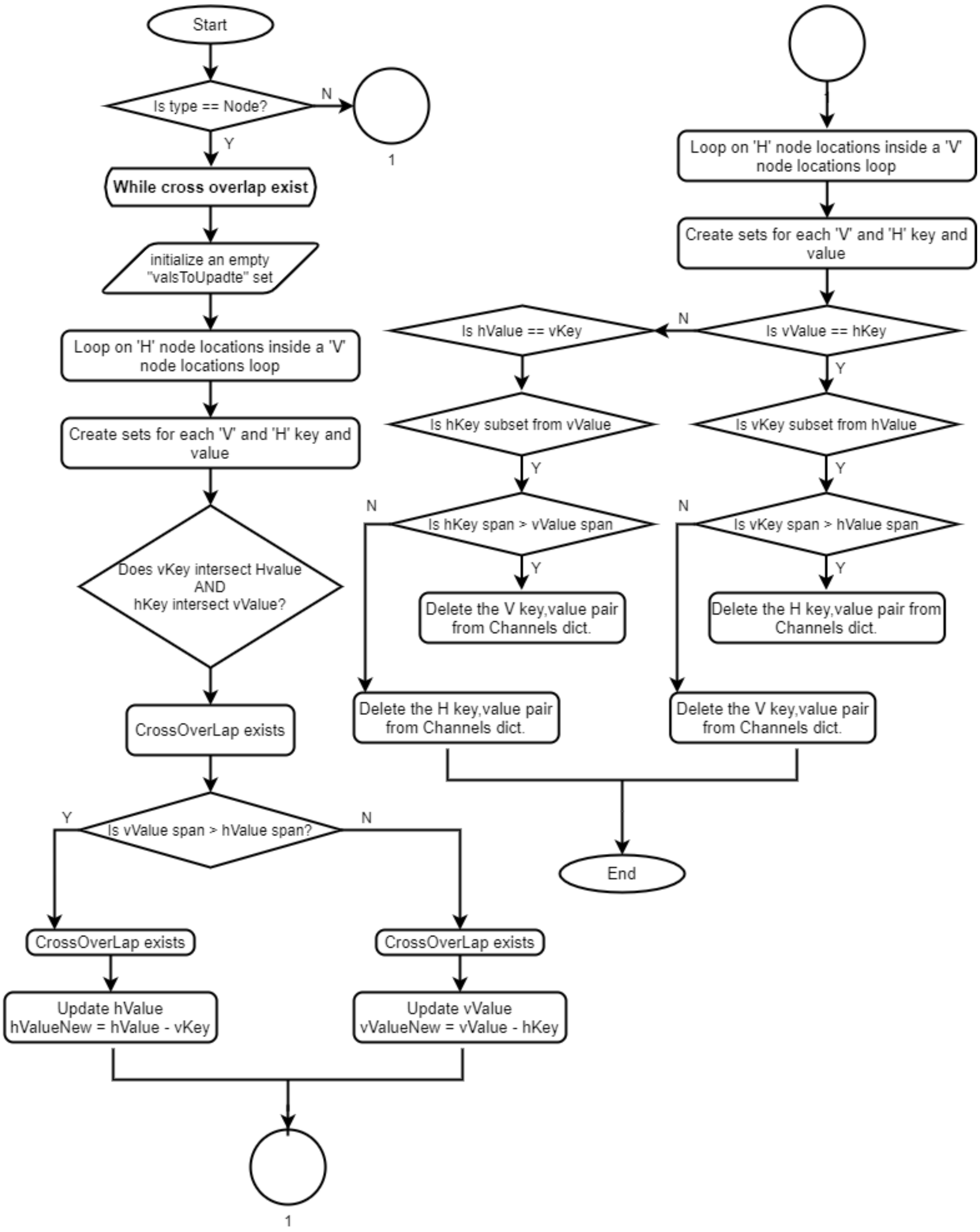

Start
Is type == Node?
N
Y
1
While cross overlap exist
initialize an empty "valsToUpadte" set
Loop on 'H' node locations inside a 'V' node locations loop
Create sets for each 'V' and 'H' key and value
Does vKey intersect Hvalue AND hKey intersect vValue?
CrossOverLap exists
Is vValue span > hValue span?
Y
N
CrossOverLap exists
CrossOverLap exists
Update hValue hValueNew = hValue - vKey
Update vValue vValueNew = vValue - hKey
1
Loop on 'H' node locations inside a 'V' node locations loop
Create sets for each 'V' and 'H' key and value
Is vValue == hKey
N
Is hValue == vKey
Y
Is hKey subset from vValue
Is vKey subset from hValue
Y
Y
N
Is hKey span > vValue span
N
Is vKey span > hValue span
Y
Y
Delete the V key,value pair from Channels dict.
Delete the H key,value pair from Channels dict.
Delete the H key,value pair from Channels dict.
Delete the V key,value pair from Channels dict.
End

FIGURE 3.5: handleCrossOverlaps Procedure Flowchart

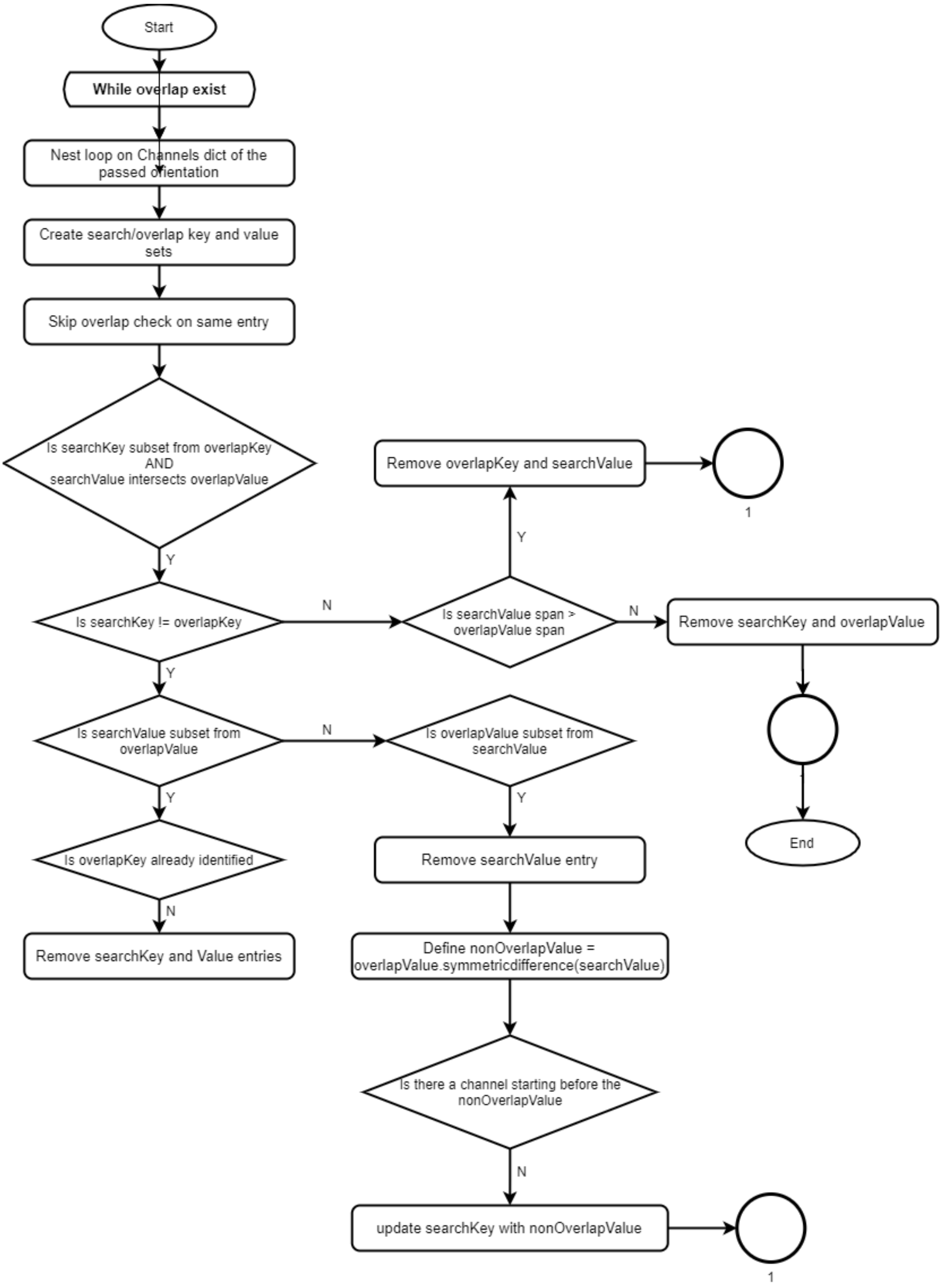

Start
While overlap exist
Nest loop on Channels dict of the passed orientation
Create search/overlap key and value sets
Skip overlap check on same entry
Is searchKey subset from overlapKey AND searchValue intersects overlapValue
Y
Is searchKey != overlapKey
N
Is searchValue span > overlapValue span
Y
Remove overlapKey and searchValue
1
N
Remove searchKey and overlapValue
Y
Is searchValue subset from overlapValue
N
Is overlapValue subset from searchValue
Y
Is overlapKey already identified
N
Remove searchKey and Value entries
Y
Remove searchValue entry
Define nonOverlapValue = overlapValue.symmetricdifference(searchValue)
Is there a channel starting before the nonOverlapValue
N
update searchKey with nonOverlapValue
1
End

