POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Study and development of design techniques for 3D integrated circuits

Supervisors

Prof. Luca STERPONE

Ph.D Sarah AZIMI

Candidate

Davide MASSIMINO

July 2021

Acknowledgements

I would first like to thank Professor Luca Sterpone, whose expertise and motivating passion have guided me into the development of the project. Your introduction to a such interesting topic stimulated my interest and pushed me to expand my cultural background, and for this I will always be grateful to you.

I would also like to express my thanks to Dr. Sarah Azimi who has followed my work step by step, always ready to answer my questions or clarify my doubts. Your insightful feedback prompted me to hone my knowledge and develop more critical thinking. You provided me with the tools I needed to successfully complete the project.

A special thanks goes to my family who have never made me miss anything in all my years of university. They have always been ready to listen to me in times of need and were the first to push me to give my best. I would not have gotten this far without them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last five decades was great for the semiconductor industry, the development of ever smaller component allows to riding high on the back of the device scaling and improve in a tremendous way the performance, power, and cost of the integrated circuits (ICs). From the late 1960s[1], thanks to the improvement on those metrics, the semiconductor industry products were in line with the Moore's law. Moore said that the number of transistors in a microprocessor would double roughly every 18 months and his law was used frequently by the main firms of the semiconductor sector to make prediction and focus their investment. Over the years many new device and interconnect innovation like High-k Metal Gate, strained silicon, FinFET and porous low-k dielectric were able to boost the performance and follow the increasing demand of the market. In the last few years, the focus on new technology like Artificial Intelligence (AI) and Internet of Things (IoT) create the need of even better and performing device that can manage and work on huge amount of data. Today we have microprocessors with transistors' size of 7nm (Apple A12 Bionic) that allow a very large-scale integration, but we are reaching a point in which the Moore's law break apart due to physical limitation. Transistors that reach the nano-scale enter in the world of quantum mechanics in which the behaviour of mass and energy are different form the macro-world, this brings the problem of electron tunnelling where the gate of the transistor cannot prevent the flow of the electrons. This physical limitation is a huge problem for the semiconductor industry because the firms are not able anymore to counting on the traditional scaling approach base on the reduction of the size of ICs' components, while the market demand for better performance is still rising. To meet the requirements of next generation of Information and Communication Technology (ICT) systems, big firms like Intel, AMD and Nvidia have started to look for some possible solution. A first attempt for increase the integration density was made by 2.5D ICs in which the area of the silicon plane is reduced using a silicon interpose that allow dice to be placed into a single package. The 3D approach was defined later and unlike the previous ones it increases the integration density of ICs by stacking the dice one above another. Like in the natural world where high-rise buildings are the solution to accommodating large population in small area, stacking dice allow to increase the number of components that we can place in a fixed silicon area. 3D integration can be an enabler for performance and speed, but it can also reduce the power consumption and push the development of modular circuits based on independent tiers.

3D ICs are arousing more and more interest from researchers and companies thanks to its features, but of course all the advantage come with a price. The development and the integration of a 3D IC is still quite complex task. The way how the components of a circuit are placed into a tier and the selection of the tier will affect the performances of our device like the delay time, thermal dissipation, reliability, consumption, and area footprint. Moreover, with an additional dimension, is critical the management of inter-tiers connections that can become the bottleneck of the communication. It became essential to define a design path that allow to find the best placement and optimize the performance.

1.1 Motivation

The huge limitation that an 3D architecture implementation brings slowed researchers down and turned investors away. Fortunately, the IT and technology industry has continued to have an ongoing thirst for better performance, which has resulted in major manufacturers being unable to ignore the hardware limitation. New technology was implemented to develop the components of 3D ICs, but there still a lack in the design practices. 3D architecture brings new challenges respect to 2D, the placement of the elements needs to take in account also the vertical space and the inter-tiers and intra-tiers connections, because they affect in different way the total performance. Despite having the necessary technologies, limiting oneself to building a 3D IC by joining only the different components not considering the new constraints leads to bad results. It became more and more clear the need of best practices to follow for the development of the circuits. In 2D architecture the Place and Route algorithm is used to find the best placement for the components of the circuits and then check if their routes are feasible. What is needed now is an update of this algorithm that should take in consideration all the new aspect that a 3D architecture brings. Many companies have already begun to work to fill this gap, but for now no definitive solution has been found. The exploration of this kind of design become even more impellent with the Big Data shift. The amount of information produced by device increase exponentially every year creating value both for companies and client. New application like CNN (Convolutional Neural Network) and ML (Machine Learning) created their own place in the market arising a high interest from everyone. The need of even better IC able to manage such amount of data is now quite important. Joined with the parallelization of multiple devices the 3D architecture can be an enabler for these new applications. The develop of 3D IC is still now quite complex because it depends on the goals defined and technology used, but the advantage that it can brings are enormous. For this reason, the implementation of a Place and Route algorithm for 3D architecture can be the starting point for the development of new powerful devices and performing technologies.

1.2 Goals

The project goal is to define a Place and Route algorithm that, starting from a classic 2D IC, can define a 3D IC, finding for each element the best position. The number of tiers that characterize the output circuit is defined by the users, and the algorithm will focus on two main aspect: minimize the number of inter-tiers connection that represent a critical path and can slow down the performance; minimize the total wire length of the circuit. These two optimizations allow to reduce the delay time required for the communication between two components and minimize also the area required for place all the elements of the circuit. The obtained 3D IC will be tested to understand its performance and which advantage it brings compared to the 2D IC.

Chapter 2

Background

To implement 3D design, it is required to understand the different technology that can be used to create the circuit. Each of them is characterized by features that should be taken in consideration because lead to specific approach and solution. There is no generic method that can be use in any case, but there is a distinct design that derived from the goals and the technologies of the problem. The Place and Route algorithm can fallow an ASIC (Application Specific Integrated Circuit) design with the goal of create a circuit able to solve a precise problem. The ASIC approach allow to reach higher clock frequency and lower delay, but it arises thermal issues that should be manage especially when are present multiple tiers. On the other hand, there is the FPGA (Field Programmable Gate Array) design where the IC is programmable so it can execute different functions. Since the FPGA approach have a lower power density, the thermal issues are less important, but the cost of connectivity rise. This can be attributed to the fact that a larger number of possible connections must be facilitated in FPGAs, and this entails an overhead of the silicon area that must be used to implement pass transistor switches, buffers and SRAMs that enable this capability; in ASICs all we need to add is an inter-tier via that connects one active device tier to another one. This dichotomy of the design style is the starting point for the development of the algorithm.

Once the design style is defined, another important point is the interconnection between the IC's tiers. 2D ICs are based on a single flat mono-layer of silicon called die, on which all the elements that compose the circuit are placed. The interconnection inside the layer is made by channels which are of the order of magnitude of nanometres. Small size like that facilitate the management of the interconnection, over more having all the channel on the same layer help the thermal dissipation. 3D ICs are based on multiple silicon layer stack on to another that increase the complexity of interconnection management and thermal dissipation. Each tier will produce a thermal energy that will affect the upper layer, for this reason are required power constraint for both single layer and total stack. If two connected elements are placed on the same layer, their communication is managed like the 2D case, while if they are on different tiers the connection is managed by a particular channel able to pass through the silicon layers. Typically, the inter-tiers channels are slower than the ones inside the same layers and they are more easily subject to external disturb, for this reason is a good practice to reduce them when the placement is made. The main technology use for the inter-tier connections are two, the TSV (Through Silicon Via) and the MIV (Monolithic Inter-tier Vias). These two channels are characterized by different behaviours that give rise to two approaches: TSV-based 3D IC and Monolithic 3D IC. The 3D TSV-based integration is the focus of this thesis project.

2.1 TSV-based 3D IC

TSVs are few micrometres (≈5µm) in diameter, and they have large pitch (30-50µm) and keep-out-zone (KOZ) requirements. The KOZ define an area around the channel that cannot be used, due to thermal management components, cooling, and mounting constraints. In addition to that, they have large parasitic capacitance that become more evident whit the increase of temperature and reduction of diameter. With logic gate size scaling to less than 0.5µm^2 in 14nm technology nodes and below, such large TSVs will be beneficial only for coarse-partitioning for block-level or die-level memory-on-logic 3D IC designs. It become a crucial need to consider the impact of TSVs in the floor-planning solutions. TSVs are not only the I/O and power/ground ports of the layers, but they also introduce many uncertainty in performance, reliability and power consumption of 3D ICs. For example, TSVs can modulate the power supply noise and thermal profile of a 3D stack. Hence, TSV-aware and TSV-unaware 3D floor-planning may lead to very different thermal/noise profile for the entire 3D stack. TSVs are etched through the silicon layer with deep reactive-ion etching, insulated with thermal oxide, and then filled with liner and conductor. The contacts are made by backside metallization and depending on when the TSVs are fabricated, the TSV formation has various versions: Via-first, Via-middle, Via-last, and Via-after. The Via-first approach create the TSVs before building transistor and metal layer. The Via-middle approach create the TSVs after building transistor, but before building metal layer. The Via-last approach create the TSVs after building transistor and metal layer. The Via-after approach create the TSVs only after the connection with another silicon layer. The process of bonding two tiers is also important and can affect the performance of our IC. A single die is composed by two macro-parts, the "face" that represent the top metal layer, and the "back" that represent the bottom silicon substrate. In a face-face bonding the face of the lower die is connected to the face of the upper bound, while in the face-back the face of lower die is connected to the back of the upper die. The TSVs fabrication and the bonding approach need to be defined together and they will set the bases for the design of the IC. Last important step for the technological characterization of the 3D IC is the definition of the stacking strategy that effect the overall yield and chip size. The thin slice of silicon on which the die is fabricated are called wafer and can be used as interconnection base between to tiers. The main stacking policy[2] are three: Wafer-Wafer, Die-Wafer and Die-Die. The Wafer-Wafer stacking have the lower operating cost but typically lead to huge chip size and lower performance. The Die-Wafer and Die-Die approach enables the assembly of Known-Good-Die (KGD) and combination of dice featuring different sizes or aspect ratios. The main drawback of the individual die placement is the low throughput. A KGD is defined as a package type fully supported by suppliers to meet or exceed quality, reliability, and functional data sheet specifications, with non-standardized (die specific) but completely and electronically transferable mechanical specifications. The original intent of the KGD was to signify that bare die or unpackaged ICs had the same quality and reliability as equivalent packaged devices.

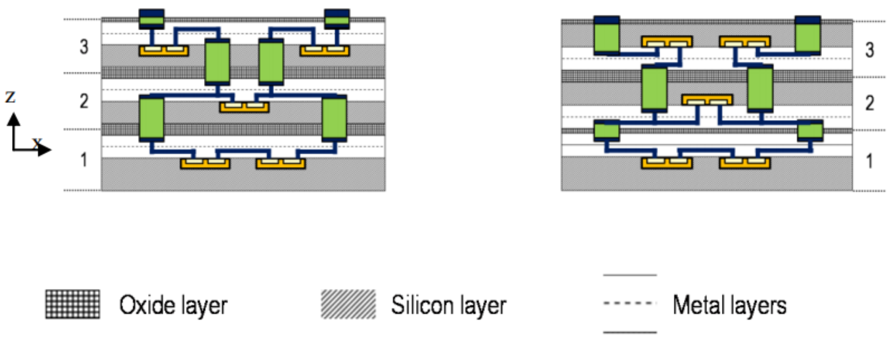


Figure 2.1: Two examples of TSV-based 3D IC cross-section view

Due to their large size and their strictly requirements, the TSVs should be manage carefully. A large number of TSVs can degrade the yield of the final chip. Also, under the current technologies, TSV pitches have huge size compared to the regular metal wires, usually around 5-10µm[3]. In 3D ICs TSVs are usually placed at the space between the macro blocks or cells, so the number of TSVs will not only affect the routing elements but also the overall silicon areas. Therefore, the number of TSVs in the circuit needs to be controlled and minimized. The placement of the circuit's elements defines the number of inter-tiers interconnections and their position, it means that an optimize placement can lead to a minimal number of TSVs with the best position. The floor-planner and placer must take in consideration also the latency and power consumption that characterize the circuits' inter-tiers connections. Stack several dice on top of each other and managing the communications with large wires create a critical thermal issue[4]. The multiple vertically piled layers of active components cause a rapid increase in power density that arise consumption and dissipation problems. The thermal conductivity of the dielectric layers between the different tiers is low compared to silicon and metal layer. For instance, in a closed space characterized by a temperature of 300K, the thermal conductivity for the epoxy used for gluing die is 0.05 W/m/K, and the thermal conductivity of the metal layers embedded in a dielectric is about 2 W/m/K. Both values are much smaller than the thermal conductivity of silicon (150 W/m/K) and copper (401 W/m/K). Therefore, the thermal issue needs to be considered during every step of the 3D physical design flow. To mitigate those problems a new kind of TSVs connections called mini-TSVs was created, characterize by smaller size around 2µm.

2.2 Monolithic 3D IC

The Monolithic 3D integration is a new design strategy that enable a huge performance improvement of our ICs. The architecture of a circuit is based on vertically stacked layers like the TSV-base method, but now the inter-tier connections are made by nano-scale vias called MIV. The MIVs are orders of magnitude denser than conventional through silicon vias (TSVs). Those new kind of system architectures can achieve remarkable performance and energy consumption compared to classic design. The increase of the vertical density connectivity makes those new designs particularly attractive for Big Data applications that impose high constrains with respect to low-latency data processing, high-bandwidth transmission, and energy storage of massive amounts of information. The tiers that compose the IC are in this case fabricated directly on the below layer allowing in this way the nano-scale interconnection. All the layers are built on a single semiconductor wafer so there is no substrate or bonding strategy. MIVs and pitch dimensions are the same of the metal layers and can enable a massive vertical integration of orders of magnitude denser then TSVs (≈1000X)[5].

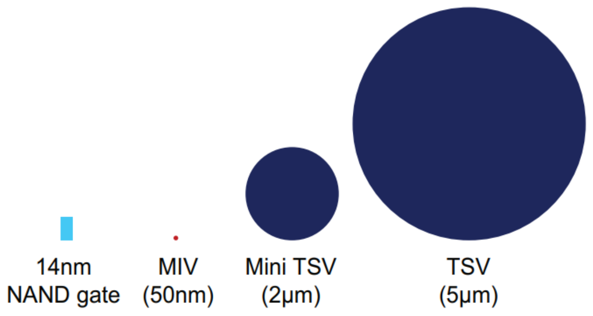


Figure 2.2: Size comparison between different inter-tiers vias

Despite all the promise the Monolithic approach made, it still in a R&D

phase due to processing obstacles that posed major roadblocks. The fabrication of the circuit on the upper tier must be done in a low temperature (<400°C), while preserving the performance of the underlying components and connections. Recent advances in semiconductor industry and nanotechnologies facilitated the development of Monolithic architecture and pushed big companies, like Qualcomm, to start new project base on it. Alternative to the silicon CMOS technology is being explored, like the carbon nanotubes (CNTs) that have size around 1nm. The CNTs are the base of the CNFETs (Carbon Nanotube Filed-Effect Transistor), a new kind of transistor projected to improve the energy-delay product (EDP, a measure of energy efficiency) of very-large-scale integrated (VLSI) digital systems by an order of magnitude compared to silicon CMOS. CNFETs make also possible to increase the integration level and the performance thanks to their nanoscopic size. The emerging memory technology such Resistive RAMs make the rise of Monolithic integration closer, thanks to higher storage density. Some low temperature procedure that can be used for building the Monolithic vertical stack of tiers are: Rapid melt re-growth, template-based epitaxy, and programmable interconnect. The Rapid melt re-growth is based on a heat micro pulse able to produce bulk on the top-oxide layer while maintaining the underlaying structure below 400°C. The template-based epitaxy method works like the previous one, but instead of using the laser pulse to melt the top-oxide layer we melt instead the top-silicon layer and then we re-crystallized it between the oxide grids. The programmable interconnect is based on the addiction of materials that change the resistance of metal oxides into our tiers. Doing so it will be possible on the top layer to reach higher temperature without effect the underlying level. Today the Monolithic 3D ICs are not ready yet for the commercialization, but with high probability they will become the dominant design of the future IC's generations.