FIGURE 3.6: handleOverlaps Procedure Flowchart

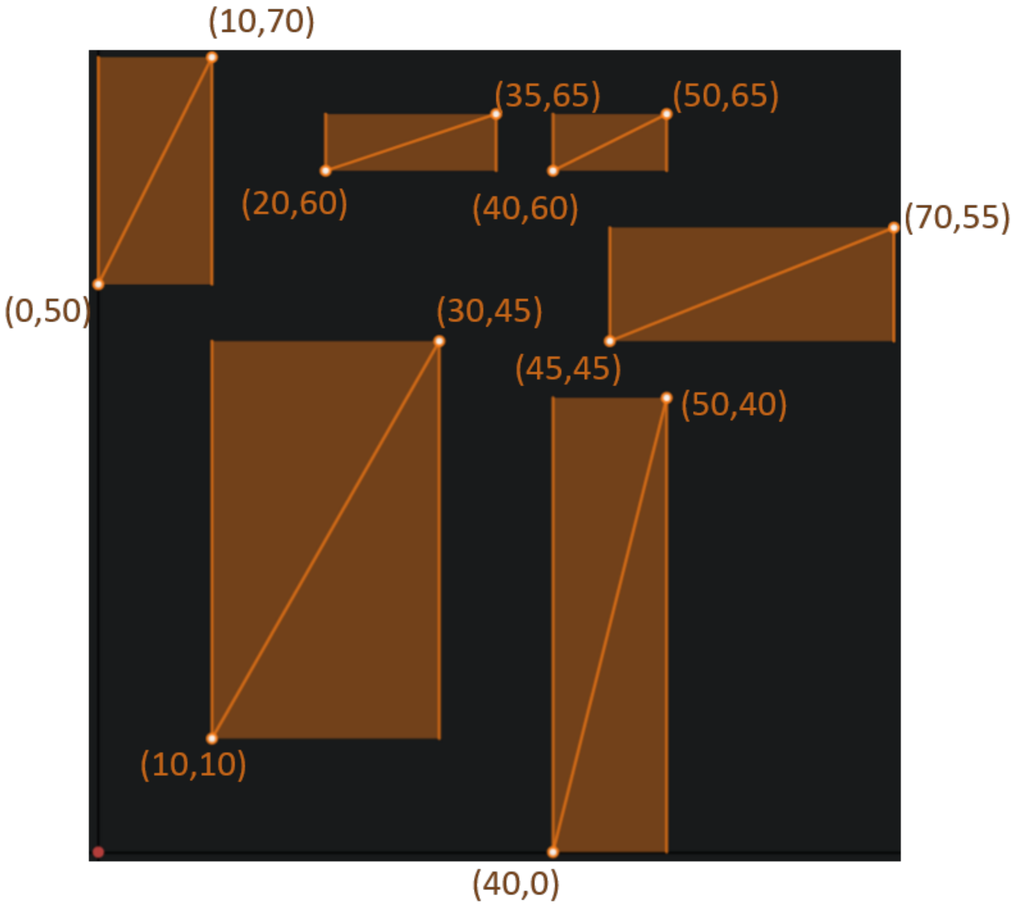


FIGURE 3.7: Channel Estimator Algorithm Visualized - Starting Point

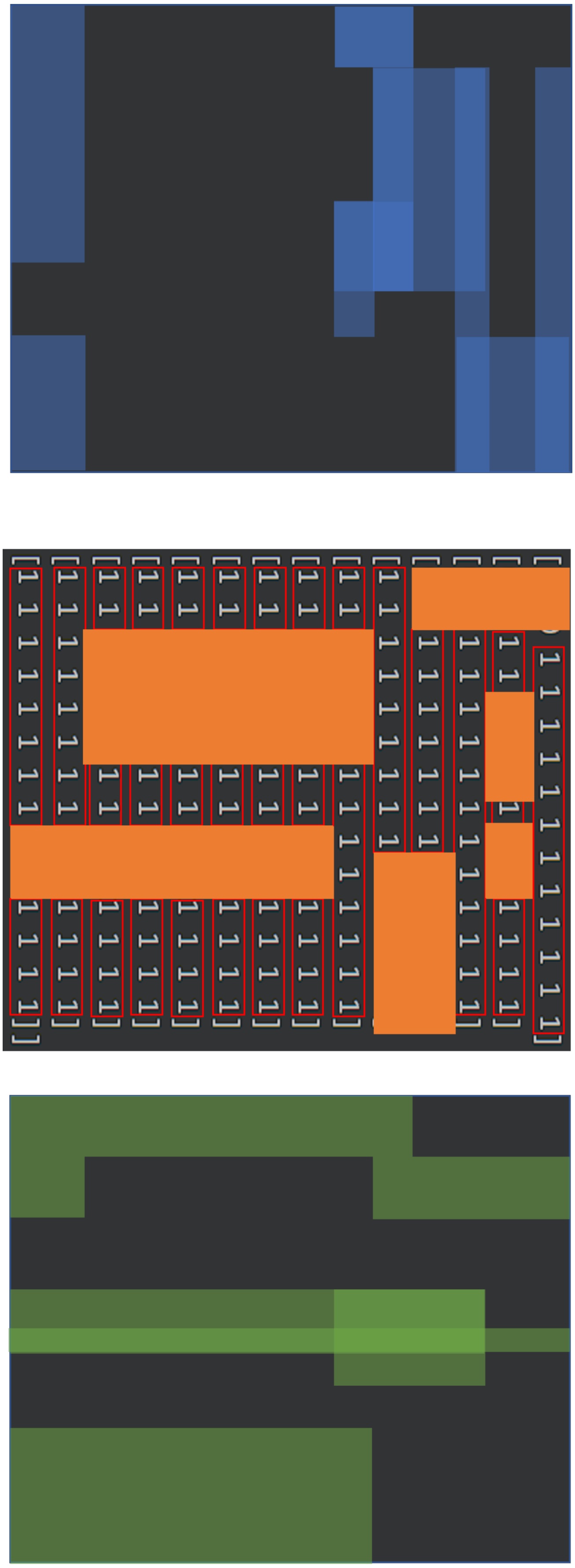


FIGURE 3.8: Channel Estimator Algorithm Visualized - Initial Channel Locations - Overlaps exist

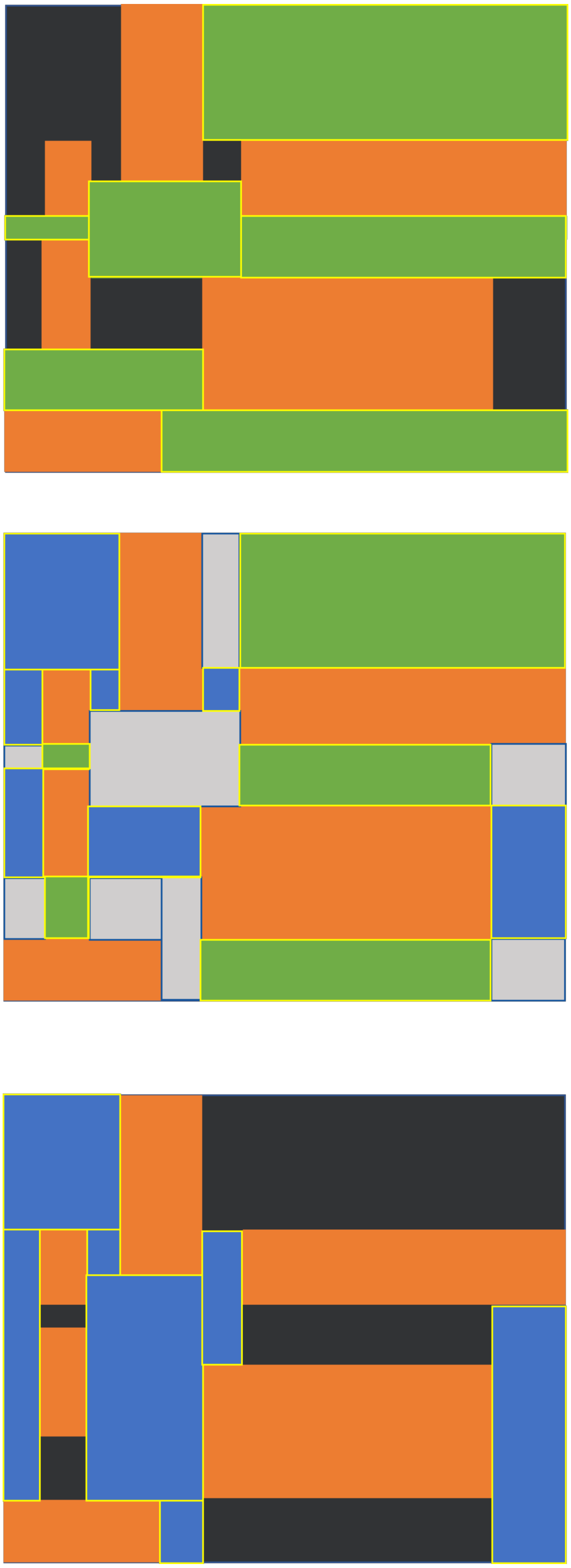


FIGURE 3.9: Channel Estimator Algorithm Visualized - Final Channel Locations - Overlaps resolved and nodes identified

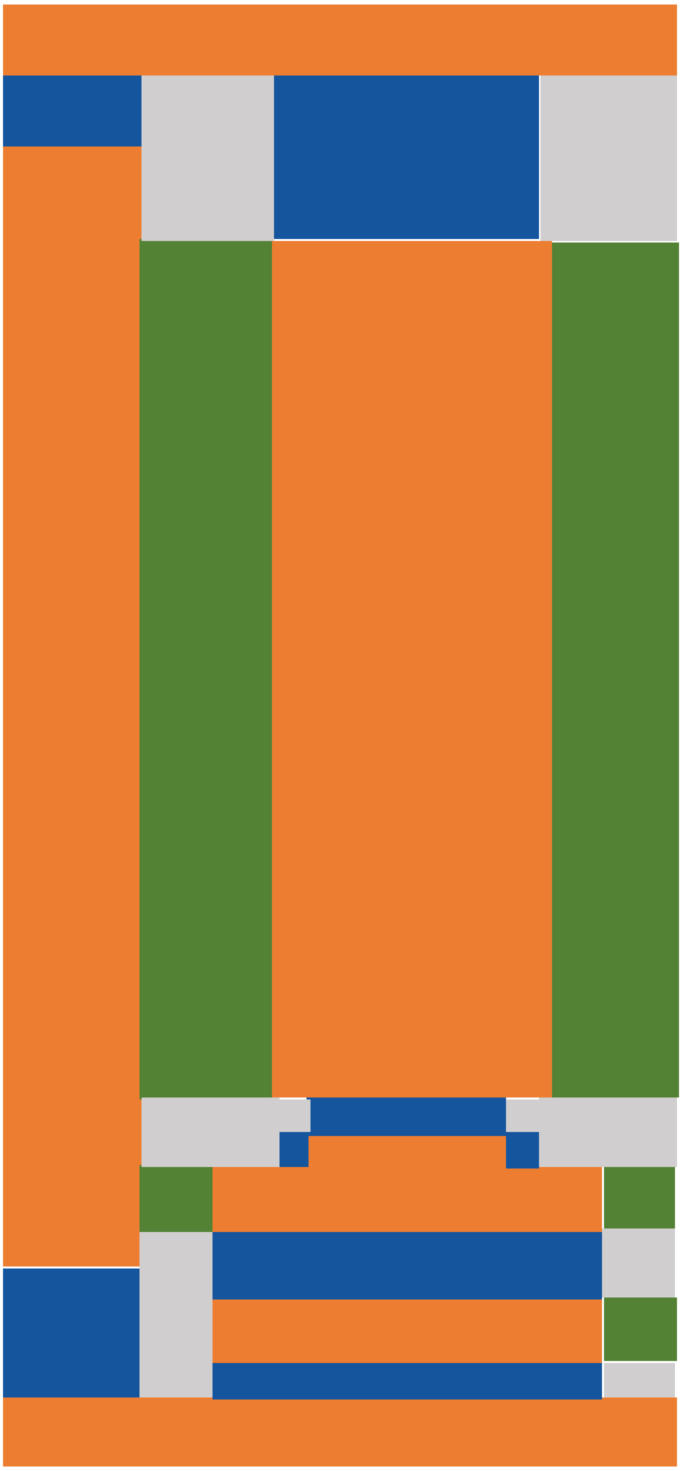


FIGURE 3.10: Channel Estimator Algorithm Visualized - Extended Usage Example

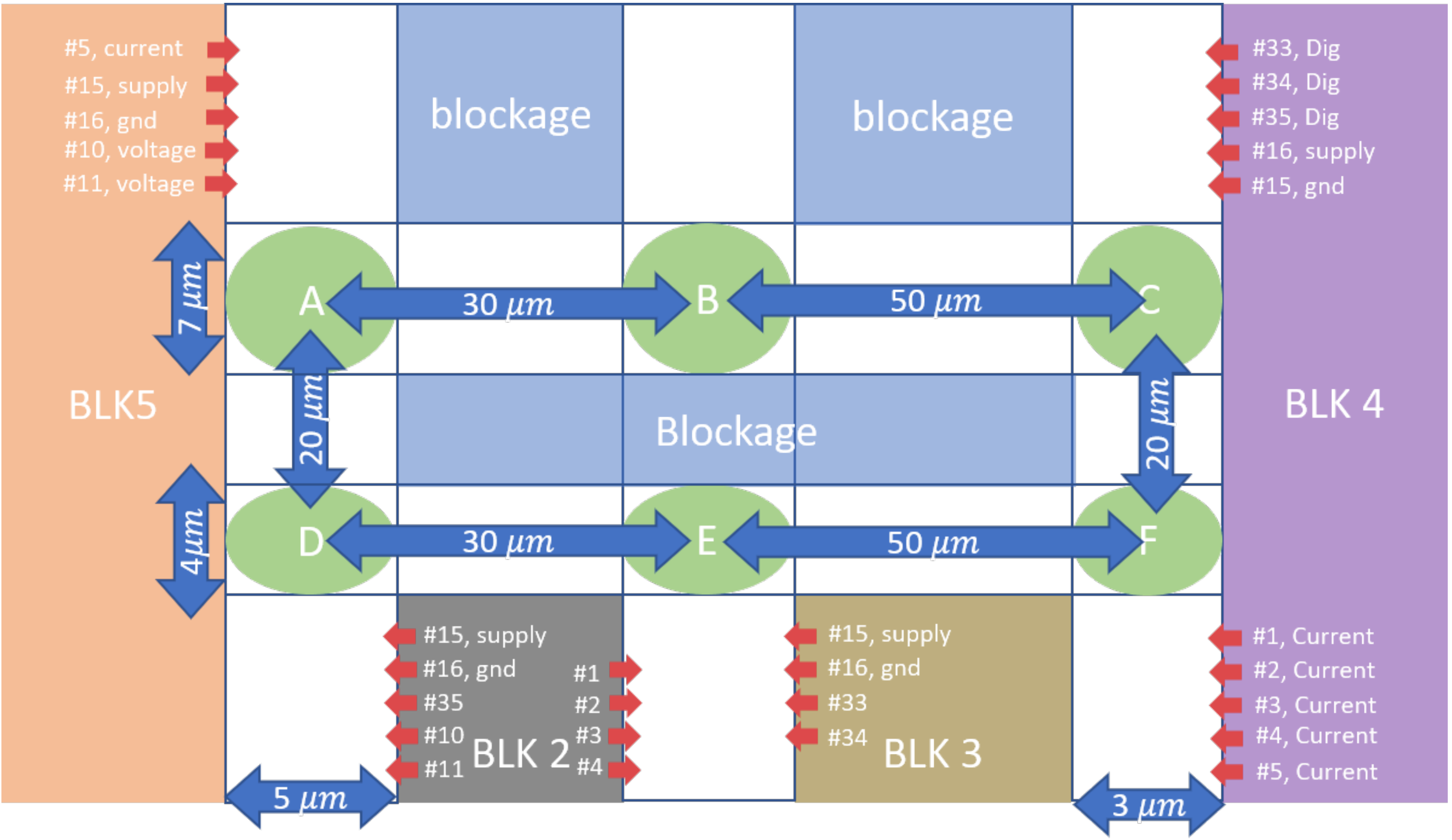


FIGURE 3.11: Channel Estimator Algorithm Visualized - Simple Example