2.3 3D Packaging

The 3D packaging enables to integrate 3D architecture relying only on the traditional interconnections like wire bonding and flip chip to achieve a vertical stack system.

This kind of strategy is not focus anymore on stack physical die on top of each other, but increase the density level by integrate multiple dice on the same package that will be place on a single layer. 3D packaging is becoming an increasingly used approach to multiply integration densities and performance in a single package. These packaging system solutions provide the necessary compute, IO, and memory scaling to address specialized workloads in the compute intensive markets like machine learning or AI. Advanced packaging solutions bring huge advantages, but they also create some challenges due to a larger form-factor, need for larger silicon interposer, higher power consumption, increased thermal and longer design cycles, that all must be addressed. All these issues can be managed by a unified platform with tight integration of system level signal, power, and thermal analysis, delivering automated power, thermal, and noise aware optimization. Having a consolidated view of the entire system is especially important because power and thermal analysis of an individual die in isolation is no longer enough in a multi-die environment, the full system needs to be analysed together. The main package strategy for 3D architecture is 3D SiP (System in Package), 3D WLP (Wafer Level Package) and BGA (Ball Grid Array). The SiP approach is based on share different resource between dice that will be placed in the same package. The dice will be bounded with the package using standard wires on flip chip technology. The WLP approach, instead, is based on packaging an IC while still part of the wafer, in contrast to the more conventional method of slicing the wafer into individual circuit (die) and then packaging them. This strategy is essentially a true chip scaling method since the resulting package have the same size as the die. The BGA approach is based on a surface-mount packaging used for IC. A BGA can provide more interconnection pins than can be put on a classic flat package. The whole bottom surface of the layer can be used, instead of just the perimeter. It means that we will end up with higher density integration. A particular case of WLP approach is the 2.5D interpose[6] that was the predecessor of the 3D IC architecture. An interposer is an electrical component routing between one package to another. The goal of an interposer is to propagate an interconnection to a different pitch or change its route. Most of the advantages of 3D integration could be reach by placing bare dies together on an interposer instead of placing them on top of each other. If size of the pitch is small and the interconnection are short, 2.5D approach will bring a solution with better size, weight, and power consumption compared to classic 2D ICs[7].

Chapter 3

Previous Works

Before the advent of 3D strategies, the circuits were based on a single layer of silicon and took the name of 2D IC. The first IC was built in the half of the 50s and it was much less powerful compared to the today ones. The creation of IC was the starting point of the enormous electronic and information development that has radically changed today's world. The market, education, entertainment, and healthcare are just some of the sectors that have developed a strong dependence on electronic devices and software application over the time. In recent years, this trend has strengthened to the point that some sectors have been completely renewed by new technologies and devices. Although today there is a high integration of electronic and digital components in society and it is difficult to imagine how things worked before, the evolution that led the first ICs to become more and more performing has been gradual. As mentioned in the previous chapter, there has always been a strong demand for better performance in the semiconductor industry. The real challenge in building 2D ICs was meeting these demands while keeping power consumption and size small. As in the 3D ICs became important to integrate more and more elements into a single circuit following Moore's law. The semiconductor industry investment started to focus on smaller transistor technology that led to today's nanometric scale components. Also, the way how the elements are placed on silicon layer become essential because it will affect the performance, delay time and thermal dissipation of our circuit. For this reason, with the goals of finding the best placement for all the IC's components and optimize the performance, the Place and Route algorithm was defined. The Place and Route is a stage of the IC design. As suggested by the name, this stage is composed by two steps, placement and routing. The placement step is the segment of the design flow that find and set an exact location for all the elements of the circuit, taking in consideration the generally limited amount of space available. Non optimized placement will not only affect the circuit performance but might also make it non manufacturable due to the presence of excessive wire-length, which exceeded the available routing resources. Consequently, is important to define several objective that the placer should perform while making the position assignment, only in this way the IC can be compliant with the constrains and meets the performance demand. The routing step follow the placement and adds the wires needed to properly connect the components of the IC. The interconnection of components is quite complex because it needs to be compliant with the circuits design rules. Those rules are geometric limitation[8] imposed by process engineers based on the capability of their processes to realize design. The main constrains applied to an IC are check if the width of a route is sufficient, check if the spacing between routes is sufficient and check if the enclosure of a route inside a component is regular and with a sufficient thickness.

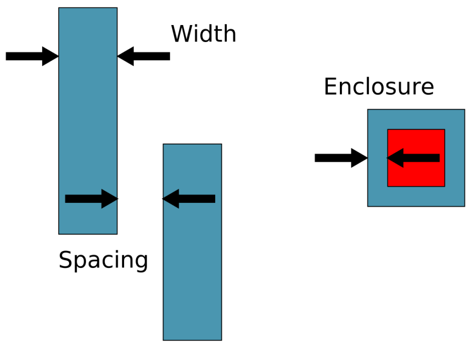


Figure 3.1: Three main design rules

Due to the frequent changing in technology and performance exist many kinds of routing process[9]. The first type to be studied were the maze-routers in which the interconnection between two terminals is made by using a maze-searching technique. Maze-router first searches the cells that are closest to the starting terminal, and then proceeds in precise spread path by searching the cells adjacent to those that have already been searched. When the end terminals have been found, the search stops, and the wire is created. The spread path is defined by the builder of the IC. This kind of approach always find the connections with minimum length, but not guarantee that the total length of all wires will be minimized. The channel-routers are an efficient alternative to the maze-routers, that solve the limitation of the previous approach but work only with specific restriction. While with maze-routers the wiring area have not constrain in size or shape, and the terminals can be placed anywhere, with channel-routers the wiring area (channel) is assumed rectangular with fixed terminals on the top and bottom edge. The simplest channel-router laid out on a horizontal grid and use two layers, one for vertical wire sections, and another for horizontal sections. A single net can have at most one horizontal segment called trunk which extends from the rightmost terminal to the leftmost terminal. Depending on the type of application the trunks can be fixed or variable. A trunk is connected to different terminals with vertical segments called branches. If a single branch contains terminals for two different nets produce one vertical constrain, the set of verticals constrain is usually represented as a directed graph with the nets represented by vertices ad constrains represented by direct arcs. The graph is used to implement the simplest of the channel routing algorithm called the left-edge algorithm. This algorithm goals are to find the minimum number tracks with shorter path that allow to interconnect all the nets with their terminals. A variation of the channel-routers approach that gives more flexibility to the routing process is the dogleg approach. With dogleg we define a vertical wire segment able to connect two trunks, doing so nets can be assigned to multiple trunk and this typically led to more compacts routing. A further development of the routing process was pushed by the switch-box approach that work like channel-routers but allow to place the terminal in all the edge of the rectangular wiring area. This new approach is more complex to mange but can lead to better solution. There are many other classes of routing process, most of which are designed to solve specialized problems such as power consumption or area optimization.

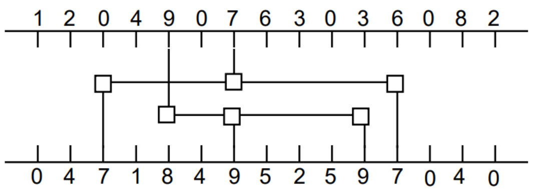


Figure 3.2: Example of Channel routing

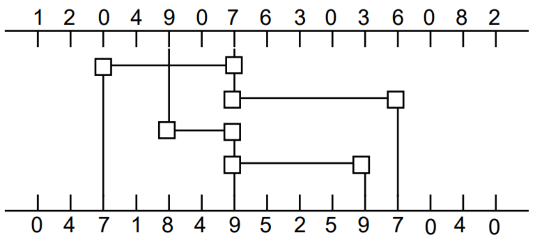


Figure 3.3: Example of Dogleg routing

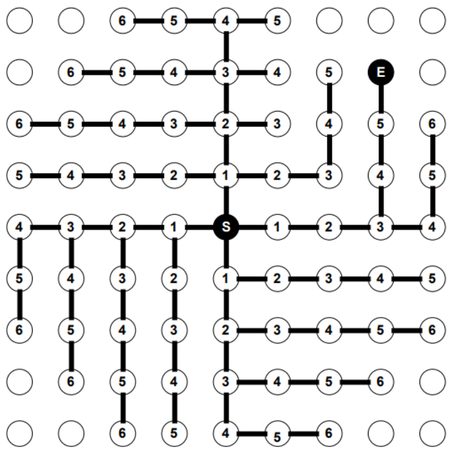


Figure 3.4: Maze route search pattern

The definition of good Place and Route algorithm is now essential for the development of an optimized IC. In the 2D case already exist a lot of different solution. The increasing focus on 3D ICs create the need of new algorithms able to take in consideration also the vertical dimension in the placement and routing phase, for this reason the thesis will focus on exploring those new approach. During the execution of the 3D algorithm, a 2D Place and Route algorithm (FT-TDP) is used to optimize the placement within the individual layers which are treated as independent 2D circuits. The management of the IC as a graph is also essential to the algorithm, for this reason a specialized library called METIS is used.

3.1 FT-TDP

Before the development of the 3D architecture, the ICs were built on a single layer. Despite the lack of vertical development, the positioning of the elements on the plane was essential to optimize the performance of the product. Like the 3D case, a lot of strategy were developed for placing and routing the units in a single layer. There is no better approach, there are different strategies depending on the goals you want to achieve. The typical objectives of the 2D placement are the reduction of heat flow, the minimization of the wirelength and the maximization of the reliability. This section briefly explains the FT-TDP (Fault Tolerant Timing-Driven Placement) strategy, which is used in one step of my 3D Place and Route algorithm.

The goal of the FT-TDP algorithm is to find a valid allocation for each configuration logic block while minimizing the total interconnection segments required and without inserting single point of failure (SPOF)[10] in the circuit. The SPOF is a potential risk posed by a flaw in the design, implementation or configuration of a circuit in which one fault or malfunction causes an entire system to stop operating. Starting from a circuit description, which includes Configurable Logic Blocks (CLBs), I/O pins, and their interconnections, the algorithm is able to return a 2D placement in line with the previous aims. The FT-TDP core is based on min-cut optimization technique applied on the units, mixed with quadratic placement that minimize the distance between the elements. The overall algorithm is composed by three phases[11]. Initially, a preliminary phase, initializes the environment by setting all the structures required for the placement. The second phase aims at reducing the overall delays of connections in a global sense, while the last phase is a local optimization to improve the specific critical paths related to the units. The circuit is represented in these steps by an undirected graph, where the vertices are the units to be positioned and the edges are the interconnections between the elements. Each vertex of the graph is characterized by a location and a kind. The location corresponds to the physical position of the logic element on the layer, and the kind describe the type of logic resources. The second phase of the FT-TDP consist in a global placement of all the units where, step by step, the circuit is partitioned, minimizing the number of cuts of the nets that connect component between the partitions. In this way the highly connected blocks are allocated in the same partition. The process is repeated until the number of logic element within a partition is limited to few blocks defined by the user. The goal of this min-cut processes is to partitioning the initial circuit while minimize the wires cut. The third and last step instead, apply a quadratic algorithm that tries to minimize the total squared length by solving linear equation of the distance between units that create a critical path. The critical path in a net defines the connection between an input and output with the maximum delay. Is important to manage those kinds of links because they can work as a bottleneck for the circuit performance. This step is repeated until there is not a significant improvement of the estimated delay.

FT-TDP is used in 3D algorithm after all the units have been assigned to their best tier. Is possible to apply the 2D placement to find the best position of each unit inside a tier. Unfortunately, the placement within the layers is conditioned by the presence of the inter-tier connections, so it is not possible to treat the layer independently. For this reason, FT-TDP is used only as a starting point for positioning the units in their tier of belonging.

3.2 METIS package

METIS[12] is a serial software package for partitioning large irregular graphs, partitioning large meshes, and computing fill-reducing orderings of sparse matrices. METIS has been developed at the Department of Computer Science & Engineering at the University of Minnesota and is freely distributed. For the scope of the thesis, we will focus only on graph concerning process. Algorithms that find a good partitioning of highly unstructured graphs are critical for developing efficient solutions for a wide range of problems in many application areas on both serial and parallel computers. Graph partitioning is becoming more and more important for solving optimization problems arising in numerous fields such as design of very large-scale integrated circuits (VLSI), storing and accessing spatial databases on disks, transportation management, and data mining. The algorithms implemented in METIS are based on the multilevel graph partitioning paradigm, which has been shown to quickly produce high-quality partitioning and fill-reducing orderings. The multilevel paradigm is composed by three steps: graph coarsening, initial partitioning, and uncoarsening. In the coarsening phase a series of smaller graphs is derived from the initial one. Each of the sub-graphs are created by collapsing together a maximum size set of adjacent pairs of vertices. This process continue until the size of the graphs is reduced to just few hundred vertices. The initial partitioning phase consist in partitioning of the coarsest, hence, smallest graphs, using relatively simple approaches such as the algorithm developed by Kernighan Lin[13]. Due the fact that the coarsest graph are also small this lead to a very fast execution of this step. In the uncoarsening phase the partitioning of the smallest graph is projected to the successively larger graphs by assigning the pairs of vertices that were collapsed together to the same partition as that of their corresponding collapsed vertex. After each projection step, the partitioning is refined using various heuristic methods to iteratively move vertices between partitions as long as such moves improve the quality of the partitioning solution. The uncoarsening phase ends when the partitioning solution has been projected all the way to the original graph. During coarsening, METIS employs algorithms that make it easier to find a high-quality partition at the coarsest graphs. During refinement, METIS focuses primarily on the portion of the graph that is close to the partition boundary. These highly tuned algorithms allow METIS to quickly produce high-quality partitions and fill-reducing orderings for a wide variety of irregular graphs, unstructured meshes, and sparse matrices.