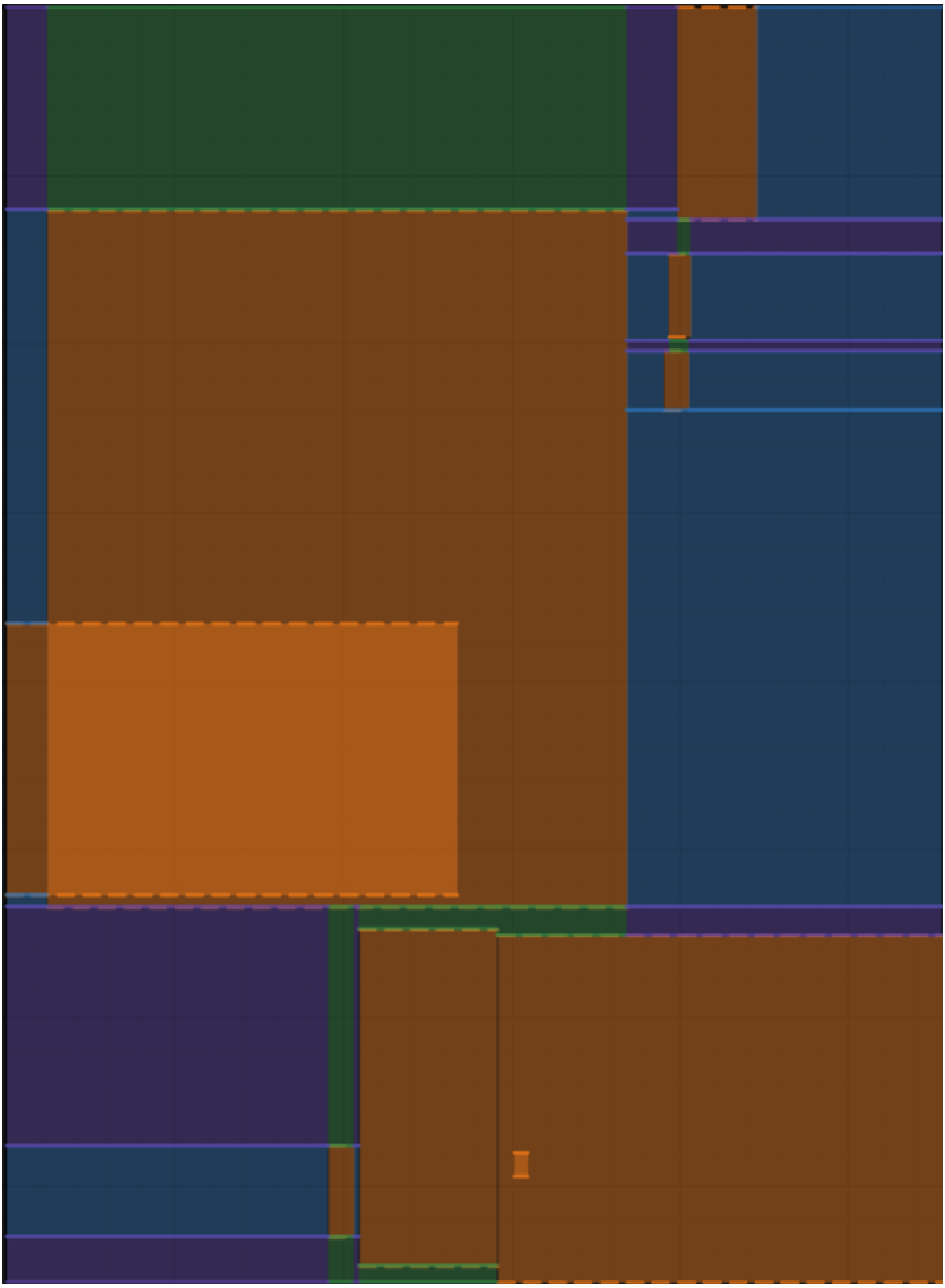


FIGURE 3.12: Channel Estimator Algorithm Visualized - Si-Vision Provided Example

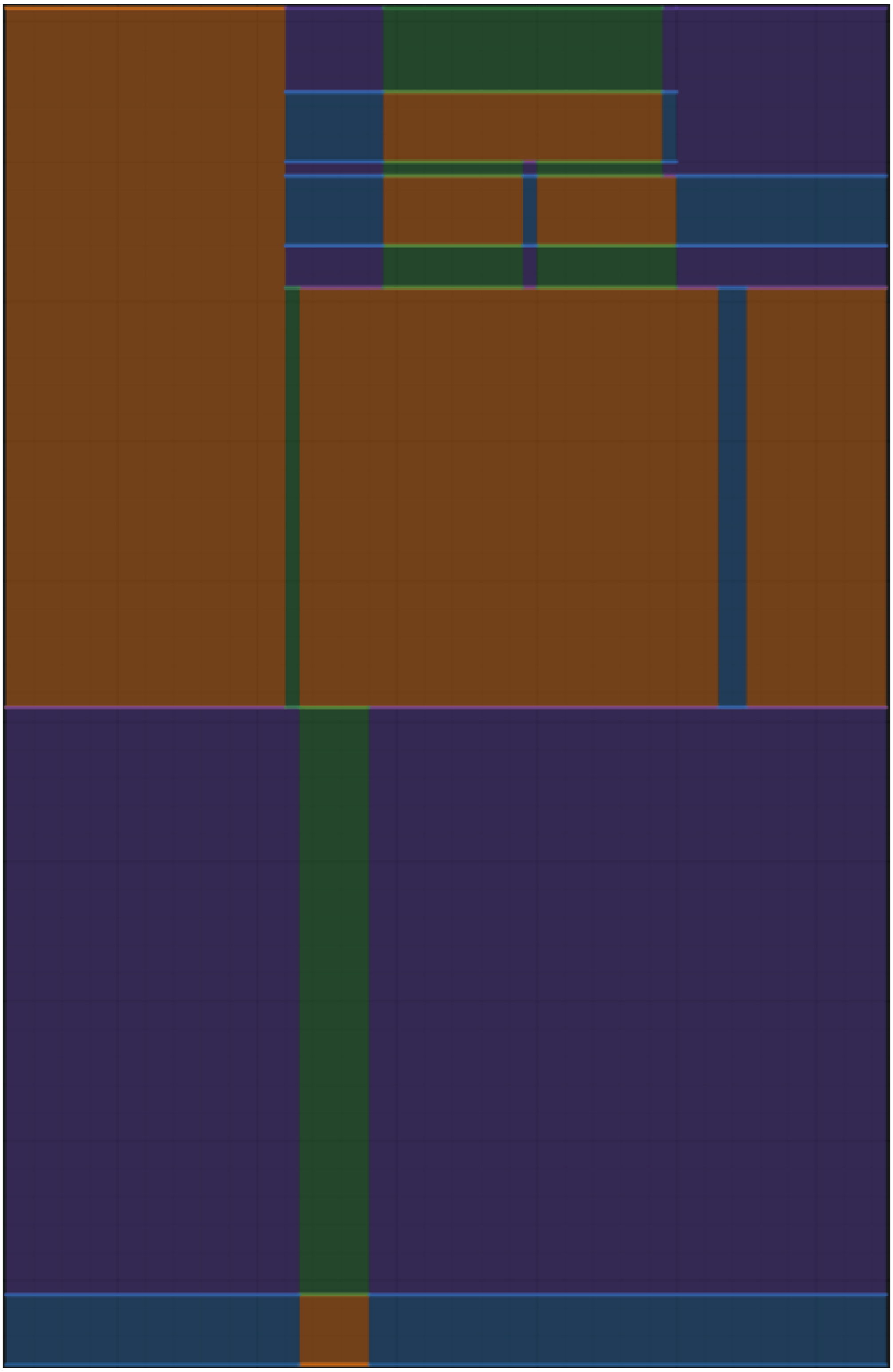
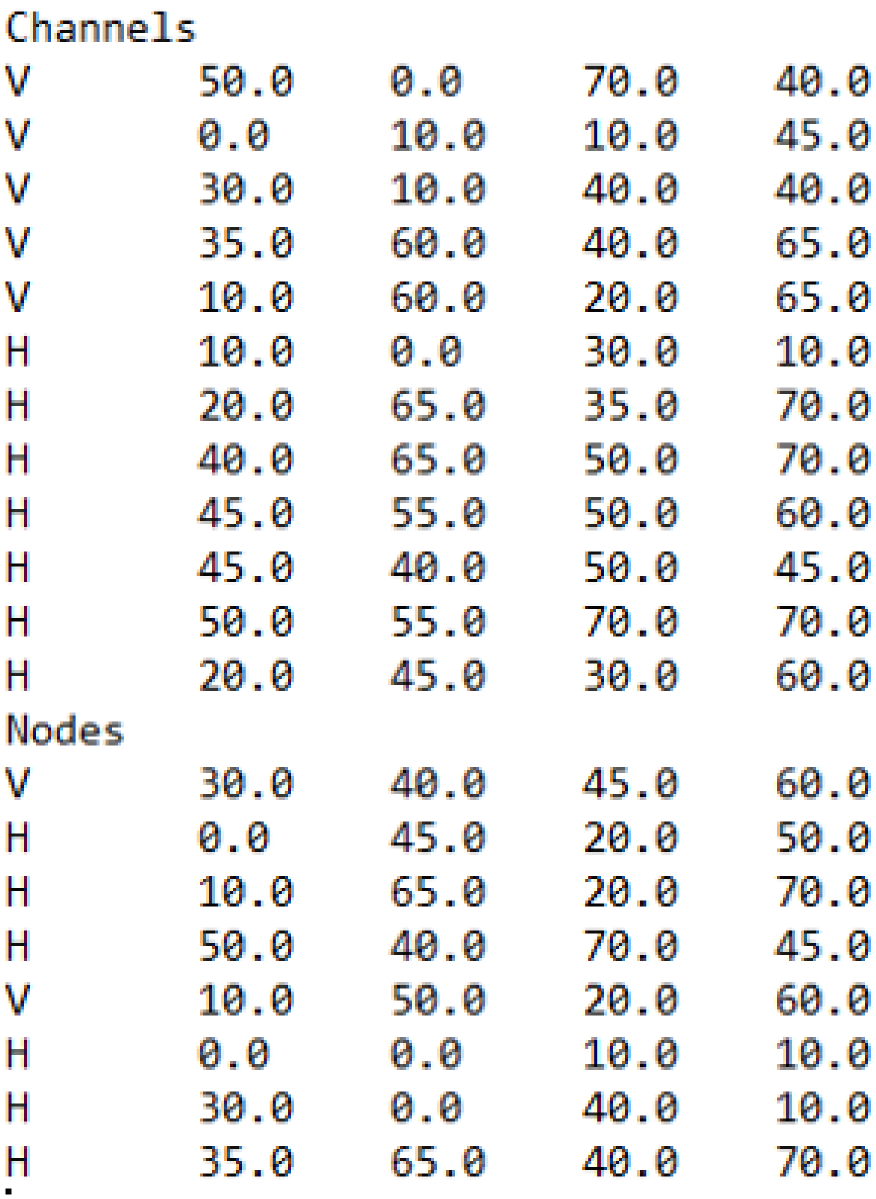


FIGURE 3.13: Channel Estimator Algorithm Visualized - Ain Shams Uni. Provided Example



```
Channels
V       50.0    0.0     70.0    40.0
V       0.0     10.0    10.0    45.0
V       30.0    10.0    40.0    40.0
V       35.0    60.0    40.0    65.0
V       10.0    60.0    20.0    65.0
H       10.0    0.0     30.0    10.0
H       20.0    65.0    35.0    70.0
H       40.0    65.0    50.0    70.0
H       45.0    55.0    50.0    60.0
H       45.0    40.0    50.0    45.0
H       50.0    55.0    70.0    70.0
H       20.0    45.0    30.0    60.0
Nodes
V       30.0    40.0    45.0    60.0
H       0.0     45.0    20.0    50.0
H       10.0    65.0    20.0    70.0
H       50.0    40.0    70.0    45.0
V       10.0    50.0    20.0    60.0
H       0.0     0.0     10.0    10.0
H       30.0    0.0     40.0    10.0
H       35.0    65.0    40.0    70.0
```

FIGURE 3.14: Channel Estimator Algorithm - OutputFormat

3.4 Node Builder

Now that we have the channel Bbox, orientation and the node Bbox from the channel estimator block, and the pin numbers and positions from the floor planner block, what expected from that block is to give the following inputs to its succeeding modules:

- The shortest Path finder:
  - Channel length, width and orientation between each node pair.
  - Pins associated to each node.
- The detailed router:
  - Channel length, width and orientation between each node pair.
  - Pins associated to which interface of the node/channel.
  - Pins coordinates on each interface.

All the node builder does is to virtually allocate the pins positions on the node interface .So first of all according to the channel orientation we start by allocating the pins on the corresponding channel interface if vertical the pins will be allocated right and left ,if horizontal the pins will be allocated to the top or to the bottom so the channel orientation determines which coordinates will be compared. In both cases ,the X and Y range should be checked to correctly allocate the channel. After allocating the pins on the channel interfaces ,for the general case where the channel is surrounded by nodes from the left ,right or from top ,down ,the distance from the pin to the node is calculated the minimum distance is chosen and then the pin is assigned to the nearest interface .For the special case where the channel has a node to the top or at the bottom , the pins are dragged to the corresponding interface immediately. The following example in Fig 3.15.
And the Final output file is giving in this text file fomat in Fig 3.16.

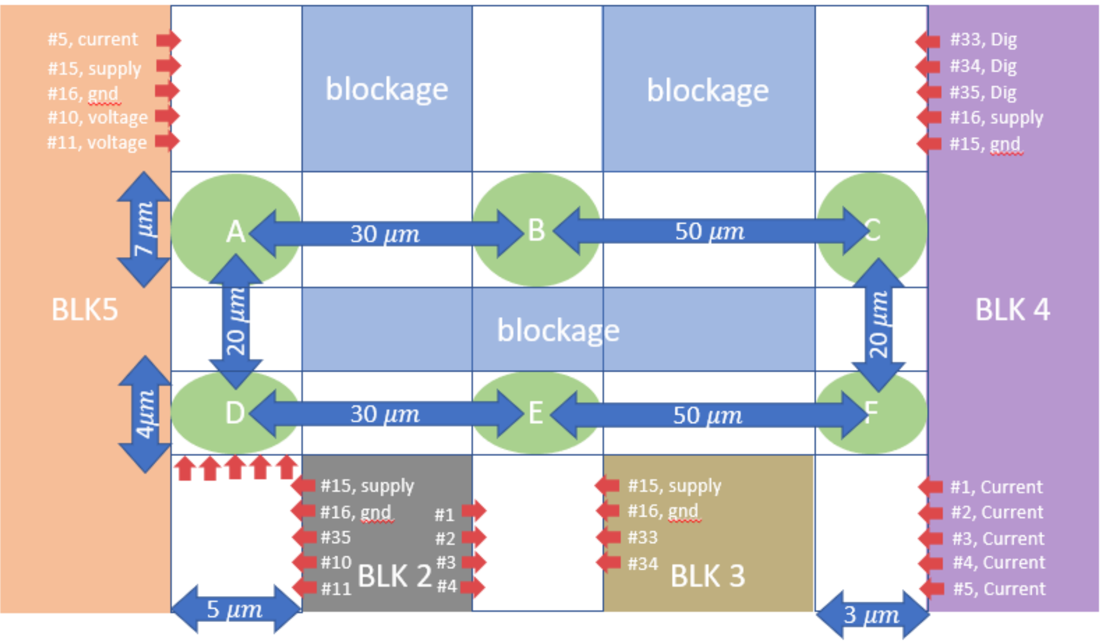
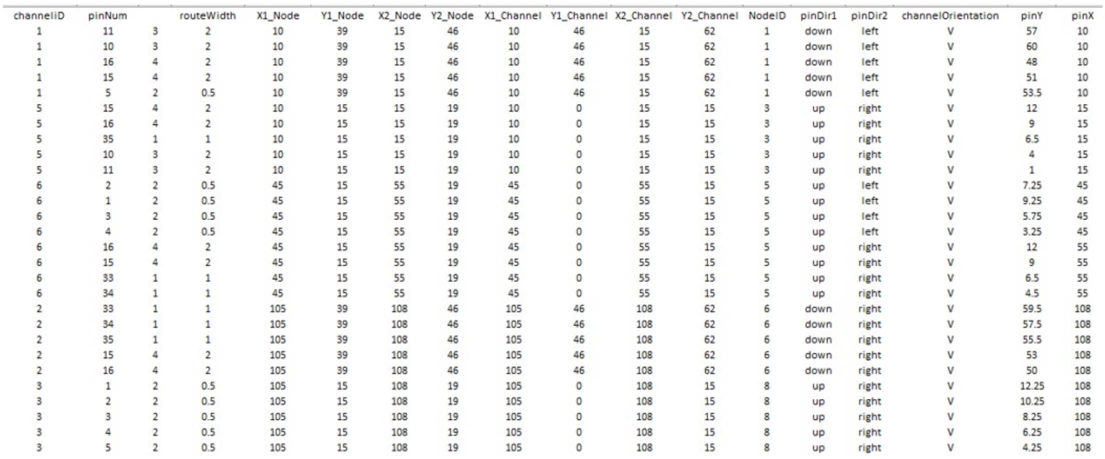


FIGURE 3.15: Pins allocated to node D interface



channeliD	pinNum		routeWidth	X1_Node	Y1_Node	X2_Node	Y2_Node	X1_Channel	Y1_Channel	X2_Channel	Y2_Channel	NodeID	pinDir1	pinDir2	channelOrientation	pinY	pinX
1	11	3	2	10	39	15	46	10	46	15	62	1	down	left	V	57	10
1	10	3	2	10	39	15	46	10	46	15	62	1	down	left	V	60	10
1	16	4	2	10	39	15	46	10	46	15	62	1	down	left	V	48	10
1	15	4	2	10	39	15	46	10	46	15	62	1	down	left	V	51	10
1	5	2	0.5	10	39	15	46	10	46	15	62	1	down	left	V	53.5	10
5	15	4	2	10	15	15	19	10	0	15	15	3	up	right	V	12	15
5	16	4	2	10	15	15	19	10	0	15	15	3	up	right	V	9	15
5	35	1	1	10	15	15	19	10	0	15	15	3	up	right	V	6.5	15
5	10	3	2	10	15	15	19	10	0	15	15	3	up	right	V	4	15
5	11	3	2	10	15	15	19	10	0	15	15	3	up	right	V	1	15
6	2	2	0.5	45	15	55	19	45	0	55	15	5	up	left	V	7.25	45
6	1	2	0.5	45	15	55	19	45	0	55	15	5	up	left	V	9.25	45
6	3	2	0.5	45	15	55	19	45	0	55	15	5	up	left	V	5.75	45
6	4	2	0.5	45	15	55	19	45	0	55	15	5	up	left	V	3.25	45
6	16	4	2	45	15	55	19	45	0	55	15	5	up	right	V	12	55
6	15	4	2	45	15	55	19	45	0	55	15	5	up	right	V	9	55
6	33	1	1	45	15	55	19	45	0	55	15	5	up	right	V	6.5	55
6	34	1	1	45	15	55	19	45	0	55	15	5	up	right	V	4.5	55
2	33	1	1	105	39	108	46	105	46	108	62	6	down	right	V	59.5	108
2	34	1	1	105	39	108	46	105	46	108	62	6	down	right	V	57.5	108
2	35	1	1	105	39	108	46	105	46	108	62	6	down	right	V	55.5	108
2	15	4	2	105	39	108	46	105	46	108	62	6	down	right	V	53	108
2	16	4	2	105	39	108	46	105	46	108	62	6	down	right	V	50	108
3	1	2	0.5	105	15	108	19	105	0	108	15	8	up	right	V	12.25	108
3	2	2	0.5	105	15	108	19	105	0	108	15	8	up	right	V	10.25	108
3	3	2	0.5	105	15	108	19	105	0	108	15	8	up	right	V	8.25	108
3	4	2	0.5	105	15	108	19	105	0	108	15	8	up	right	V	6.25	108
3	5	2	0.5	105	15	108	19	105	0	108	15	8	up	right	V	4.25	108

FIGURE 3.16: Final Nodebuilder output

3.5 Shortest path finder

The shortest path finder module is the heart of the tool. The algorithm enables the whole flow to accomplish its hard task of implementing the analog layout designer′s way of thinking into a procedure.