The core of METIS is written in C/C++ to optimize the speed of execution. METIS provide us a set of stend-alone command-line programs that can work on different operating system and can be called from several programming language. In the project it is used a specific package of METIS called hMETIS specialized on partitioning hypergraphs that arise from circuit design. hMETIS provides the shmetis, hmetis, and khmetis programs that can be used to partition a hypergraph into k parts, but in the project only shmetis is needed. Shmetis program accept as input a file containing the initial graph, the number of partitions needed by the

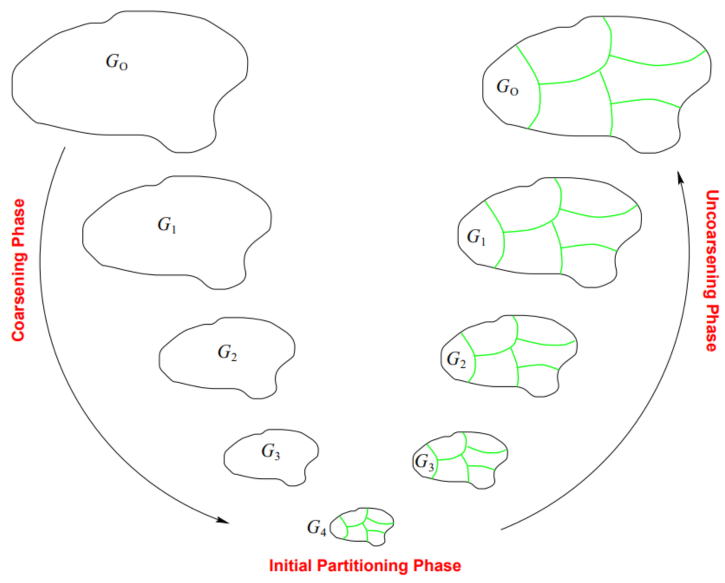


Figure 3.5: METIS multilevel partitioning

user and the balancing factor that define the level of unbalance allowed between the finial partitions. An optional parameter is a file containing the nodes with a fixed partition of belonging, sometimes it can help when elements of the circuit must be placed in a specific position of the silicon layer. Shmetis[14] return as result a file containing for each node of the graph the corresponding partition index. This program is used in the first step of the Place and Route algorithm described in the next chapter.

3.3 Strategies

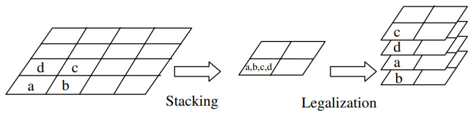
The 3D IC have aroused a lot of interest in both researchers and companies. Different strategies were studied and developed to define a Place and Route algorithm able to manage in a proper way this new kind of architecture. As explained in the previous chapters, the presence of multiple technology on which is possible to apply a 3D strategy and the huge number of parameters that should be taken in consideration for the optimization process, made this task quite hard to complete. Therefore, many of these algorithms were not able to fulfil their purpose, while others only partially achieved the objectives they had set. The first step for defining a new Place and Route algorithm was the documentation. Studying the previous approaches help to understand which are the most critical part of the development and the main limitation that arise during the implementation. Looking at old strategies is possible learn their strengths, that can lead to new ideas and optimizations, but also their weaknesses for avoiding making the same errors of other developers. The need for a new Place and Route algorithm arose from the observation of past documents, where very often the processes that determined the algorithm had been studied only in a theoretical way but never written in the form of a code. This is a lack that does not consider possible problems or limitations that derive from the software compartment. Furthermore, none of the algorithms examined manages the positioning of the TSVs considering the dependence with the position of the elements connected by it. They typically focus only on the minimization of the inter-tiers interconnections. Below are some of the main strategies for the Place and Route algorithm defined in the past few years.

3.3.1 Stacking and Folding

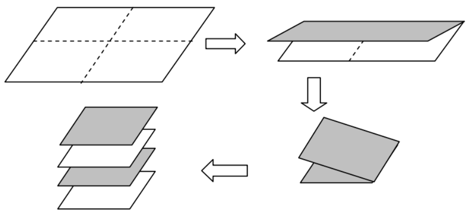
The Stacking/Folding is a thermal-aware 3D IC placement strategy. Starting from a 2D placement is able to adapt it into a 3D architecture using a sequence of transformations. The function of this strategy can be summarized as starting from a single layer that represents the 2D IC, we divide the plane into sub-sections of rectangular or square shape. Each subsection represents a tier of the final 3D architecture. The subdivision of the initial layer and the order with which the subsections are placed to create the pile of tiers can follow two approaches: The Local Stacking Transformation (LST) and the Transformation Through Folding (TTF). The LST consist in two steps, stacking and legalization. The stacking step shrinks the initial layer uniformly in N subsections, all the elements will belong to one of the sections characterise by an area that is N time smaller than the initial one. N is defined by the user and represent the number of tiers of the final 3D IC.

The legalization step minimizes the on-chip temperature and reduce the number of TSVs through the position assignment of the elements in each subsection. The elements' order is defined by two cost functions[15]. The legalization function represents the cost of moving an element from its initial position (x, y) to the final position (x, y, z), taking in consideration the effect of moving a cell through tiers on the TSVs number. The thermal function represents how the cells placement effect the total heat. Under the current 3D IC technologies, the heat sinks are usually attached at the bottom and top sides of the tiers' vertical stack, with other boundaries being adiabatic. It means that the main heat flow within the 3D IC stack is vertical towards the heat sink. Each cell will affect the total heat in different way, in particular their z location will influence the final temperature much more than the x and y location. This function allows to avoid putting the hottest cells into critical thermal spots of the circuits, leading to lower total heat flow. LST archives short wirelength by stacking the neighbouring cells together. However, a huge number of TSVs will be generated when cells of close nets are placed on different tiers. If the technology used required a limited TSV density we should use the TTF strategy. The TTF consist in folding the original 2D tier like a piece of paper without cutting off any parts of layer. In this way a TSV is required only when a net crossing the folding line, but the number of such nets should be quite small. The folding transformation define both the subdivision of the initial 2D IC and the vertical placement order of the different subsections. Is very important in this strategy the selection of the folding policy. There are many approaches for folding the IC, typically the best one depends on the kind of technology used and the structure of the circuit.

LST achieves greatest wirelength reduction at the expense of large amount of TSVs, while folding results in much smaller TSV number but longer wirelength and possibly high via density along the folding lines. Both strategies have pros and cons, so a good idea is merge them to increase the optimization of the solution while mitigate their problems. The windows-based stacking/folding strategy try to reach this goal by subdividing the initial 2D placement into KxK windows, then applying the TSV or TTF on each of this window. The final 3D IC is composed by placing on the same tiers all the subsection of each window that belong to the



(a) Local Stacking Transformation.



(b) Transformation Through Folding.

Figure 3.6: Example of LST and TTF approaches
J. Cong, G. Luo, J. Wei, and Y. Zhang ~"Thermal-Aware 3D IC PlacementVia Transformation"

same layer. Wirelength reduction is due to the following reasons: the wirelength of the nets inside the same square is preserved; the wirelength of nets inside the same window is most likely reduced due to the effect of stacking/folding; and wirelength of nets that cross the different windows is reduced. Therefore, the overall wirelength quality is improved. Meanwhile, the TSVs are distributed evenly among different windows and can be reduced by choosing proper layer assignments.

3.3.2 Optimal blocks

The strategy proposed demonstrates that allocating blocks of the initial 2D IC into the tiers that compose the final 3D IC in a compact fashion, lead to solutions with lower wirelength and less TSVs in most of the cases. Again, this algorithm focuses on minimize the wirelength and the number of inter-tiers interconnection, while satisfying the space limitation. The block is defined as a subsection of the initial 2D placement. If multiple blocks are bounding together a net is created. The placement in multi-tiers can be modelled in term of hyper-graph. The set of blocks of the circuits define the nodes, while the nets define the hyper-edges. The number of tiers K is selected by the users and the tiers' area will be equal to the initial total surface over K. The area constrains is typically increase of the 15% to allow easier manoeuvre like blocks replacing and moving. The wirelength value is estimate by adding the half-perimeter length of the bounding box of the net in every layer and the length of TSVs required for the net. The bounding box define the rectangle that contain in each layer the different blocks that belong to the same net. Is possible to have for a single net multiple bounding boxes, one for each tier. The number of TSVs for a net is define as the difference between the highest and the lowest layer where the net is being used, while the length of a TSV is given by the technology used. The algorithm is composed by three parts, first we go on placing the blocks in each tier according to their interconnections, starting from bottom layer. The second phase consist to allocate the unplaced blocks in the vacant spaces left in each tier. The third phase is nothing but the re-iteration of first and second phase, which will be executed if there is at least one block remaining to be placed after the second phase. An important fact is that all blocks are allowed to be rotated, to be placed in a compact fashion, in every step of the allocation procedure, except the initial block, which is being placed at first in every tier. Before the first step we need to compute for each blocks its connection degree, then sort them according to a non-increasing order of this value. This is done in order to deal with densely connected blocks first.

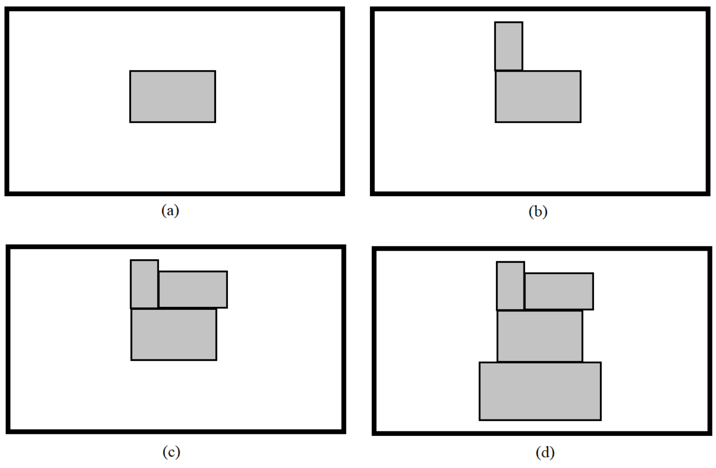


Figure 3.7: Adjacent placement of blocks according to their connectivity

The first step starts from the bottom tier and the most densely connected block. The block is placed in the centre of the layer and is it marked as placed. Then, is it selected the second most densely connected unplaced block that have at least a connection with the already placed block. In other words, the next block is selected on the basis of its degree of connectivity to the already placed blocks. We try to place this selected block adjacent to one of those already placed blocks with which it has a connection. This process continue till no more blocks can be placed in this layer. The block may be placed on any of the four sides of the previously placed block. The selection of an unplaced block for the tiers above the first one is based on two criteria. The first criterion is the area of the block. If there is a block with surface greater than or equal to 40%[16] of the area constraint of the layer it is selected first because big blocks like these hinder the compactness of the placement, so we allocate them in one of the four corners of the tier. The second criterion is based on degree of connectivity, like the placement for the first layer, and it is use only if there no block with surface greater than or equal to 40% of the tiers ‘area constraint. Once the placement for each layer is completed, the second step start. For each block not placed in the first phase we compute their connectivity degree and number of TSV added if they are allocated in one of the tiers. This approach allows to define for the remaining blocks the best tiers and fix their position. In the event that after the execution of the first two phases some blocks still remain unplaced; the third step restart the previous operations to find a feasible solution.

3.3.3 Multifactor placement

This strategy provides a multi-parameter criterion and an appropriate approach for the assignment of IC’s elements into a 3D architecture. The main goals of the algorithm are the minimization of the total length of connections between the components and the optimization of the power consumption for the final 3D IC. The tiers in this case are named crystal, and their number K is defined by the users. Each crystal consumes a specific quantity of power and the total power consumption will be equal to the sum of all crystals consumption. The final temperature reached by the 3D IC will derived from the total power consumption, the environment temperature, and the circuit shell temperature. Between the several crystals are present multiple resistance that can affect the heat flow of our circuit. From the point of view of 3D design with the given powers of crystals, for minimizing temperature the layers must be placed in descending order of powers.[17].

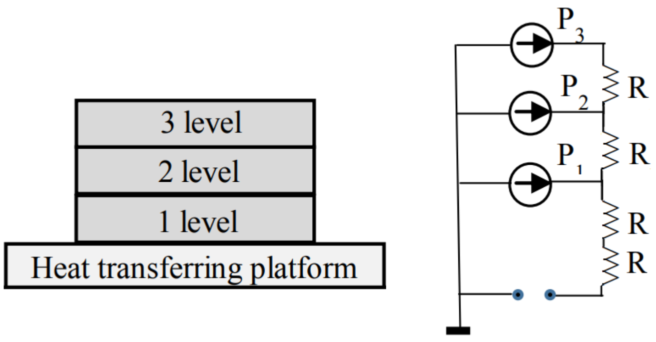


Figure 3.8: Example of power and resistance distribution into a 3-tiers architecture V. Sh. Melikyan and A. G. Harutyunyan ~"3D integrated circuits multifactorplacement"

Given all the information specified above, the algorithm involve three phases. The first step consists in subdividing the initial 2D placement into K crystals, while minimize the number of interconnections between layers. The second step is the definition of the position of the crystals in the 3D IC's vertical pile according to their power consumption. The third and last step is the placement of the cells inside the crystal with thermal field levelling and with minimal interconnections' length. For the first step a fragmentation sequential algorithm is used. This algorithm achieves the minimization of connections among crystals by the recursive repetition of the following steps. For each cell that belong to the initial circuit we compute how the quantity of interconnection between two crystal change by assign the cells to one of the possible layers. When the crystal that bring less interconnections is found the cell is assigned to it. This process continues until all the cells are allocated to one crystal. During the assignation phase, it is required that the total area of each crystal not exceed a value equal to the 2D IC area divided by number of crystals. The second step assumes that the power characteristic of each crystal's cells is known, so is possible to compute the total power of each crystal. Then, the crystals' vertical position in the 3D architecture is set according to a descending order of power. For the third phase a sequential placement algorithm is proposed which will consider powers values and interconnections at the same time. The management of thermal reduction is achieved by computing how the average temperature of a cell changes with different placement of the elements on the crystals. In this way is possible to understand how a single cell placement affect the heat of a layer. Knowing that, is possible to find for each cell the position that minimize the total temperature. To minimize the wirelength on a single crystal the geometric distance between two cells is taken in account while the average temperature is computed. The final goals is finding for each cell the position that reduce the temperature while minimize the distance with the other interconnected cells.

3.3.4 FPGA TPR

For the development of an FPGA architecture several considerations must be taken into account. As designer is necessary to provide a good balance between fabrication cost and speed. A 3D architecture brings new challenges like the routability and the area overhead. An important factor affecting the performance and area efficiency of the 3D FPGA is the routing architecture. Route with too much capillarity will waste area, and huge quantity of inter-tier interconnections will hurt the performance of the design. The algorithm proposed to provide an 3D FPGA architecture taking in account all the previous parameters is the Three-dimensional Place and Route (TPR)[18]. TPR is composed by three steps, the first one employs a partitioning phase using the hMETIS algorithm to divide the circuit into a number of balanced partitions, equal to the number of tiers for the 3D IC. The goal of this first min-cut partitioning is to minimize the connections between tiers, which translates into reducing the number of TSVs and decreasing the area overhead associated with the circuit. After dividing the initial 2D IC into seveal tiers, TPR continues with the second step where the placement on each layer is made by using a hybrid approach that combines top-down partitioning and simulated annealing. The annealing process moves cells mostly within tiers. Finally, in the last step, the cells are routed to check the presence of possible critical path.

In the first step, the 2D IC, is represented as a graph where the nodes define the cells, and the vertices define the interconnections. Once the initial IC is subdivided by hMETIS into clusters we need to place the different tiers in a way that the wirelength is minimized. This is achieved by mapping this problem to the

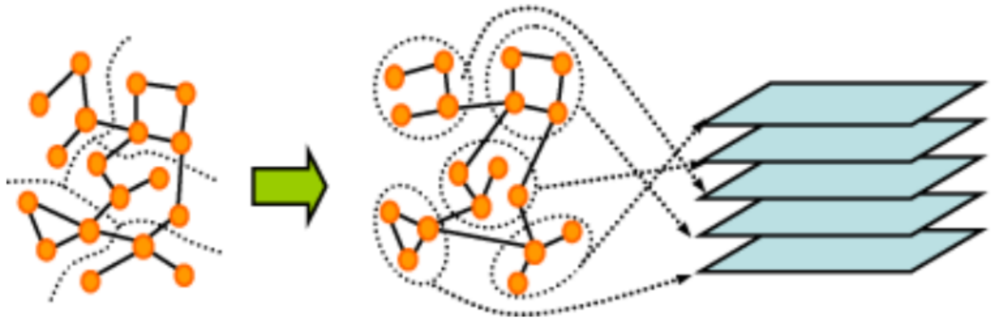


Figure 3.9: Partitioning 2D IC into tiers
C. Ababei, Y. Feng, B. Goplen, Hushrav Mogal, T. Zhang, K. Bazargan, and S. Sachin ~"Placement and routing in 3D integrated circuits"

bandwidth minimization of a matrix, using an efficient bandwidth minimization heuristic algorithm. The bandwidth of a matrix is defined as the maximum bandwidth of all its rows. The bandwidth of a row is defined as the distance between the first and last non-zero entries. The clusters and their interconnections are represented by a matrix that contain a number of columns equals to the number of clusters, and a number of row equal to the number of edges. An entry a_{ij} is non-zero if cluster j is incident to edge i. Minimizing the matrix bandwidth achieves two main goals: it minimizes the span of every row and distributes the bands across columns. The processes used to solve the bandwidth minimization problem apply row swaps in order to sort rows such that non-zero elements are moved towards the main diagonal. The second step focus on placement within tiers. Starting from the top layer and proceeding tier after tier, the cells' position is defined by an edge-weighted partitioning, computed again by hMETIS. Following this placement, low-temperature annealing allocation is made to further improve wirelength and routability. The annealing is based on a cost function that consider the number of tiers, the span between tiers and the area of the different cluster. The last step is the routing algorithm in which the 3D FPGA architecture is represented as a routing resource graph. Each node of the routing resource graph represents a wire (horizontal vias into a tier, TSVs into inter-tier interconnections) or a logic block I/O pin. A directed edge represents a unidirectional interconnection. A pair of directed edges represents a bidirectional interconnection. Extra penalties are added to communications based on route composed by a horizontal track and a TSVs as well as to TSVs alone, in order to discourage the routing engine to use vertical vias. Avoiding, in this way, that the majority of cells placed in one tier to be routed using tracks in different tiers.

Chapter 4

Algorithm

This chapter explain how the Place and Router algorithm is defined and how all its features have been modelled. The new algorithm is developed for the placement and routing of TSV-based 3D ICs in both FPGA and ASIC devices. The optimization of our result is based on different processes which goals is initially minimize the number of TSV inter-tier connections that cause delay, space and thermal problems. Then minimize the wirelength of each layer that compose the IC by finding the best position of each module inside its own tier. During the optimization process is needed to consider different constrain. The algorithm require as input the description of a starting 2D IC containing all the information needed to manage the circuit, like the interconnections and the identifier of each element. Is possible to represent an IC using specific tools like Libero Soc[19], that use several file to show different aspect and characteristics of our system. The Physical Design Constrains (PDC) is a file that contain for each cell its coordinates in a Cartesian plane that represent the silicon layer. Verilog is a hardware description language (HDL) used to represent digital electronic systems. A Verilog file will contain all the interconnection between the cells of an IC. The Standard Delay Format (SDF) file is produced by implementation tools and contains for each cell the delay and timing check. To define a proper representation of the 2D IC the PDC and the Verilog files are merged together to create the Physical Design Description (PDD) file. The PDD file contains the number of elements that belong to the IC and for each of them show main information like identifier, name, 2D-coordinates, and the list of interconnected elements. The PDD file is used as input for the new algorithm and its format is also employed for the output file that will contain for each element also the vertical coordinate that define the tier of belonging in the 3D placement. Is important to notice that the PDD representation manage the circuits like a graph in which the vertices are the elements and the edges are the interconnections. This graph-like definition is important because it push us to adopt solution feasible for hypergraphs and synergizes very well with the METIS package and its processes.

```
9                // ID
\d_in[2]         // name
C DFN1C1         // labels
3 4              // n.inputs - n.outputs
47               // X coordinate
68               // Y coordinate
1                // fixed value
U                // Z coordinate (to set)
1 89 CLK Y       // list of input elements
2 90 CLR Y
3 24 D Y
1 23 Q A         // list of output elements
2 48 Q B
3 72 Q B
4 63 Q A
```

Figure 4.1: Section of PDD file

Before developing the algorithm, it is important to define, in addition to the objectives, the constraints to be respected during the execution of the processes. The area of each tier should be smaller than the initial surface of the 2D IC because the density level must increase. Over more, the tiers area should be decided in relation to the number of elements to be placed in order to avoid wasting space. Even if we try to minimize them, an excessive number of TSVs can lead to solutions that are not feasible for real production, so we should anyway check their quantity. The position of a TSV should be chosen carefully because the elements belonging to different levels, but connected by the same TSV, should have the same 2D-coordinates, thus being one above the other or at least as close as possible to this solution. The position of the pins is also subject to constraints; they can be placed only in the first or in the last layer of the 3D IC, in order to facilitate the connection with the main board. Moreover, pins should be allocated in a way that minimize the solution wirelength. To check if the algorithm is compliant with those constraints, during the execution, partial results are printed as output and graphical representation of the 3D IC are shown.

The Place and Route algorithm is composed by five steps that, starting from a 2D PDD file, are able to produce a final 3D PDD file containing for each element a 3D-coordinates that define the tier of belonging and the position on it. The objectives of the first three phases are the minimization of the number of TSVs, therefore the identification of the best level for each component of the IC. The fourth step begins when each element has been assigned to a tier, now the main goal is finding the best 2D placement in each layer in order to minimize the wirelength. In this step is extremely important the placement of elements connected through TSVs with other tier's components, for this reason different approaches were made to understand how the final solution can be affected. The fifth and last step focus on the placement of the pins which depend on the position of all the other components, for this reason it must be done as final operation. Besides the parameters necessary for execution, such as the input file and the number of tiers, the user is able to define other options that will change the algorithm's behaviours. Is possible to set by a specific parameter if the tiers' area should be computed starting by the total number of elements of the IC, or by the surface size of the initial 2D IC, that can be obtained by the minimum and maximum values of the 2D-coordinates. The user can also link an input file containing a list of elements' names. All the components identify by these names belong to a specific tier and during the placement phase cannot be moved from it. This kind of feature can be very useful with IC characterized by peculiar architecture that require fixed position for some components. To give an example, for CNN inference[20] IC architecture was developed in which fixing the position of specific blocks can implementing low-latency and energy-efficient solution. To summarize how the Place and Route algorithm works, its different steps are listed below with a brief description that will be expanded in further chapters.

1 - Modularization: Starting from the 2D PDD file a graph of the IC is composed. The graph will be divided in sub-graphs (modules) that will become the base component of the algorithm, that will move them across the different tiers.

2 - BFS vertical placement: An initial tier is assigned to all the modules thanks to the exploration of the graph using Breadth First Search (BFS) algorithm.

3 - TSV optimization: By computing the cost of moving modules through all the distinct tiers is possible to find solution different to the initial ones that have a lower number of inter-tier interconnections.

4 - 2D placement: Once the components belonging to each tier have been defined, the modules are decomposed into the initial nodes which are placed in their layer following a policy of reducing the wirelength.

5 - Pins placement: The IC's pins are placed on the first layer according to the distance with the others components.

4.1 3D Place and Route

In this paragraph is it shown how my algorithm work, going into details on the different steps' goals and their characteristics. As previously explained, mine Place and Route algorithm is composed by five steps: Modularization, BFS vertical placement, TSV optimization, 2D placement and pins placement. The main goal of the algorithm is the minimization of both wirelength and number of TSVs, while satisfy the area and architectural constraints. When a step manages a quite complex process its pseudo-code is shown, so it will be easier to understand it.

4.1.1 Modularization

The Modularization is the first phase of the Place and Route algorithm. Its goals are the definition of the main structures that will be used several times in the whole process, and the subdivision of the initial 2D IC into multiple modules. Initially, the 2D PDD file that represent the circuit as a graph is read and all its information are stored. The main structure created are the *nodes* that contain all the elements belonging to the IC with all their characteristics, and the *edges* that store all the interconnections between IC's components. As explained previously the PDD file identify each element with a univocal ID, for this reason an interconnection is represent through a pair of IDs. To avoid storing the same edges multiple time with different ID sequence, the ID pairs are sorted in ascending order. Together, *nodes* and *edges*, define the initial 2D IC graph that will be used often during the algorithm. Two other important structures are the *pins* and the *units*; these respectively store the IDs of all pins and the IDs of all units (non-pin elements) belonging to the 2D IC. To distinguish these two types of elements it is necessary to observe their initial coordinates in the PDD file. The pins are fixed in the origin of the Cartesian plane (0,0), while the units cannot have null coordinates. Is required to discriminate between pins and units because, to minimize the wirelength, the pins placement is computed on the units' location. For this reason, the pins position is defined in the last phase of the algorithm, when the units' location is already fixed and cannot be changed. To manage initially only the units without care about the pins a mapping process is applied. This process removes from the IC graph all the nodes identified by a pin's ID and detach all the edges where one of the elements is a pin. Before the remotion two new structures are defined, *elements* and $elements_{reverse}$. Those two variables are required because by deleting the pins' ID, the remaining units have a non-sequential list of IDs that contains some gap, but the hMETIS algorithm used for the Modularization accept in input only graph with sequential nodes' ID. Into *elements* is it stored the mapping between the true units' ID with a temporary ID that assume a sequential value between 1 and the number of units. Into $elements_{reverse}$ is it stored the opposite mapping, from temporary ID to true ID. The main structure on which the development of the new graph is based is *connections* that contains for each unit all the IDs of connected elements. *Connections* is useful because allow to access to all the interconnection of a single units without having to read the whole *edges* structure. Once the elements' ID are mapped and the pins are eliminated, the final result is a new graph based only on the units.

The Modularization process is now applied to the new graph. As previously explained this phase consist in subdivide the initial IC into multiple modules. A module is the main item for the tier selection process, and it can contain one or more units. The partitioning of the 2D IC bring two advantages: computational optimization and cut minimization. The majority of the IC, also the less complex, are composed by huge number of elements that form a dense network of interconnections. By representing the circuit as a graph, managing all the single nodes and interconnections directly involves an enormous amount of time and required computational power, but often this type of approach is not feasible. The modularization group multiple units together, so are possible to manage them as a single element. In this way, the number of components considered is reduced and consequently also the number of operations required to execute the algorithm. Even if the use of the modules involves an approximation of the best result, if the subdivision of the units is carried out with specific approaches (min-cut in our case), the result obtained will be very performing. The partitioning of the 2D IC is made by hMETIS script that provide a min-cut approach. Having the minimum number of connections between modules means that inside each partition the nodes have a high interconnection density. In this way is possible to treat together the units most connected to each other, which typically must be positioned close in the circuit. Using hMETIS allow to reduce the computational power required for running the algorithm, while providing a performing placement result. An important step for the Modularization is the definition of the number of modules, which can heavily affect the partitioning result. Too many partitions increase the complexity of the execution, while a scart number of partitions decrease the mobility of the units. The best number of modules can change in based on the characteristics of the initial circuit; it is up to the user to select it according to the different scenarios. In my project the 2D IC on which the algorithm is applied is characterized by several elements always between 100 and 21000. Initially, I thought it was necessary to

Algorithm 1 Mapping algorithm. Reads the 2D PDD file and generate the main structures.

```
procedure MAPPING($pdd$)
    ▷ $pdd$ Location of the 2D PDD file
    $nodes \leftarrow dict()$    ▷ Dictionary for storing all the IC's components
    $edges \leftarrow set()$    ▷ Set for storing connections between components
    $pins \leftarrow list()$    ▷ List for storing pins' IDs
    $units \leftarrow list()$    ▷ List for storing IDs of non-pins components
    $elements \leftarrow dict()$    ▷ List for mapping true ID with temporary ID
    $elements_{reverse} \leftarrow dict()$    ▷ List for mapping temporary ID with true ID
    $mapping_{id} \leftarrow 1$    ▷ Set first temporary ID
    $connections \leftarrow dict()$ ▷ Dictionary that store for each unit its connections

    ▷ Execution
    ReadFile($pdd$)    ▷ Update $nodes$, $pins$, $units$
    for $id$ in $units$ do    ▷ Mapping the nodes IDs excluding pins
        $elements[id] \leftarrow mapping_{id}$
        $elements_{reverse}[mapping_{id}] \leftarrow id$
        $mapping_{id} \leftarrow mapping_{id} + 1$
    end for
    for $k, info$ in $nodes$ do    ▷ Generation of new graph without pins
        if $k$ in $units$ then
            $k_{map} \leftarrow elements[k]$
            if $k_{map}$ not in $connections$ then
                $connections[k_{map}] \leftarrow set()$    ▷ Define a new entry
            end if
            for $id$ in $info[connections]$ do    ▷ Mapping the connected IDs
                if $id$ not in $pins$ then
                    $id_{temp} \leftarrow elements[id]$
                    $connections[k_{map}]$ add $id_{temp}$
                    if $k_{map} \geq id_{temp}$ then    ▷ Sort pair to avoid duplication
                        $edges$ add $(id_{temp}, k_{map})$
                    else
                        $edges$ add $(k_{map}, id_{temp})$
                    end if
                end if
            end for
        end if
    end for
end procedure
```

have a fixed number of partitions equal for each tier, so I defined a baseline of 4 module for each layer. This approach worked well in the presence of few elements but is limiting for circuits with many units. For example, a circuit with three layer and 21000 units can have at most 12 partitions with an average of 1750 units in each module. This leads to low flexibility in movement. Consequently, I observe that other than depending on the number of tiers, the number of modules also had to derive from the quantity of units. Initially I tried to assign to each tier a number of modules equal to the number of units divided by the number of layers. As expected, the result obtained was an excessive number of modules which only made execution very slow and did not lead to good partitioning. I started to assign to each layer a number of modules equal to a percentage of the total units. I made different test where, with a higher percentage the number of modules increase and bring more process complexity and the possibility of having some void partition that are useless and make the computation heavier. On the other hand, with a lower percentage the number of modules decrease and brings a faster solution but with excessive modules size. If the partition contains too many units it will be hard to move it in the TSV optimization phase due to the lack of space on the tiers. After performing several partitioning, I found that with a number of units ranging between 100 and 21000, the best compromise between module mobility and interconnection minimization is achieved by assigning to each layer a number of modules equal to 1% of the total units. This solution is also based on the assumption that in a real 3D IC the maximum number of tiers used hardly exceeds five. In case the user wants to modify this percentage, he can do it through a specific parameter.

To call the hMETIS partitioning algorithm three parameters are requires. The *modules* that define the number of modules which computation is explained above. The *balance* variable that specifies the allowed disparity between the partitions. It can assume value between 1 and 49 and specifies the allowed load imbalance in the following way. Consider a hypergraph with n vertices and let b be the *balance*. If the number of modules is two, the number of vertices assigned to each partition will be between $(50 - b)n/100$ and $(50 + b)n/100$. It means that low value of b brings low unbalance, while high value allows higher disparity.