The single most important constraint on analog layout routing in general is minimizing the parasitics. Other constraints include matching between sensitive nets as well as shielding them from noisy signals.

Analog routers face several challenges that separate them from their digital counter part. Variable widths and the need to assign metal layers to specific net types and define an order by which those nets get routed are the most critical of the aforementioned challenges.

This leads us to defining the objectives of the shortest path finder algorithm.

- Perform virtual routing. i.e. Allocation
- Maintaining minimum parasitics and minimum wire loads. i.e. shortest path by means of modified Dijkstra shortest path algorithm.
- Define an order by which nets are virtually routed. [Supply, Voltage, Current, Digital Control]
- Follow Manhattan routing scheme. [Even – Odd metal layers]
- Assign metal layers to net types in the following order.
    1. Supply net are assigned to M6 and M7.
    2. Sensitive nets(Voltage) are assigned to M4, M5 and rest of M6.
    3. Noncritical nets (Current) are assigned to rest of M4, M5 and both M2, M3.
    4. Noisy nets (Digital Controls) are assigned to rest of M2 and M1.

The reasons behind why the algorithm employs Dijkstra shortest path algorithm is due to several facts. Firs of all, it is classified as a proactive routing algorithm, meaning it is always updated with the status of the graph. It is table based i.e. all nodes know the path to all other nodes in the graph, and with the proper implementation (Python dictionaries) table access is simple and fast. And finally Dijkstra is a well established algorithm making it easy to find an already implemented and optimized version.

The algorithm faced several challenges that needed addressing. The fact that Dijkstra can only deal with 2D graphs and the algorithm requirement of representing the metal stack in the design created the need for adjusting the graph implementation to be 3D. Moreover, Dijkstra fails when dealing with negative weight edges, this case can easily happen as more routes get allocated to the same metal layer in any channel, eventually leaving no space in the metal layer for further allocation i.e. channel width in a specific metal layer equals to zero.

By proper conditioning both the main algorithm and Dijkstra procedure this implementation was able to overcome these limitations. As mentioned, the algorithm implements the graph data structure in a way such that it enables representing the metal stack, overcoming the first limitation.

Fig 3.17 explains the conditioning employed to overcome the second limitation and enable proper allocation process. Channels are checked if available during the allocation process. In addition to that, the algorithm makes sure that the assigned metal layer has enough space to accommodate the net in turn.

In case the "sufficient width" condition fails, metal layer is made temporarily unavailable. Dijkstra is then called and routing table is updated with a new path. Dijkstra procedure implementation is modified such that even if the assigned metal layer has insufficient width or became unavailable mid-allocation, it allows for a single lower metal layer dive of the net maintaining shortest path and consequently minimum parasitics.

Nodes are removed from graph if three conditions are true.

1. Node exists in graph.
2. All associated nets are virtually routed successfully.
3. All connected metal layers to the node are unavailable.

The algorithm fails upon satisfying three conditions.

1. Node exists in graph.
2. NOT All associated nets are virtually routed successfully.
3. All connected metal layers to the node are unavailable.

In Fig 3.18 the algorithm's output format is shown. it follows the following structure column wise:

1. Channel Identifier.
2. Net number.
3. Net width.
4. Assigned metal layer.
5. Columns 5 through 8 represent the channel Bbox.
6. Channel orientation.

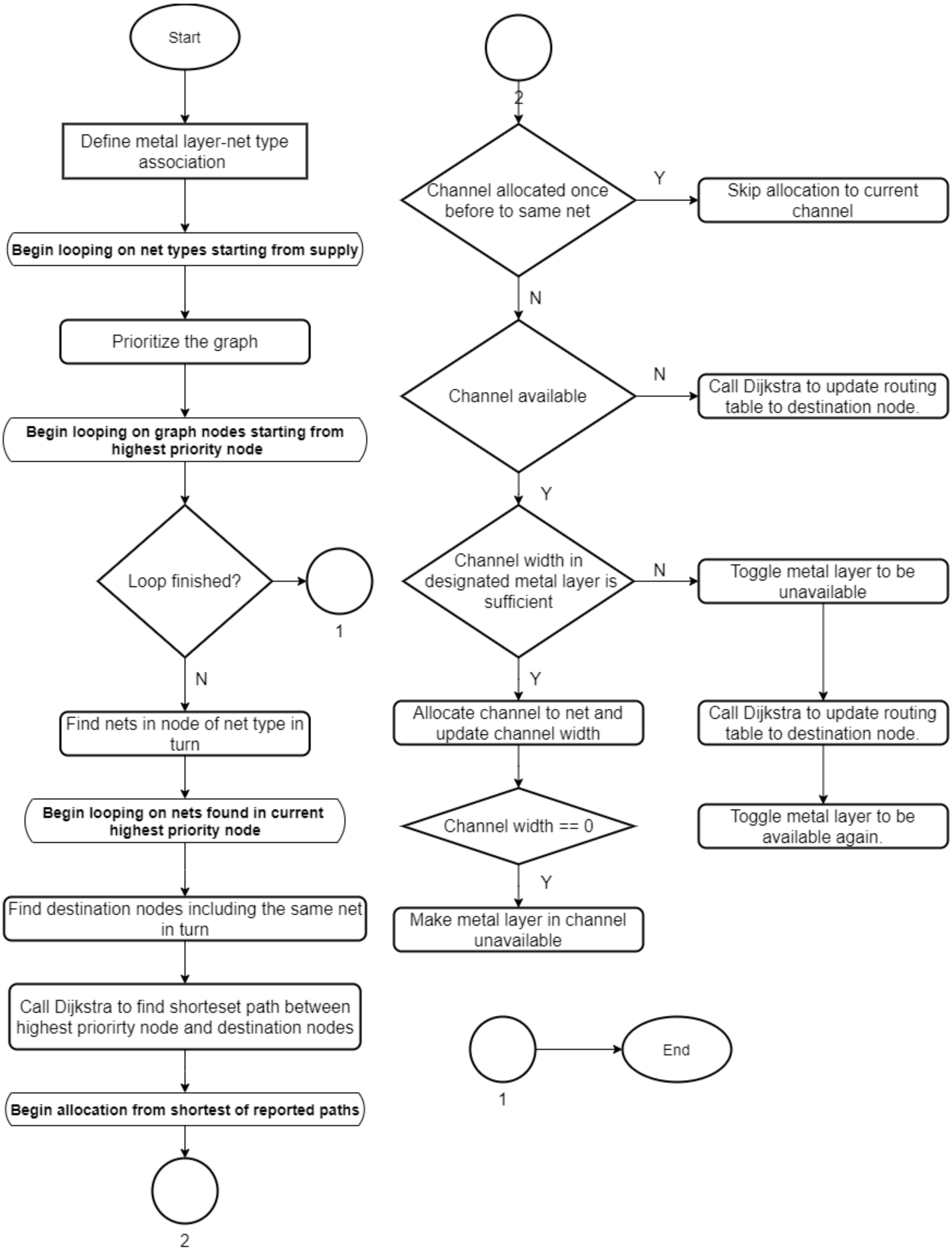
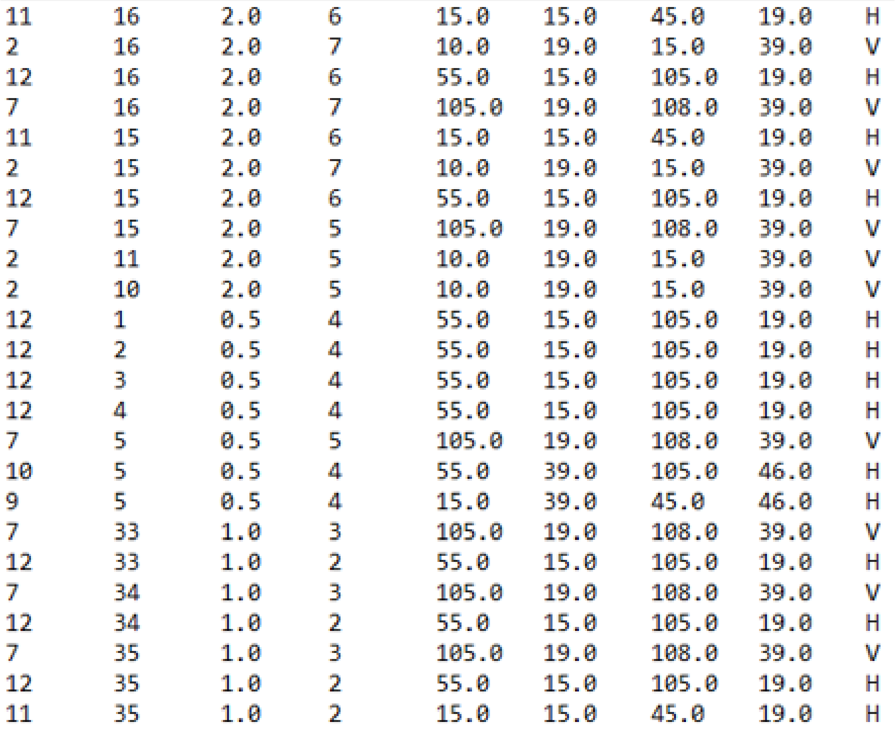