For example, for b equal to 10, we will be allowing a 40-60 bisection, that is the number of vertices in each partition will be between $0.40n$ and $0.60n$. This allowed imbalance is applied at each bisection step, so if instead of 2 modules we are interested in 4 modules, then a *balance* of 10 will result in partitions that can contain between $0.40^2n = 0.16n$ and $0.60^2n = 0.36n$ vertices. The optimal solution would be having all the modules composed by the same number of units, but this is not feasible because it would force the partitioning algorithm to give less importance to the cuts minimization. Maintaining a good balance between partitions makes it easier to move modules between tiers and prevents a single block from taking up most of the area of a layer.

Algorithm 2 Modularization algorithm. Graph subdivision into modules by hMETIS partition algorithm.

```
procedure MODULARIZATION($graph_{METIS}$, $edges$, $nodes$, $tiers$, $balance$)
    ▷ $graph_{METIS}$ Name of the output METIS graph file
    ▷ $edges$ Set containing the connections between IC's units
    ▷ $nodes$ List containing the ID of all IC's units
    ▷ $tiers$ Number of tiers
    ▷ $balance$ Value that define the balance between modules
    $f \leftarrow$ open($graph_{METIS}$)                          ▷ Open the file in write mode
    $partitions \leftarrow list()$                        ▷ Store for each module its units

    ▷ Execution
    $f.write(len(edges), len(nodes))$    ▷ Write firts line (numedges, numnodes)
    for $pair$ in $edges$ do
        $f.write(pair)$                          ▷ Write pair of IDs that define an edge
    end for
    $f.close()$
    $modules \leftarrow tiers * ceil(len(nodes) * 0.01)$              ▷ Number of modules
    hMETIS($graph_{METIS}, modules, balance$)            ▷ Call hMETIS algorithm
    $f \leftarrow$ open($graph_{METIS}$+".part."+$modules$)
    for $id_{partition}$ in $f.read()$ do                ▷ Read the partitioning output file
        $partitions$ add $id_{partition}$
    end for
end procedure
```

For those reason, the *balance* parameter is set to 1 (lower possible value), in this way the smallest difference in size between the modules is guaranteed. The last parameter is $graph_{METIS}$, a file representing the graph to be partitioned. The structure of the $graph_{METIS}$ is composed by a first line containing the number of edges and nodes, while the other lines contains edges pairs ID. Moreover, as explained before, the nodes must have a sequential ID without gaps. hMETIS algorithm generate as output a file containing a number of line equal to the number of units, each line contains the partition ID of the corresponding node. The core of Modularization consists in two steps. First the $graph_{METIS}$ file is created and the hMETIS algorithm called. Then, the output file generated by hMETIS is read and is content is saved in the *partitions* structure.

4.1.2 Vertical BFS

The BFS vertical phase is applied on the partitions generate by the Modularization. It assigns to each module an initial tier with a selection base on the BFS (Breadth First Search) algorithm. The main idea behind this initial positioning is to keep the modules that have a high level of interconnections between them in the same tier. Doing so, the number of TSVs required for communication between two partitions on different layer are minimized. BFS algorithm is used for traversing a graph or a tree. It starts from a root node and explore all the neighbour nodes at the present depth prior to moving on the nodes at successive depth level[21]. By making a small modification the algorithm can base its exploration on the interconnections density between the nodes. Therefore, once the starting root has been defined, the BFS return all the nodes in a sequence which represent the best placement order for minimizing the number of TSVs. To generate a graph that represent the modules and their interconnections a pre-processing step is define. It will create all the structures required to execute in a proper way the BFS algorithm.

The pre-processing step define three main structures. The $modules_{nodes}$ that for each module store the IDs of the connected partitions and other information. The $modules_{edges}$ that store pairs ID that represent the links between two modules. The $module_{max}$ that define the ID of the module with the higher number of connections.

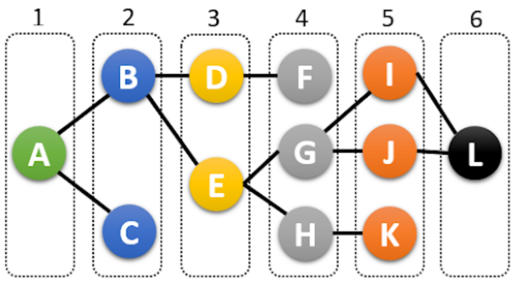


Figure 4.2: Breadth First Search algorithm with starting node A

The first two structures allow to manage the partitions as a graph, making more feasible the application of the BFS algorithm, while the $module_{max}$ will be used as starting point for graph exploration. Using the partitions define by hMETIS, the pre-processing algorithm initialize for each module several information and save them in $modules_{nodes}$:

- **position:** Sequential ID representing the membership tier. It can assume value between 0 and $n-1$, where n is the number of tiers.
- **size:** Value counting the number of units inside the module. It will be used for defining the space required into a tier to accept the module.
- **nodes:** List of unit IDs contained in the module.
- **outputs:** Value counting the total number of connections with other modules. It will be used for leading the BFS algorithm exploration.
- **modules:** List containing for each module the number of connections with the current module. It will be used to define the level of interconnections between partitions.

The $modules_{nodes}$ is populated by cycling through the list of units. When a unit belong to a specific partition the corresponding module is update with several information. The *size* is increase by one and the unit's ID is added to the *nodes*. To update *outputs* and *modules* the list of elements connected to the unit belonging to the current partition is needed. The structure *connections*, that store for each unit the IDs list of the connected elements, is used for this goal. *outputs* is incremented by one each time a connected unit belongs to a different partition, while the value of *modules* cells identified by the partition's ID of each connected unit are incremented by one. Each time two interconnected units belong to different partitions the $modules_{edges}$ is update with a new edge represented by the pair of partitions IDs. Before moving on to the next unit it is checked whether it is necessary to update $module_{max}$. If *outputs* value is the highest reached up to that point, the ID of the current partition becomes the new $module_{max}$. Once all the units have been processed, the generated graph represents all the modules and their interconnections, so it can be explored using the BFS. The BFS algorithm is based on three different structures. $Visited$ is the list containing the IDs of the already checked modules. *Queue* contain the sequence of IDs that need to be visited. *Bestpath* is the output of the algorithm and represent the sequence of modules to follow for the vertical placement. The BFS algorithm start the exploration of the graph from the module with higher level of connections ($module_{max}$). Then it checks its neighbourhood defined by the closest nodes with same depth. When a node is visited for the first time, its ID is added to the *queue*. When all the closest nodes are visited the algorithm remove the first module inside the *queue* and start to check the neighbourhood of the new root node, always considering the already visited nodes. When a module leaves the *queue* it is added to the final *bestpath* sequence. To ensure that the nodes with the highest number of interconnections are treated first, two precautions must be taken. As previously explained, the starting node of the BFS must be the module with the highest number of connections. In this way, modules that can lead to many TSVs have a higher probability of being placed on the same layer. Furthermore, the nodes belonging to a neighbourhood will be sorted according to the number of their interconnections with the root node. In doing so, the exploration prefers, among the nodes closest to the starting one, those characterized by more connections. This means that modules with multiple interconnections between them are more likely to be contained in the same tier. Once all the modules have been visited, the *bestpath* contains the IDs order that will be used for vertical placement. This sequence focuses on keeping modules with a lot of connections close together (same tier), minimizing the number of TSVs.

Algorithm 3 Preprocessing algorithm. Preparation of the structures used for the initial placement.

```
procedure PREPROCESSING($units$, $partitions$, $connections$)
    ▷ $units$ List of units' ID
    ▷ $partitions$ List containing for each unit the module of belonging
    ▷ $connections$ List containing for each element the IDs of connected units
    $modules_{nodes} \leftarrow dict()$        ▷ Dictionary storing for each module its nodes
    $modules_{edges} \leftarrow set()$            ▷ Set storing the modules interconnections
    $module_{max} \leftarrow -1$               ▷ Module with max degree of interconnections
    $output_{max} \leftarrow -1$                           ▷ Max degree of interconnections

    ▷ Execution
    for $id_{element}$ in $units$ do
        $id_{partition} \leftarrow partitions[id_{element}]$
        if $id_{partition}$ not in $modules_{nodes}$ then                ▷ Define a new entry
            $modules_{nodes}[id_{partition}] \leftarrow$
            $\{position : -1, size : 0, outputs : 0, nodes : list(), modules : list()\}$
        end if
        $module \leftarrow modules_{nodes}[id_{partition}]$
        $module[nodes]$ add $id_{element}$
        $module[size] \leftarrow module[size] + 1$
        ▷ For each units check all its connected nodes
        for $id_{connection}$ in $connections[id_{element}]$ do
            $idconn_{partition} \leftarrow partitions[id_{connection}]$
            $module[modules][idconn_{partition}] += 1$          ▷ Count number of ties
            if $id_{partition}$ != $idconn_{partition}$ then
                $module[outputs] \leftarrow module[outputs] + 1$
                if $id_{partition} \geq idconn_{partition}$ then   ▷ Avoiding edge duplication
                    $modules_{edges}$ add $(idconn_{partition}, id_{partition})$
                else
                    $modules_{edges}$ add $(id_{partition}, idconn_{partition})$
                end if
            end if
        end for
        if $module[outputs] > output_{max}$ then            ▷ Max connections update
            $output_{max} \leftarrow module[outputs]$
            $module_{max} \leftarrow id_{partition}$
        end if
    end for
end procedure
```

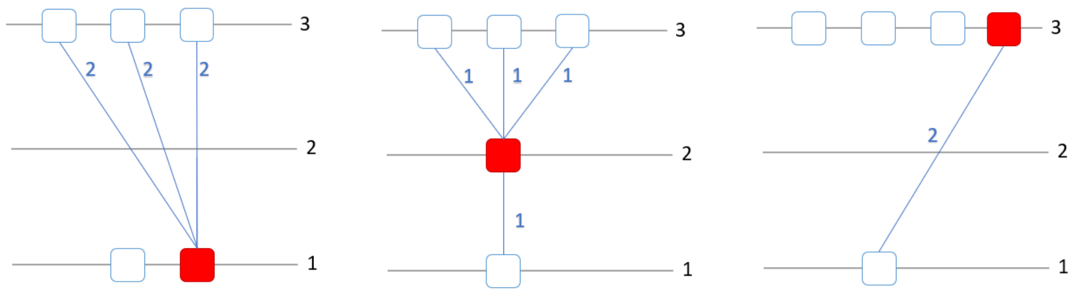
Algorithm 4 BFS vertical algorithm. Initial placement of the modules within the tiers.

```
procedure BFS VERTICAL($module_{max}$, $modules$, $layersize$)
    ▷ $module_{max}$ ID of the module with max number of output connections
    ▷ $modules$ Modules information generated in the preprocessing
    ▷ $layersize$ Max number of elements that can be placed into a tier
    $graph \leftarrow dict()$        ▷ For each module the IDs list of connected modules
    $visited \leftarrow list()$                      ▷ List of visited modules during BFS
    $queue \leftarrow list()$                        ▷ List of module to visit during BFS
    $bestpath \leftarrow list()$                       ▷ Best placement path for modules
    $tiersize \leftarrow list()$       ▷ Number of elements in each tiers initially set to 0
    $z \leftarrow 0$                                                      ▷ ID first tier

    ▷ Execution
    for $k, v$ in $modules$ do
        $graph[k] \leftarrow sort(v[modules])$   ▷ Sorting on the number of connections
    end for
    $visited$ add $module_{max}$
    $queue$ add $module_{max}$
    while $queue$ not $empty$ do                                ▷ Start BFS algorithm
        $s \leftarrow queue.pop()$
        $bestpath$ add $s$
        for $neighbour$ in $graph[s]$ do
            if $neighbour$ not in $visited$ then
                $countconnections \leftarrow modules[s][modules][neighbour]$
                if $countconnections$ != 0 then ▷ Check presence of connections
                    $visited$ add $neighbour$
                    $queue$ add $neighbour$
                end if
            end if
        end for
    end while
    for $k$ in $bestpath$ do                          ▷ Assign to each module a tier
        $mod \leftarrow modules[k]$
        if $tiersize[z] + mod[size] > layersize$ then               ▷ Check tier size
            $z \leftarrow z + 1$
        end if
        $mod[position] \leftarrow z$                                       ▷ Set tier
        $tiersize[z] \leftarrow iersize[z] + mod[size]$                ▷ Update tier size
    end for
end procedure
```

Vertical placement begins on the lower tier. Following the *bestpath* sequence, modules are allocated in the tier as long as there is sufficient space. Each time a module is placed the *tiersize*, containing for each layer the number of units, is updated. When the space is exhausted, the placement continues on the above layer. While the size of each module is defined by the number of units contained and is computed in the pre-processing phase, the size of the layer is derived from the initial 2D IC. The constraint on the number of elements placeable into a single tier is equal to the number of units plus a 15%, divided by the number of layers. 15% more units are considered because it is useful to have an extra margin of space to facilitate the movement of the modules in the TSV optimization phase. At the end of the BFS vertical phase all the module generated form the Modularization phase are assigned to a specific tier.

4.1.3 TSV optimization

The BFS vertical phase define an initial disposition where all the partition are allocated into a specific layer of the final 3D IC. This process takes in consideration the space constraints and prioritize the placement on the same tier of those modules with high level of interconnections. The only flaw of this method is that using the BFS algorithm to define the best sequence with which to place our modules, does not necessarily lead us to the optimal solution. This means that there still some unnecessary TSVs that can be removed. The TSV optimization phase is defined to overcome this problem. The goal of the Place and Route algorithm third step is still focused on the TSV minimization. Starting from the initial disposition obtained by the BFS vertical phase, The TSV optimization try to reduce the number of TSVs by moving modules in the different layers. Moving a module between two tiers is performed only if it involves a reduction in the number of TSVs and does not violate spatial constraints. This phase is highly affected by the choice of the modules' size and the tiers' area. Having modules of not excessive dimensions facilitates their movement because they require less space to be placed. On the other hand, modules composed by several units need larger area to be hosted, this makes TSV optimization difficult due to lack of space. As explained in the previous paragraph, to further ease movement between layers, the tiers' area is computed from the number of units increased of 15%. This makes space constraints less strict. The TSVs minimization is based on the computation of a cost parameter for each module-tier combination. This variable defines the number of TSVs needed to place a module in a specific layer, considering also their length and is computed as follows. For each module M and tier T, the list of partitions with at least a connection with M that are not placed on T are computed and stored in the *connected* structure. The cost for placing M in T is equal to the sum of the distance between the tiers multiplied by the number of connections, calculated for each module contained in *connected*. While the number of connections between two modules is equal to the number of connections between the units belonging to those two partitions, the tiers' distance is the absolute difference between the tiers' height. By computing the cost associated with each tier, it is possible to identify the layer that bring less TSVs, so the best vertical placement for the module.



(a) Tier 1; cost 6. **(b)** Tier 2; cost 4. **(c)** Tier 3; cost 2.

Figure 4.3: Cost computation on a 3D IC with three tiers

Once the cost strategy has been defined, it must be applied to all the present modules to identify the exchanges necessary to minimize the total number of TSVs. Initially, the $switch_{vals}$ structure is generated. This structure store for each partition all the information required for selecting which module it should be moved and where, in particulars the variables used are:

- **initialval:** This value represents the initial cost associated with the tier define by the BFS vertical phase. It is used for the computation of the *gain*.

- **gain:** His value is equal to the difference between *initialval* and the cost of moving the module on another layer. It represents the reduction or the increase of the number of TSVs.
- **layer:** ID that define for a movement the destination tier.
- **cost:** List of values that store the cost associated with each tier.
- **size:** Value defining the number of units contained inside the module. It is used for checking the space constraint.
- **movable:** Boolean value that define if the module can be moved or not.

Algorithm 5 TSV cost algorithm. Computes the number of interconnections between tiers.

```
procedure TSVCOST($m$, $layer$, $modules$, $nummodules$)
    ▷ $m$ Module ID on which the cost is computed
    ▷ $layer$ The main module tier
    ▷ $modules$ Modules information generated in the preprocessing
    ▷ $nummodules$ Total numbers of modules
    $cost \leftarrow 0$                                   ▷ Initial cost set to 0

    ▷ Execution
    for $i$ in $nummodules$ do
        ▷ Number of connections between main module and i-module
        $connections \leftarrow modules[m][modules][i]$
        if $i! = m$ AND $connections! = 0$ then
            $cost \leftarrow cost + abs(layer - modules[i][position]) * connections$
        end if
    end for
    $Return(cost)$
end procedure
```

Once the $switch_{vals}$ content is initialized, the main loop start. For each module characterized by *movable* equal to TRUE, the cost associated with moving them on all the different layers is computed. The results are stored in *cost* structure. Each tier is identified by an index that assume value from 0 to the number of

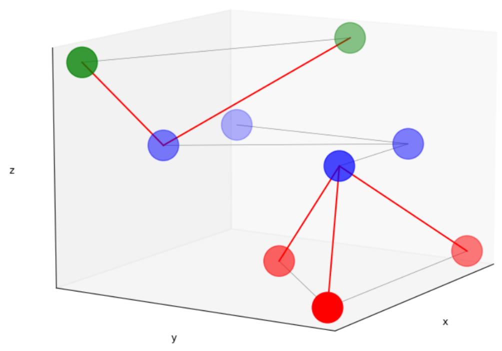
Algorithm 6 Max gain ID algorithm. Finds module with higher gain.

```
procedure MAXGAINID($switch_{vals}$)
    ▷ $switch_{vals}$ For each module contain the information for move it
    $gain_{max} \leftarrow -1$                ▷ The max gain values reach by modules
    $id_{max} \leftarrow -1$                  ▷ Module ID with higher gain value

    ▷ Execution
    for $k, v$ in $switch_{vals}$ do
        ▷ Check the gain value and mobility of the module
        if $v[gain] > gain_{max}$ AND $v[movable] = TRUE$ then
            $gain_{max} \leftarrow v[gain]$
            $id_{max} \leftarrow k$
        end if
    end for
    $Return(id_{max})$
end procedure
```

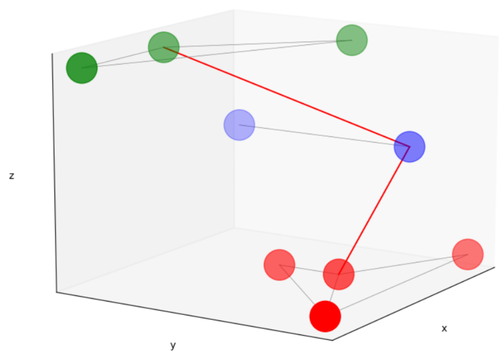
tiers (*numtiers*) minus one. The layer that brings the lower cost become the best tier for the considered module and its index is saved in *layer* variable. Then, the *gain* is computed by subtracting to *initialval* the new cost associated with the *layer* index. If *gain* is equal to 0 it means that the module is already on its best vertical placement, consequently there are no reasons for moving it. To block its position, *movable* is set to FALSE. Once these variables have been updated for all the modules present, all the information needed to make a move will have been defined. The process to change the layer of a module is composed by two steps. The first step is the identification of the module that should be moved. This decision is made according to the *gain*. The module that brings the greatest reduction in the number of TSVs is also the one with the higher *gain*. Its selection is based on the exploration of $switch_{vals}$, where for each movable module is it check if its *gain* is the highest reached up to that moment. Once all partitions have been checked, id_{max} will identify the module with the greatest *gain*. If no module can be moved, so all the *movable* variables are equal to FALSE, the searching loop will return id_{max} with value -1. It means that all the modules that should be moved were already moved and the TSV optimization phase is over. The second step consists in moving the selected module from the initial tier (*oldlayer*) to its best tier (*newlayer*). If the size of *newlayer* can accommodate the module, then *movable* is set to FALSE and the transfer proceeds. Otherwise, among the available tiers, the one that involves the greatest *gain* and is also able to host the module is searched and become the *newlayer*. At the end of this searching if the *newlayer* is equal to *oldlayer* it means that no tiers are able to accommodate the module, so its layer does not change and *movable* is set to FALSE, otherwise we proceed with the latest *newlayer*. The transfer is carried out by updating the number of elements contained in the two tiers involved and by changing the position of the module in $modules_{nodes}$. Once a module has been moved, the loop restarts with all the values contained in $switch_{vals}$ updated. As previously explained, the TSV optimization phase continues until there are no more movable elements.

Algorithm 7 TSV optimization algorithm. Minimization of the number of TSVs by moving the modules between tiers.

```
procedure TSV OPTIMIZATION($modules$, $numtiers$, $tiersize$, $layersize$)
    ▷ $modules$ Modules information generated in the preprocessing
    ▷ $numtiers$ Number of tiers
    ▷ $tiersize$ Number of elements inside each tier
    ▷ $layersize$ Max number of elements that can be placed into a tier
    $switch_{vals} \leftarrow dict()$                    ▷ Information required for moving modules

    ▷ Execution
    for $k, v$ in $modules$ do
        $ini \leftarrow TSVcost(k, modules[k][position], modules, len(modules))$
        $switch_{vals}[k] \leftarrow \{initialval : ini, gain : 0, layer : -1, cost : list(), size : v[size], movable : TRUE\}$
    end for
    while $TRUE$ do
        for $k, module$ in $switch_{vals}$ do
            ▷ If the module has not yet been placed
            if $module[movable] = TRUE$ then
                $costs \leftarrow list()$
                ▷ Compute cost of moving the module into others tiers
                for $i$ in $numtiers$ do
                    $costs[i]$ add $TSVcost(k, i, modules, len(modules))$
                end for
                $bestlayer \leftarrow index(min(costs))$    ▷ Module bringing lower cost
                $module[layer] \leftarrow bestlayer$
                $module[cost] \leftarrow costs$
                $module[gain] \leftarrow module[initialval] - cost[bestlayer]$
                ▷ If gain is null then block the module
                **if** $module[gain] = 0$ **then** $module[movable] \leftarrow FALSE$
                **end if**
            **end if**
        **end for**
        $id_{max} \leftarrow MAXgainID(switch_{vals})$  ▷ Identify module with higher gain
        **if** $id_{max} = -1$ **then** $STOP$              ▷ Stop if no module is movable
        **end if**
        $module \leftarrow switch_{vals}[id_{max}]$
        $newlayer \leftarrow module[layer]$
        $oldlayer \leftarrow modules_{nodes}[id_{max}][position]$
        ▷ If there is space in the tier block the module
        **if** $tiersize[newlayer] + module[size] \leq layersize$ **then**
            $module[movable] \leftarrow FALSE$
        **else**
            ▷ Find the best tier with with available space
            **while** $tiersize[newlayer] + module[size] \geq layersize$ AND $newlayer! = oldlayer$ **do**
                $module[cost][newlayer] \leftarrow inf$
                $newlayer \leftarrow index(min(module[cost]))$
            **end while**
        **end if**
        **if** $newlayer = oldlayer$ **then** $module[movable] \leftarrow FALSE$
        **else**
            ▷ Move the module
            $tiersize[oldlayer] \leftarrow tiersize[oldlayer] - module[size]$
            $tiersize[newlayer] \leftarrow tiersize[newlayer] + module[size]$
            $modules_{nodes}[id_{max}][position] \leftarrow newlayer$
            $module[initialval] \leftarrow module[initialval] - module[gain]$
        **end if**
    **end while**
**end procedure**
```



(a) Before TSV optimization.



(b) After TSV optimization.

Figure 4.4: Example of TSVs minimization on a 3D IC with three tiers

4.1.4 2D placement

Once the first three steps of Place and Route algorithm are executed, the obtained 3D architecture define an intermediate solution compared to the one sought. In fact, placing each module on a specific layer, according to the TSVs minimization, led to fixing the vertical position for the units contained in the partitions. This means that each unit have their z coordinate defined, but to complete the 3D architecture it is needed to also set the x and y coordinates. The 2D placement phase is designed to solve this problem. Its goal consists in finding for each unit the best position inside the tier of belonging, minding several optimizations and constraints. Now that the objective of minimize the inter-tiers connections is achieved, the modules are no longer needed and the focus change on the single units. For this reason, this is the first step in which the modules graph representation of the circuit is forsaken to adopt a units graph representation. The main optimization this phase provide is the minimization of the wirelength on the tiers without inserting single point of failure, while the main constraints to care about are the TSVs placement and direction. During the 2D placement, if the interconnections between the units not belonging to the same layer are not considered, is possible to manage the tiers as independent circuits. For example, with an 3D architecture compose by three tiers is possible to apply to each layer the FT-TDP algorithm that provide best x and y coordinates for each unit. The FT-TDP algorithm works on a PDD file generated by a single layer, which contain all the units and interconnections related to that tier. Since the PDD file characterize the elements with a sequential identification starting from a value of 1, it is necessary to map the units to new IDs for each layer. The FT-TDP alone not achieve good results because it can place units linked by a TSV (belonging to different tiers) in very distant positions increasing the wirelength needed to make the connection. TSVs connect two tiers through vertical holes in the silicon, for this reason they cannot directly link two units that are not on top of each other. Once the TSV tie two levels, horizontal links will reach the affected units allowing the communication. For this reason, is important to manage in a proper way the TSVs position because they will affect the total wirelength. Moreover, due to their technological characteristics the TSVs allow slow communications compared to the vias inside the same layer, for this reason they can create critical path. To identify the best location for the TSVs two strategies are defined. For each node to place, the centroid strategy compute the centroid of the connected units that belong to a different tier. In geometry the centroid is defined as the arithmetic mean positions of all the points in a figure. Computing the centroid allow to define the TSV's position that minimize the average distance with all the connected units, and so also the wirelength. On the other hand, the vertical strategy compute again the centroid of the connected units, but then place the TSV above the connected element closest to the centroid coordinates. In this way the inter-tiers link add zero horizontal wirelength with the selected unit. These two strategies are both used and their results are compared to identify the best one, if present. Since it is not possible to deal independently with the layers for the placement of units, an approach must be defined that take in considerations the relations and the inter-tiers constraints.

The 2D placement algorithm allocate the units starting from the bottom tier because it contains the elements with the higher connection density. Once the first layer is managed, it moves on the next ones. The placement within the tiers following the first is more complex because these contain links with nodes belonging to the lower layers that must be managed. Before starting the main process all the units belonging to a tier are mapped with a sequential ID (seq_{id}). For each tier seq_{id} is initialized to 1 and increase unit by unit. The correlation

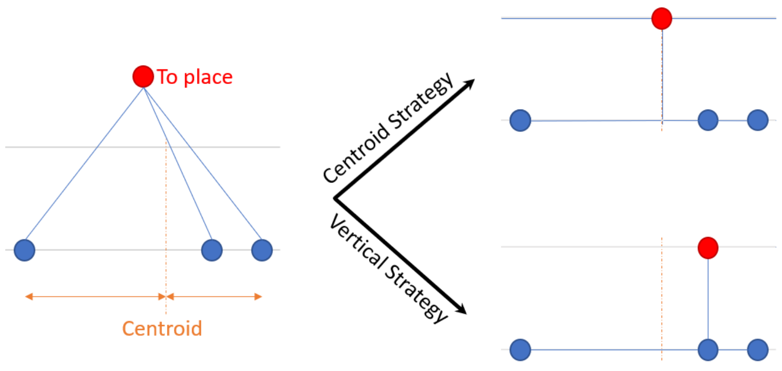


Figure 4.5: Placement strategies for node with TSVs connections

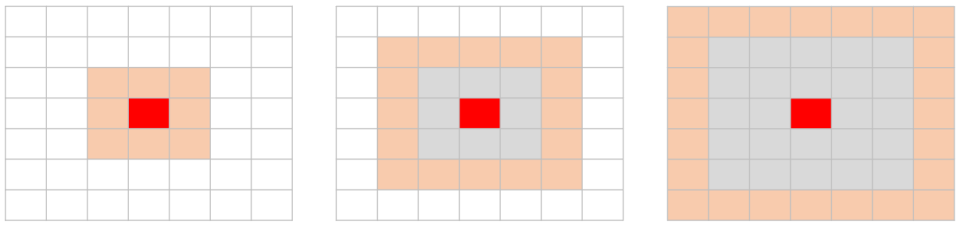
between real ID and mapped ID, essential to restore the initial circuit, is stored in the $tiers_{mapping}$ structure.

Algorithm 8 Centroid algorithm. Computes the centroid of a group of nodes.

```
 1: procedure CENTROID($info_{nodes}$, $below_{connections}$)
 2:     ▷ $info_{nodes}$ Dictionary containing units' information
 3:     ▷ $below_{connections}$ List of connected units ID
 4:     $nconnections \leftarrow len(below_{connections})$        ▷ Number of connected units
 5:     $sumx \leftarrow 0$        ▷ Sum of the x coordinate of the connected elements
 6:     $sumy \leftarrow 0$        ▷ Sum of the y coordinate of the connected elements
 7:
 8:     ▷ Execution
 9:     for $node_{id}$ in $below_{connections}$ do
10:         $nearx \leftarrow info_{nodes}[node_{id}][position].x$        ▷ Unit's X coordinate
11:         $neary \leftarrow info_{nodes}[node_{id}][position].y$        ▷ Unit's Y coordinate
12:         $sumx \leftarrow sumx + nearx$
13:         $sumy \leftarrow sumy + neary$
14:     end for
15:     $x \leftarrow sumx/nconnections$
16:     $y \leftarrow sumy/nconnections$
17:     $Return(x, y)$
18: end procedure
```

Following the IDs mapping, the main cycle begin and, one at a time, all the tiers are managed. Starting from the first layer, the algorithm generate the PDD file.

The structure of the PDD file is the same as shown at the beginning of the chapter, it will therefore be necessary to insert the number of units in the first line and write all the necessary information for each node. Since each PDD file represents a single tier, all interconnections with units not belonging to the current layer are not reported. Even if they are not written to the file, these external connections are very important because they define the dependencies between the tiers and affect the arrangement of the units in the layers above the current one. For this reason, the $above_{connections}$ structure is defined. It is a dictionary that contain for each unit that belong to the upper tiers the list of connected units belonging to the current layer. This structure is used during the placement of the next layer to compute the best position for those nodes that have links with the tiers below. Once the PDD generation is terminated, the FT-TDP algorithm is called. This script require four parameters to work: PDD input path, PDD output path, tier width and height. In our case the tiers have a square shape, so their dimensions are both equal to $layeredge$. FT-TDP returns as a result a PDD file containing for each unit the x and y coordinates that minimize the tier's wirelength. To conclude the first-tier units' placement, the obtained PDD file is read, and the final coordinates of each unit are stores in $positions_{final}$ structure. Once the first layer has been managed, we can move on to the next ones. As previously anticipated, the positioning becomes more complex from here on, because the inter-tiers connections saved in $above_{connections}$ must be managed. Also in this case, a PDD file representing the units and the interconnections of the new layer is generated. Now instead of having all the units free to be placed anywhere on the tier, the presence of links with the below layer creates some constraints. All the units contained in $above_{connections}$ are sorted according to the number of interconnections with the below layer. If the strategy adopted is the centroid one, for each unit of the current tier contained in $above_{connections}$ the centroid of the list of nodes connected to it is computed. The centroid defines for the unit to place, the position that minimize the average distance with the connected nodes of the below layer. The centroid coordinates are used as possible position to allocate the node. Otherwise, if the strategy adopted is the vertical one, from the list of the connected units the coordinates closest to the centroid position are used as possible allocation for the node. Once one of the two strategies is applied, the result will be a pair of coordinates. If the defined position is not available, a subroutine will search for the closest free cell to the resulting one. In particular, the search starts from the cells that are only one block away from the initial one. In case there are no free positions the acceptable distance is increased by one block and the searching restart. This process continues until a free position is reached. Once the final coordinates (x, y) are defined, the corresponding cell is calculated as $y * layeredge + x$. The position is finally added to $used_{positions}$ structure that store the not available blocks.