Start
Define metal layer-net type association
Begin looping on net types starting from supply
Prioritize the graph
Begin looping on graph nodes starting from highest priority node
Loop finished?
1
N
Find nets in node of net type in turn
Begin looping on nets found in current highest priority node
Find destination nodes including the same net in turn
Call Dijkstra to find shorteset path between highest priorirty node and destination nodes
Begin allocation from shortest of reported paths
2
2
Channel allocated once before to same net
Y
Skip allocation to current channel
N
Channel available
N
Call Dijkstra to update routing table to destination node.
Y
Channel width in designated metal layer is sufficient
N
Toggle metal layer to be unavailable
Y
Allocate channel to net and update channel width
Call Dijkstra to update routing table to destination node.
Channel width == 0
Toggle metal layer to be available again.
Y
Make metal layer in channel unavailable
1
End

FIGURE 3.17: Shortest Path Finder Flowchart



```
11      16      2.0     6       15.0    15.0    45.0    19.0    H
2       16      2.0     7       10.0    19.0    15.0    39.0    V
12      16      2.0     6       55.0    15.0    105.0   19.0    H
7       16      2.0     7       105.0   19.0    108.0   39.0    V
11      15      2.0     6       15.0    15.0    45.0    19.0    H
2       15      2.0     7       10.0    19.0    15.0    39.0    V
12      15      2.0     6       55.0    15.0    105.0   19.0    H
7       15      2.0     5       105.0   19.0    108.0   39.0    V
2       11      2.0     5       10.0    19.0    15.0    39.0    V
2       10      2.0     5       10.0    19.0    15.0    39.0    V
12      1       0.5     4       55.0    15.0    105.0   19.0    H
12      2       0.5     4       55.0    15.0    105.0   19.0    H
12      3       0.5     4       55.0    15.0    105.0   19.0    H
12      4       0.5     4       55.0    15.0    105.0   19.0    H
7       5       0.5     5       105.0   19.0    108.0   39.0    V
10      5       0.5     4       55.0    39.0    105.0   46.0    H
9       5       0.5     4       15.0    39.0    45.0    46.0    H
7       33      1.0     3       105.0   19.0    108.0   39.0    V
12      33      1.0     2       55.0    15.0    105.0   19.0    H
7       34      1.0     3       105.0   19.0    108.0   39.0    V
12      34      1.0     2       55.0    15.0    105.0   19.0    H
7       35      1.0     3       105.0   19.0    108.0   39.0    V
12      35      1.0     2       55.0    15.0    105.0   19.0    H
11      35      1.0     2       15.0    15.0    45.0    19.0    H
```

FIGURE 3.18: Shortest Path Finder - Output Format

3.6 Detailed Router

As mentioned before the routing problem is usually solved by using a two-stage approach of global routing followed by detailed routing due to its complexity. In this section the proposed detailed routing algorithm will be presented for given channel.

This block job is to get the processed floor-plan from the previous blocks and do the actual route placement with physical routes, its desired output is a physical route to be placed in the tool by the appropriate scripting language (ex:TCL for synopsys tools).
The detailed router consists of two main parts Channel router and Node router (switch box).

The channel router: was made in order to connect the nodes with the pins and route the pass-through routes in each channel the main difference between the channel router and the node builder that the node have 4 interfaces and need to connect the pins on those interfaces in general case while the channel routing having only two interfaces to route the node router much more complex and most of the found algorithms are gird based which will have constant width which is not desired in analog layout ,A grid-less algorithm is needed and this will be a part of the future work

- **Required Inputs:**
  - Pins and their connected Nodes(Node Builder).
  - Channel Coordinates ,length and it's orientation(Channel Estimator).
  - Connected pins on each channel and their metal pairs(Shortest-path finder).
- **Required Output:**
  - TCL script for route placement.

The Algorithm:

- For each side
  - For each pin
    - Start by the nearest unconnected to its node interface.
    - Extend Horizontal route using its width with the specified Horizontal Metal.
    - Extend Vertical route to the Node with the same width spaced by DRC distance from the starting point.
    - Update the starting point.
  - For each of the pass-through Routes

    * Move DRC distance from the last left Vertical Metal and place a vertical route with the channel length
    * Update last left Vertical space
  - Do axes Transmission to move the routes to the real channel origin rotation if it was a horizontal one.

In order to understand the algorithm it's better to take a visual Example in Fig. 3.19 we start by taking the left pins to know which pin will go to which node in this example we only have one node so the node builder will direct all the pins to (Node A) starting by pin 1 a horizontal route extended from the node with the required width.

The node builder the used algorithm was implemented from the greedy algorithm in [16] and then modified such that the width is variable the main problem that this algorithm is grid based which needed some modifications and no covering the whole cases other algorithm was found more suitable for the node routing [17] in the tool but will be implemented in future work.

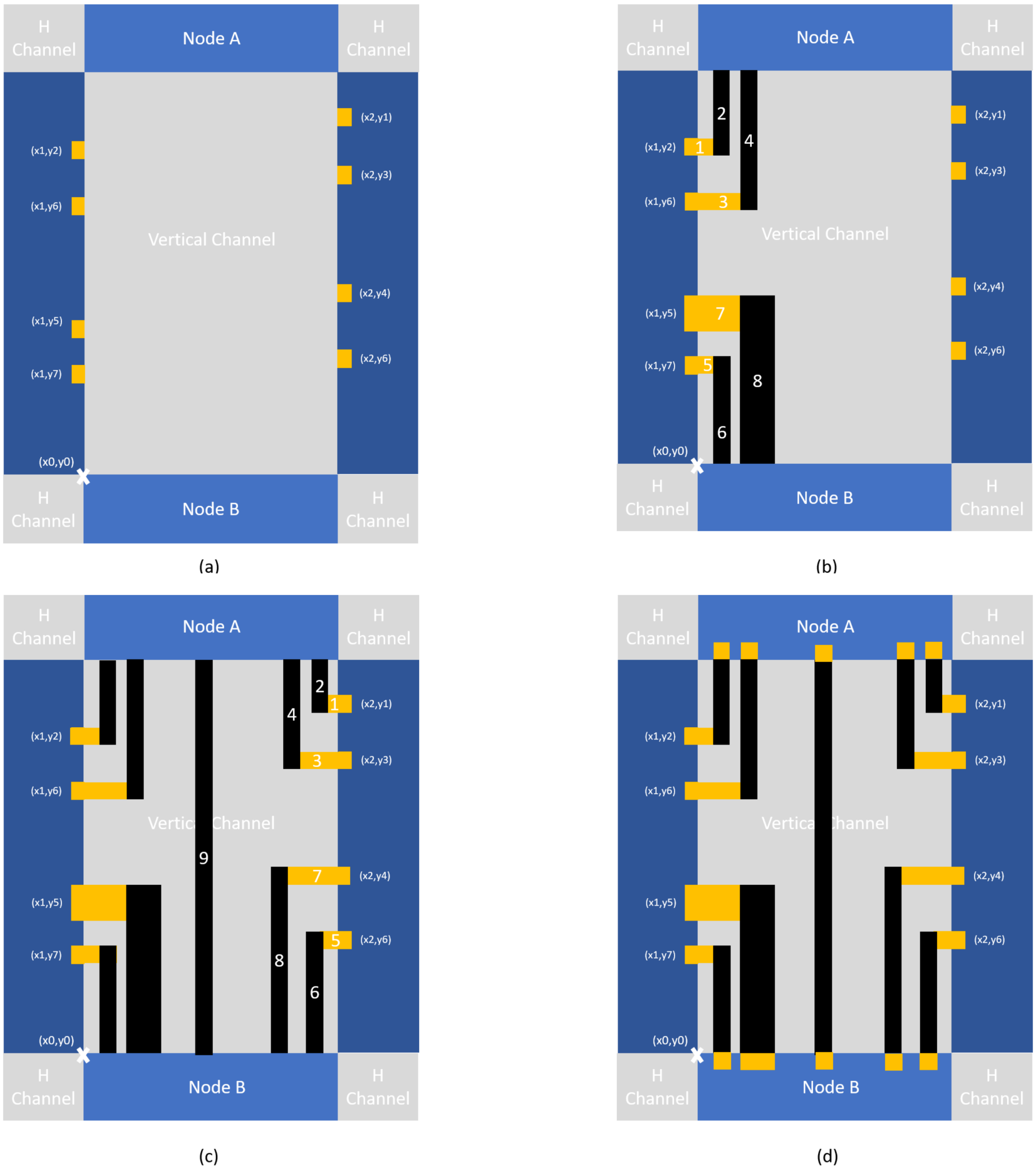


FIGURE 3.19: Illustrated example a) the channel with the pins before routing
b) starting from the left side step by step drawing a horizontal route then extending a vertical route to the desired node
c)routing the right side step by step.
d)The Final Routed channel

Chapter 4

Conclusion and Future Work

By applying the given example in Fig 4.1 to the tool the resulting output is giving in Fig 4.2 was verified and passed the DRC verification.