(a) Acceptable distance 1 Block. **(b)** Acceptable distance 2 Blocks. **(c)** Acceptable distance 3 Blocks.

Figure 4.6: Searching path for free cell

Once their coordinates are fixed, the units characterised by links with below tiers should not be moved, not even by the FT-TDP algorithm. But the FT-TDP must place all the remaining units characterise only by connections with nodes on the same tier. To manage it, the FT-TDP source code has been modified to accept the presence of fixed nodes that must not be moved during the execution of the algorithm. A new parameter defines a path to a file that contain all the fixed units ID with the corresponding x and y coordinates. When the FT-TDP is called, it will set the positions of the fixed units, then it will run the placement algorithm on the remaining ones. In this way the wirelength will be optimized on the current tier, while the links with the lower layer are managed in a proper way. This process is repeated for all the remaining tiers, and in each step it manages the reduction of the wirelength and the vertical constraints. Once the units of the last layer are placed, $positions_{final}$ contains the definitive x,y and z coordinates of all the units that composed the initial 2D PDD file. To conclude the 3D architecture the only remaining nodes to be placed are the pins.

Algorithm 9 2D placement algorithm. Defines the best x and y coordinates for all the units.

```
procedure 2DPLACEMENT($tiers_{nodes}$, $info_{nodes}$, $layeredge$, $strategy$)
    ▷ $tiers_{nodes}$ Dictionary containing for each tiers its units
    ▷ $info_{nodes}$ Dictionary containing units' information
    ▷ $layeredge$ Size of the tiers' edges
    ▷ $strategy$ Allocation strategy (centroid or vertical)
    $tiers_{mapping} \leftarrow dict()$                ▷ Units' IDs mapping for each tier
    $above_{connections} \leftarrow dict()$     ▷ Save for each unit the links with upper tiers
    $positions_{final} \leftarrow dict()$        ▷ Contain the final coordinates of each unit

    ▷ Execution
    for $tier_{id}, nodes$ in $tiers_{nodes}$ do
        if $tier_{id}$ not in $tiers_{mapping}$ then
            ▷ Initialize the mapping structure for the tier
            $tiers_{mapping}[tier_{id}] \leftarrow \{true_{aus} : dict(), aus_{true} : dict()\}$
        end if
        $seq_{id} \leftarrow 1$                        ▷ Sequential ID used on each tier
        for $node$ in $nodes$ do
            $tiers_{mapping}[tier_{id}][true_{aus}][node] \leftarrow seq_{id}$      ▷ From true to aus ID
            $tiers_{mapping}[tier_{id}][aus_{true}][seq_{id}] \leftarrow node$      ▷ From aus to true ID
            $seq_{id} \leftarrow seq_{id} + 1$
        end for
    end for
    ▷ Main cycle
    for $tier_{id}, nodes$ in $tiers_{nodes}$ do
        ▷ Generate PDD with mapped IDs
        $f \leftarrow$ open($tier_{id}$+".PDD")
        $f.write(len(nodes))$                                ▷ Numbe of units
        for $node$ in $nodes$ do
            $f.write(info_{nodes}[node])$                    ▷ Node's information
            for $conn_{node}$ in $info_{nodes}[node][connections]$ do
                if $conn_{node}$ not in $nodes$ then
                    ▷ If the node belong to another tier store the connection
                    if $conn_{node}$ not in $above_{connections}$ then
                        $above_{connections} \leftarrow list()$
                    end if
                    $above_{connections}[conn_{node}]$ add $node$
                end if
            end for
        end for
```

```
40:         $f.close()$
41:         if $tier_{id} = 0$ then
42:             ▷ Run directly the FT-TDP algorithm
43:             FT-TDP($tier_{id}$+".PDD", $tiert_{id}$+"out.PDD", $layeredge$, $layeredge$)
44:         else
45:             ▷ Sort according to descending number of links
46:             $above_{connections} \leftarrow sort(above_{connections})$
47:             $used_{positions} \leftarrow set()$          ▷ Initialize set of unavailable positions
48:             ▷ Write the units with fixed positions
49:             $f \leftarrow$ open($tiert_{id}$+"fix.PDD")
50:             for $node, below_{connections}$ in $above_{connections}$ do
51:                 $z \leftarrow info_{nodes}[node][position].z)$
52:                 $x, y \leftarrow Centroid(info_{nodes}, below_{connections})$
53:                 if $strategy =$ "$vertical$" then
54:                     ▷ Apply vertical strategy if required
55:                     $x, y \leftarrow$ Closest coordinates to $x, y$ form $below_{connections}$ units
56:                 end if
57:                 $pos \leftarrow y * layeredge + x$       ▷ Compute corresponding position
58:                 if $pos$ in $used_{positions}$ then
59:                     ▷ Find closest free cell
60:                     $x, y, pos \leftarrow$ Closest position $x, y$ not yet occupied
61:                 end if
62:                 $used_{positions}$ add $pos$
63:                 $f.write(tiers_{mapping}[tier_{id}][true_{aus}][node], x, y)$
64:             end for
65:             $f.close()$
66:             FT-TDP($tier_{id}$+".PDD", $tiert_{id}$+"out.PDD", $tiert_{id}$+"fix.PDD",
    $layeredge$, $layeredge$)
67:         end if
68:         $f \leftarrow$ open($tiert_{id}$+"out.PDD")
69:         $id, x, y = f.read()$
70:         $realid \leftarrow tiers_{mapping}[tier_{id}][aus_{true}][id]$
71:         $positions_{final}[realid] \leftarrow (x, y, z)$        ▷ Save definitive coordinates
72:     end for
73: end procedure
```

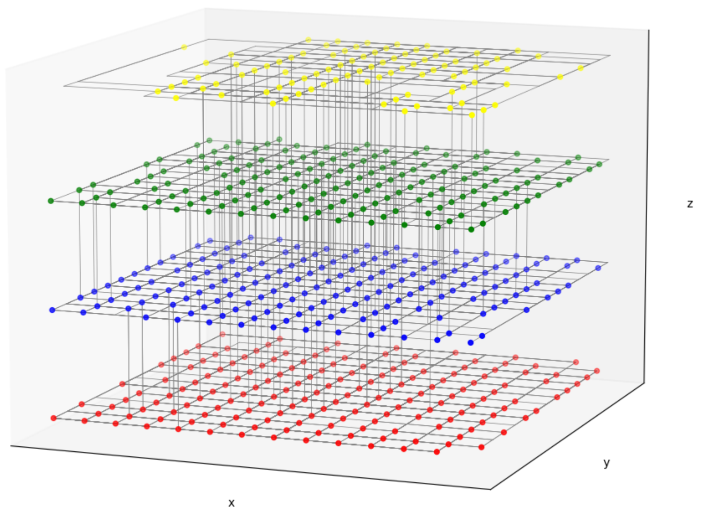


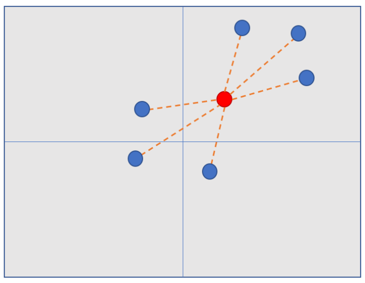
Figure 4.7: Example of 2D placement result on a 4-tiers architecture

4.1.5 Pins placement

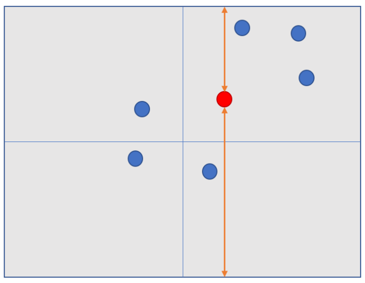
The pins allocation is the last phase of the algorithm. Once all the units are placed on the different tiers, and their position cannot change, the optimization process focus on the pins. The pins are the terminals of the IC, used to provide power supply and I/O interconnections with other circuits. In the Modularization phase these components have been set aside to favour units management, now they are instead treated to conclude the placement process that define the 3D IC. The pins placement is based on two constraints derived from architectural standards. Since the pins are the components used to connect an IC with other systems and circuits, they can only be placed on the outermost layers of the 3D IC. The architecture, in this case, is composed of a stack of tiers, so the eligible layers are the first or the last one. All the pins in this way are contained in a single tier which will be the one used as a base for the connection with an additional element. This allows to simplify the creation of interconnections between different systems. By default, the algorithm places the pins in the first layer, but the user can set also the last one. There are better performing procedures in which the I/O pins are partitioned and positioned on all tiers[22]. These strategies involve short wirelength but require a compatible system to establish a connection. To make the circuit adaptable to most situations, the positioning of the pins is carried out following the classic strategy based on a single tier. The other constraint derived from the selected pins packaging, that define where the pins should be placed into the tier. In the project the pins are allocated by default on the top and bottom edge of the layer, following an approach similar to the DIP (Dual In-line Package)[23]. This procedure is widely used for many IC related products, thus giving our architecture strong adaptability and usability. Is possible to use also another packaging approach where all the four edges of the layer are used for the pins placement. Once the tiers selection and the packaging strategy are defined, the Pins placement phase must satisfy all their constraints. Manage in a proper way the pins and their position is quite important for the performance of the 3D IC. Pins can bring two kinds of connections, pin-to-pin that in our case are harmless because they take place on the same layer, and pin-to-unit that can generate inter-tiers links. Incorrect management of the pins allocation can lead to slowdowns and higher complexity, especially in circuits with a small number of units. For this reason, the main goal of this phase is defining a strategy for placing the pins in a way that the wirelength is minimized, and the circuit performance does not drop dramatically. The algorithm proposed below reach these goals always remaining compliant with the architectural constraints.

To reduce the wirelength, the algorithm finds the position that minimize the distance between pin and all its connected elements. Starting from the *pins* structure, that contains the required IDs, the centroid of each pin is computed. The centroid coordinates are computed by summing the x (*sumx*) and y (*sumy*) coordinates of all the connected node, then dividing them by the number of elements. During the summation it is necessary to pay attention to the presence of connections with other pins that have not yet been positioned. In this case the coordinates should not be considered for the centroid computation, otherwise they would lead to wrong positions. To recognize a pin not yet positioned just read its coordinates, if these are still the same as the default ones (0,0) we must not consider it. The order in which the pins are placed and managed is also important. Is possible to have pins with most or all their connections made with other pins. This implies that their allocation will have to be managed only after that of the other pins, otherwise their final position would be conditioned only by the few units already placed. If this problem of order is not handled it can also break the code with an infinite loop where two connected pins are one waiting for the positioning of the other. To avoid those complication, the first step of this phase orders the IDs inside *pins* list according to the number of ties with other nodes. Doing so, the pins characterized by a higher number of connections with placed units will be managed before critical pins. Once the centroid coordinates are defined, is it computed which is the closest edge between the top and bottom one. If the top one is the closest the y coordinate of the centroid is changed to edge size ($layeredge$) plus padding. Otherwise, the y coordinate assumes a value equal to the negative padding. The padding is required to separate the units and the pins, it will increase the size of the initial tier. Next the shifting of the centroid on the closest edge, if the new position is available the pin is placed there, otherwise the closest free position is found. Starting from the shifted centroid, if its position has already been occupied, a recursive function checks, with a step initially set to 1, if the positions on its right and left are free.If one of this two locations are available it becomes the final position of the pin. Otherwise, the step is incremented by one until a free location is found. The $position_{pins}$ structure is used to keep track of all spots already used by pins. At the end of this phase for each pin an x and y coordinate will be define on the first layer of the 3D IC.

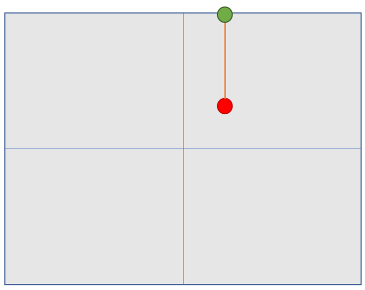
If all four edges of the first tier are used for pin allocation rather than just the top and bottom one, the algorithm changes a little. During the computation of the centroid closest edge, it is considered also the distance with the left and right edge. Furthermore, if in the search for a free position the new coordinates are closer to a different edge than the starting one, the search continues on the closest edge resetting the step to 1.



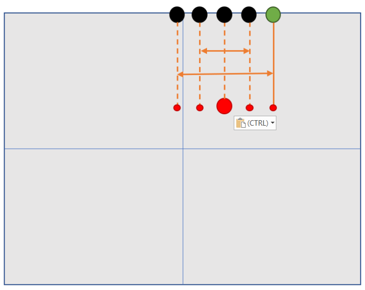
(a) Centroid computation.



(b) Closest edge identification.

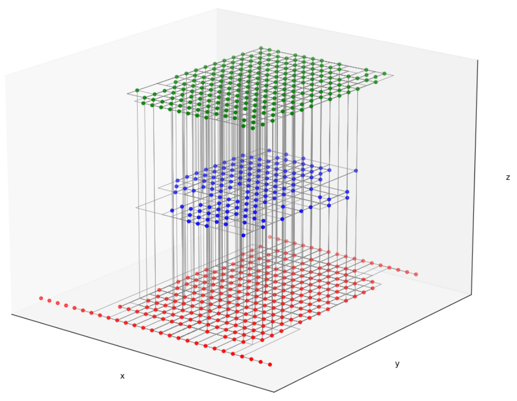


(c) Centroid shifting.

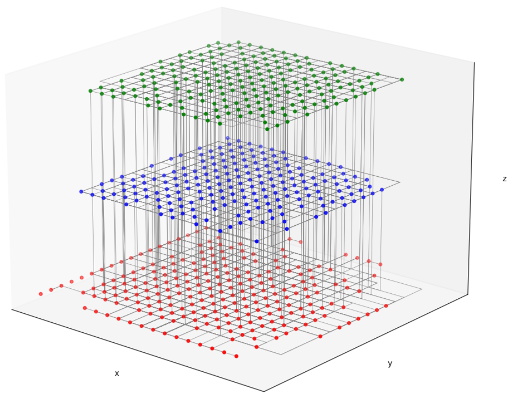


(d) Searching for an available location.

Figure 4.8: Pins placement process



(a) Top and Bottom edge.



(b) All four edges.