4.1 Conclusion

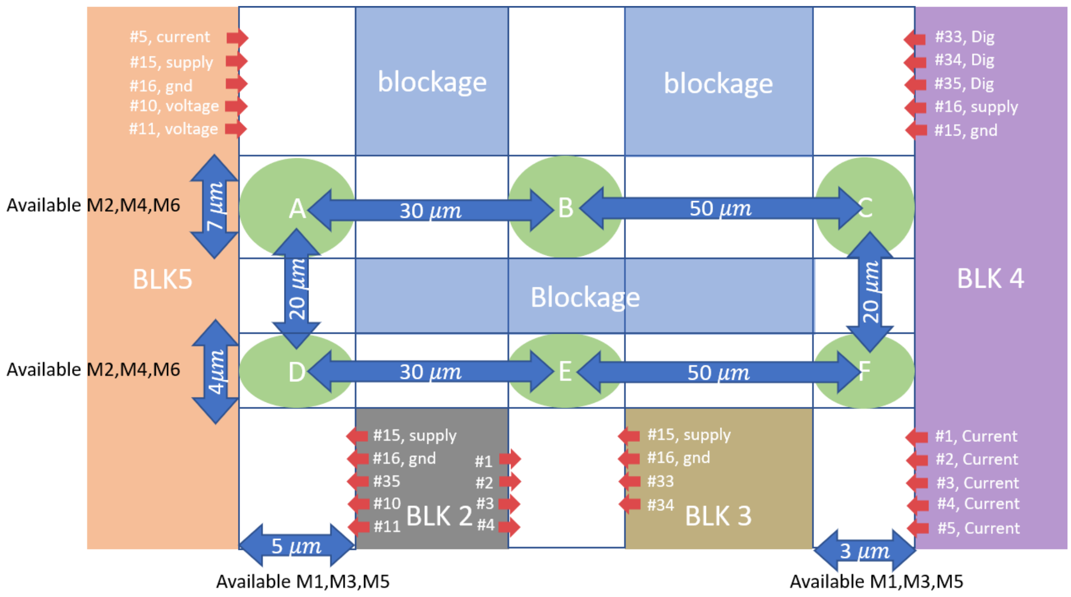
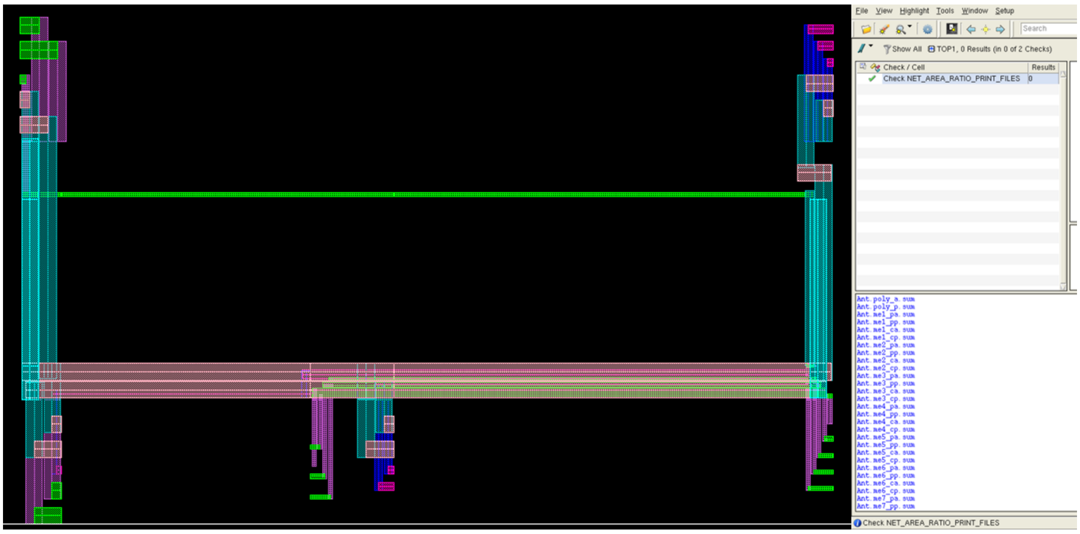


FIGURE 4.1: Numerical Example

The tool starts by taking the input floorplan and Cdl files. It then passes them to the signal identifier to determine the pins and the signals priority. The channel estimator takes the given floorplan and extract the channels and nodes giving also the channel length and it's orientation. The node builder then takes this output from the channel estimator and determine which pin goes to which node then passes it's output to the shortest pass finder and detailed router.

Finally after the shortest path finder finishes the virtual routing the actual routing is performed using the detailed router exporting a tcl script for the tool automatic placment in the desired tool as shown in Fig 4.2



TOP1, 0 Results (in 0 of 2 Checks)

Check / Cell	Results
✓ Check NET_AREA_RATIO_PRINT_FILES	0

```
Ant.poly_a.sum
Ant.poly_p.sum
Ant.me1_pa.sum
Ant.me1_pp.sum
Ant.me1_ca.sum
Ant.me1_cp.sum
Ant.me2_pa.sum
Ant.me2_pp.sum
Ant.me2_ca.sum
Ant.me2_cp.sum
Ant.me3_pa.sum
Ant.me3_pp.sum
Ant.me3_ca.sum
Ant.me3_cp.sum
Ant.me4_pa.sum
Ant.me4_pp.sum
Ant.me4_ca.sum
Ant.me4_cp.sum
Ant.me5_pa.sum
Ant.me5_pp.sum
Ant.me5_ca.sum
Ant.me5_cp.sum
Ant.me6_pa.sum
Ant.me6_pp.sum
Ant.me6_ca.sum
Ant.me6_cp.sum
Ant.me7_pa.sum
Ant.me7_pp.sum
```

Check NET_AREA_RATIO_PRINT_FILES

FIGURE 4.2: Final Output

4.2 Future Work

- Channel Estimator
  - Handle L shape blocks due to floor planning.
  - Solve the run time bottleneck for array sizes exceeding 500 × 500 element by optimizing the identified code block.
- Shortest Path Finder.
  - Incorporate differential net signals, and (Supply, GND) pairs identification.
  - Shielding considerations.
  - Allocate a longer path in the assigned metal layer OR Allocate a short path in the directly lower metal layer.
  - Density Rule considerations.
  - Add “No Star” connection feature.
- Detailed Router.
  - Node Router.
  - Handling similar pins on opposite interfaces of a channel.

List of References

[1] R. Martins, N. Lourenço, and N. Horta, “Laygen ii—automatic layout generation of analog integrated circuits”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1641–1654, 2013.

[2] H. Chi, H. Tseng, C. J. Liu, and H. Chen, “Performance-preserved analog routing methodology via wire load reduction”, in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 482–487.

[3] M. Torabi and L. Zhang, “Electromigration- and parasitic-aware ilp-based analog router”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1854–1867, 2018.

[4] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, “Invited: Align – open-source analog layout automation from the ground up”, in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4.

[5] Y. Chen, H. Chi, L. Song, C. J. Liu, and H. Chen, “A structure-based methodology for analog layout generation”, in *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019, pp. 33–36.

[6] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, “Magical: Toward fully automated analog ic layout leveraging human and machine intelligence: Invited paper”, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[7] C. Chu and Y. Wong, “Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[8] G. Dündar and A. Unutulmaz, “Routing analog circuits”, in *Analog Layout Synthesis: A Survey of Topological Approaches*, H. E. Graeb, Ed. Boston, MA: Springer US, 2011, pp. 149–201, ISBN: 978-1-4419-6932-3. DOI: 10.1007/978-1-4419-6932-3_4. [Online]. Available: https://doi.org/10.1007/978-1-4419-6932-3_4.

[9] B. Razavi, *Design of Analog CMOS Integrated Circuits*, first. McGraw-Hill, 2001.

[10] C. Saint and J. Saint, *IC Mask Design: Essential Layout Techniques*, ser. McGraw-Hill professional engineering. McGraw-Hill, 2002, ISBN: 9780071450478. [Online]. Available: https://books.google.com.eg/books?id=9ji3AQAACAAJ.

[11] F. Atef, M. H. Zaky, N. K. Ahmed, M. A. Messiha, O. T. Abdelwahab, A. A. Farid, O. M. Selim, and H. Mostafa, “Automated current mirror layout (acml) tool”, in *2019 31st International Conference on Microelectronics (ICM)*, 2019, pp. 182–185.

[12] S. A. Mohamed, M. Dessouky, F. A. Naguib, and S. Hamed, “Analog layout placement based on unit elements and routing channel estimation”, in *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019, pp. 29–32.

[13] L. Wang, S. Xiong, J. Yang, and J. Fan, “An improved elitist strategy multi-objective evolutionary algorithm”, in *2006 International Conference on Machine Learning and Cybernetics*, 2006, pp. 2315–2319.

[14] R. Martins, N. Lourenço, S. Rodrigues, J. Guilherme, and N. Horta, “Aida: Automated analog ic design flow from circuit level to layout”, in *2012 International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2012, pp. 29–32.

[15] R. Martins, N. Lourenço, and N. Horta, “Multi-objective multi-constraint routing of analog ics using a modified nsga-ii approach”, in *2012 International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2012, pp. 65–68.

[16] "https://github.com/searsm8/Channel_Routing", 2020.

[17] H. H. Chen and E. S. Kuh, “Glitter: A gridless variable-width channel router”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no. 4, pp. 459–465, 1986.