Figure 4.9: 3D IC after Pins placement phase

Algorithm 10 Pins placement algorithm. Defines the position of IC's pins.

```
procedure PINSPLACEMENT($pins$, $nodes$, $layeredge$)
    ▷ $pins$ List of pins' IDs
    ▷ $nodes$ List containing the information of all IC's components
    ▷ $layeredge$ Size of the tiers' edges
    $position_{pins} \leftarrow set()$        ▷ Set containing the already used positions

    ▷ Execution
    $pins \leftarrow sort(pins)$        ▷ Sort according to the number of their connections
    for $id$ in $pins$ do
        $x, y \leftarrow Centroid(id, nodes)$        ▷ Compute centroid
        $adder \leftarrow 1$        ▷ Step size for searching a feasible position
        $padding \leftarrow 2$        ▷ Pins' padding
        if $y > layeredge/2$ then        ▷ Find the closest edge of the tier
            $y \leftarrow layeredge + padding$
        else
            $y \leftarrow -padding$
        end if
        ▷ Check the closest left and right position until the placement is feasible
        while $TRUE$ do
            if $(x, y)$ in $position_{pins}$ then
                $x \leftarrow x + adder$
                if $adder > 0$ then
                    $adder \leftarrow (adder + 1) * -1$
                else
                    $adder \leftarrow (adder * -1) + 1$
                end if
            else
                $STOP$
            end if
        end while
        $nodes[position] = (x, y, 0)$        ▷ For pins the z coordinates is 0
        $position_{pins}$ add $(x, y)$
    end for
end procedure
```

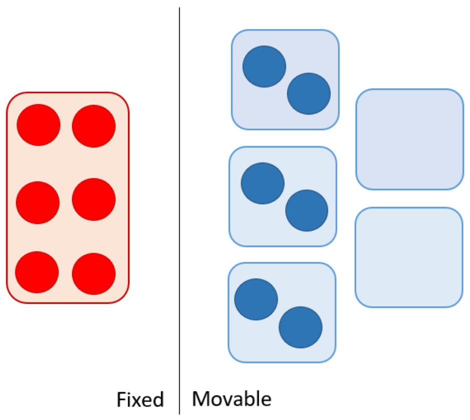
4.2 Fixed units case

The previous paragraph explains step by step the processes that compose the Place and Route algorithm. During the development, the main constraints to be careful are the limited area of each layer and the number of inter-tiers connections. In some cases, is possible to have additional limitations to care about. Until now the IC used were characterized by units free to be positioned and moved on each tier, but this is not always true. Many IC are composed by elements that should be placed in a specific positions or on a fixed tier. Components that produce a lot of heat, for example, need to be placed in strategic points so they do not affect the temperature of other units too much. One possible solution would be place the component on the higher tier, because its temperature affects the lower ones less. Sometime the architecture of an IC require that specific pair of elements should be placed close, or that entire blocks of units should be allocated on the same tiers, even if they have no connections between them. For example, the development of a CNN using a 3D IC is subject to those limitation. The CNN is composed by building blocks called DPU (Data Processing Units) that need to be interconnected following a specific logic. For this reason, its architecture is based on a topology that bond blocks in middle tiers. In these cases, fixing the position of these elements involves lower latency and higher energy efficiency.

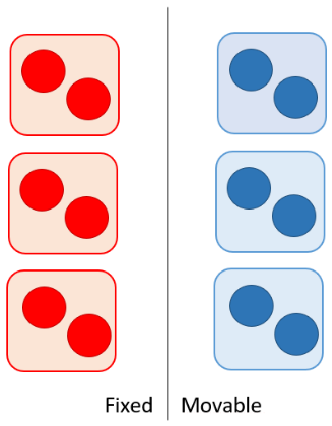
For the reasons explained above, management of units with default vertical location (tier of belonging) has been implemented in the Place and Route algorithm. The development of this feature involves new processes in the first three steps of the algorithm (Modularization, Vertical BFS, TSV optimization), while leaving the remaining two unchanged. The user can report the presence of units with fixed layer by passing as input parameter a file containing the list of names of the units concerned. By default, all these nodes will be fixed in the second tier starting from the bottom, but this option can be changed with any layer. This feature should only be used when it is strictly necessary because it involves a substantial increase in the number of TSVs. Blocking elements in certain positions leads to a lower efficiency of the hMETIS partition algorithm which now no longer has full control over all modules. Moreover, it also involves the reduction of the usable space to move modules between tiers. The implementation of this feature consists of two steps: The generation of fixed modules that groups the nodes to be placed into a specific tier; The anchorage where the final position is defined and cannot be changed by other processes.

4.2.1 Fixed modules generation

Starting from the input file containing all the fixed nodes, is generated a structure called $fixed_{nodes}$ that stores the names of the units that will placed in a specific tier. This structure is used during the reading of the 2D IC PDD file to identify the corresponding IDs and store them in the $fixed_{ids}$ structure. In the Modularization phase, to call the hMETIS script for partitioning the IC graph, is required to define the number of modules, the level of balance between modules and the hMETIS graph. With the presence of fixed units, a new file is required as input parameter. This new file is called $fixed_{graph}$ and describe for hMETIS algorithm which are the fixed units, and in which partition they should be placed. It is composed by a number of lines equal to the number of units, and each row contains a positive integer that represent the partition of belonging of the correspondent unit. If an element does not belong to a specific module, the value of its line is set to -1. Once the file is populated is possible to run the partitioning algorithm taking in consideration also the fixed vertical positions. A problem to be managed in this phase is the imbalance between the dimensions of the modules. If fixed nodes are many and they are all inserted in the same partition, the module created will be larger than all the others generated by the partitioning. hMETIS to balance the module with excessive dimensions will create several void modules, but their presence damages the performance of our processes because they force us to make useless calculations and lead to incorrect solutions. To solve this problem the average size of a module ($mean_{size}$) is computed as number of units over the number of modules. Later, while writing $fixed_{graph}$, the fixed nodes are initially associated with the partition with ID 0, but when it reaches a size equal to $mean_{size}$ the following fixed nodes are associated with the next partition. This process is repeated until all fixed nodes have been associated with a module. With this strategy, instead of creating a big module with all the fixed nodes, several modules with an average size are created. This removes the presence of empty modules because hMETIS is now working on partitions with similar sizes. The only flaw of this strategy is that to manage all the fixed units we must save all the IDs of the modules that contain them ($fixed_{partitions}$).



(a) Single fixed module.



(b) Multiple fixed modules.

Figure 4.10: Visualization of partitions with and without balancing of fixed nodes

Algorithm 11 Fixed modules generation algorithm. Subdivision of the fixed units in several balanced modules.

```
1: procedure FixedModulesGeneration($pdd$, $fixed_{name}$, $fixed_{graph}$, $nmodules$, $elements$)
2:     ▷ $pdd$ Location of the 2D PDD file
3:     ▷ $fixed_{name}$ Path to the file that contains fixed units names
4:     ▷ $fixed_{graph}$ Path to the file that will contains for each fixed unit its partition
5:     ▷ $nmodules$ Number of modules
6:     ▷ $elements$ List of units ID
7:     $fixed_{nodes} \leftarrow list()$        ▷ List containing all the name of fixed units
8:     $fixed_{ids} \leftarrow list()$        ▷ List containing all the IDs of fixed units
9:     $mean_{size} \leftarrow len(elements)/nmodules$        ▷ Average size of a module
10:    $id_{partition} \leftarrow 0$        ▷ ID of the current partition
11:    $size \leftarrow 0$        ▷ Size of the current partition
12:    $fixed_{partitions} \leftarrow list()$    ▷ IDs of the modules associated with fixed units
13:    $f \leftarrow$ open($fixed_{name}$)        ▷ Open the file in reading mode
14:
```

```
15:     ▷ Execution
16:     for $line$ in $f.read()$ do
17:         $fixed_{nodes}$ add $line.strip()$                    ▷ Store all the names
18:     end for
19:     ReadFile($pdd$)                                        ▷ Update $fixed_{ids}$
20:     $f.close()$
21:     $f \leftarrow$ open($fixed_{graph}$)                    ▷ Open the file in writing mode
22:     $fixed_{partitions}$ add $id_{partition}$                ▷ Save partition ID
23:     for $k, v$ in $elements$ do
24:         if $k$ in $fixed_{ids}$ then
25:             $f.write(id_{partition})$                       ▷ Set the partition of belonging
26:             $size \leftarrow size + 1$
27:             if $size = mean_{size}$ then
28:                 $size \leftarrow 0$
29:                 $id_{partition} \leftarrow id_{partition} + 1$   ▷ Switch to the next partition
30:                 $fixed_{partitions}$ add $id_{partition}$        ▷ Save partition ID
31:             end if
32:         else
33:             $f.write(-1)$                                   ▷ For the units not fixed
34:         end if
35:     end for
36:     $f.close()$
37:     hMETIS(..., $fixed_{graph}$)    ▷ Run the partitioning with the new parameter
38: end procedure
```

4.2.2 Anchorage

Once the fixed modules were defined, it is required to place them into the preselected tier that by default is equal to the second one. During the BFS vertical pre-processing, the fixed modules are managed like the others normal partitions. All the value of $module_{nodes}$ are computed as previously explained. Before the generation of the *bestpath* defined by the BFS algorithm, the vertical position of all the fixed modules is set to the preselected tier and the size of the layer is updated. To avoiding changing the placement later, the IDs representing the fixed modules are removed from the *bestpath* sequence. Now that the partitions are placed is required to not move the fixed ones form their original tier in the TSV optimization phase. For this reason, during the initialization of the $switch_{vals}$ structure where all the value necessary for moving a module are stored, the *movable* parameter is set to FALSE for each fixed module. In this way those partitions are ignored and are not moved even if it would bring a reduction in TSVs number.

4.3 Delays update

The 3D PDD file obtained by the algorithm is used to update the SDF (Standard Delay Format) file. As previously explained the SDF contains for each cell the delay and timing check. Each initial circuit on which the algorithm is executed is characterized by a 2D PDD file and a 2D SDF. The presence of TSVs in the 3D architecture involves the need to update the delays reported in the 2D SDF because, due to their technological characteristics, the TSVs allow a slower communication than the traditional one. If two nodes belonging to different layers are connected, the delay computation must now also consider the slowness of the inter-tier connection which can become a performance bottleneck. For this reason, starting from the 2D SDF, a 3D SDF is generated by updating the delay of all the interconnections between units that belong to different tiers and require a TSV to communicate. The 3D PDD file is essential for this process because it contains the information necessary to identify the presence of a connections between layers. In particular, if an edge is composed of two nodes characterized by different z coordinates, it implies that a TSV is required, and the delay must be updated. The additional delay that a TSV brings, can change in base of the used technology and architecture, in our case is fixed to 10ns. The process starts by reading the 3D PDD file and saving its information inside set_{nodes}. Between all the information stored into this structure is required to format the names of the nodes by removing some additional characters not supported by SDF. All the information associated to the formatted names are then stored in the $dictionary_{name}$ structure that will be used later. The first fifteen lines of the 2D SDF contains generic information about the version and format of the file and are copied to the 3D SDF. Then, the main loop starts and the 2D SDF file is read line by line. The process ends when is it reached the last line of the file characterized by a closing round parenthesis. For each element two main information are read, its type and its name, saved respectively in *celltype* variable and *instance* variable. From the $dictionary_{name}$ is it checked if the *instance* exist and its information are retrieved. If the *instance* type match the *celltype* variable we can proceed to check its interconnections. All connected components are identified by a unique character code and by the delay time. For each connected node, its layer of belonging is compared to the current instance element tier. If they are placed on the same layer the delay time is copied from the 2D SDF to 3D SDF without any change, otherwise the delay is incremented by 10ns. At the end of the process, the 3D SDF file contains all the delays of the interconnections, the value of which has been updated in case of TSV presence.

Algorithm 12 Delays update algorithm. 3D SDF file generation based on the update of the delays of the 2D SDF file.

```
1:  procedure DELAYSUPDATE($pdd3d$, $sdf2d$)
2:      ▷ $pdd3d$ Path to the 3D PDD file
3:      ▷ $sdf2d$ Path to the 2D SDF file
4:      $dictionary_{name} \leftarrow dict()$                ▷ Dictionary of formatted names
5:
6:      ▷ Execution
7:      $set_{nodes} \leftarrow ReadFile(pdd3d)$ ▷ Store nodes' information from the PDD file
8:      for $k, v$ in $set_{nodes}$ do
9:          $formatted_{name} \leftarrow v[name].replace('\\','')$        ▷ Chars removal
10:         $dictionary_{name}[formatted_{name}] \leftarrow v$
11:     end for
12:     $sdf_{2D} \leftarrow$ open($sdf2d$)
13:     $sdf_{3D} \leftarrow$ open("3d-sdf.sdf")
14:     for $i$ in $range(15)$ do                    ▷ Copy generic information
15:         $line \leftarrow sdf_{2D}.read()$
16:         $sdf_{3D}.write(line)$
17:     end for
18:     while $TRUE$ do                              ▷ Main reading and writing loop
19:         $line \leftarrow sdf_{2D}.read()$
20:         if $line =$ ")" then $STOP$                  ▷ Last line reached
21:         end if
22:         $sdf_{3D}.write(line)$
23:         $line \leftarrow sdf_{2D}.read()$
24:         $celltype \leftarrow line.split(" ")[1]$
```

```
25:         $sdf_{3D}.write(line)$
26:         $line \leftarrow sdf_{2D}.read()$
27:         $instance \leftarrow line.split(”")[1].split(”/")[0]$
28:         $sdf_{3D}.write(line)$
29:         $instance_{inf} \leftarrow dictionary_{name}.get(instance)$ ▷ Get the element instance
30:         if $instance_{inf} = -1$ then $ERROR$(name not found)
31:         else
32:             $line \leftarrow sdf_{2D}.read()$
33:             $sdf_{3D}.write(line)$
34:             ▷ If types not match the lines are copied
35:             if $instance_{inf}[labels][1]! = celltype$ then
36:                 while $line! =$ ”)" do
37:                     $line \leftarrow sdf_{2D}.read()$
38:                     $sdf_{3D}.write(line)$
39:                 end while
40:             else
41:                 while $line! =$ ”)" do
42:                     $line \leftarrow sdf_{2D}.read()$
43:                     $splitted_{line} \leftarrow line.split(”")$
44:                     $char_{ID} \leftarrow splitted_{line}[1]$
45:                     $time \leftarrow splitted_{line}[2]$
46:                     ▷ Find connected node id
47:                     for $connected$ in $instance_{inf}[connected_{e}lements]$ do
48:                         if $connected[1] = char_{ID}$ then $idv \leftarrow connected[0]$
49:                         end if
50:                     end for
51:                     $connected_{id} \leftarrow instance_{inf}[inputs][idv - 1]$
52:                     $layer1 \leftarrow instance_{inf}[position][2]$
53:                     $layer2 \leftarrow set_{nodes}[connected_{id}][position][2]$
54:                     if $layer1 = layer2$ then $sdf_{3D}.write(time)$          ▷ Same tier
55:                     else
56:                         $time \leftarrow time + 10$                          ▷ Delay update
57:                         $sdf_{3D}.write(time)$
58:                     end if
59:                 end while
60:             end if
61:         end if
62:     end while
63:     $sdf_{2D}.close()$
64:     $sdf_{3D}.close()$
65: end procedure
```

Chapter 5

Experimental result

Once all the steps of the algorithm have been performed, the result obtained is a PDD file containing all the main information about the components, including their 3D coordinates. To understand the performance of the new 3D architecture and compare them with the 2D IC, multiple tests are executed on different initial circuits. The result of these tests will show how many benefits we get from verticalization of a circuit, and what are the drawbacks. It will also be possible to understand how the performances are influenced by the main parameters and strategies defined in the previous chapters.

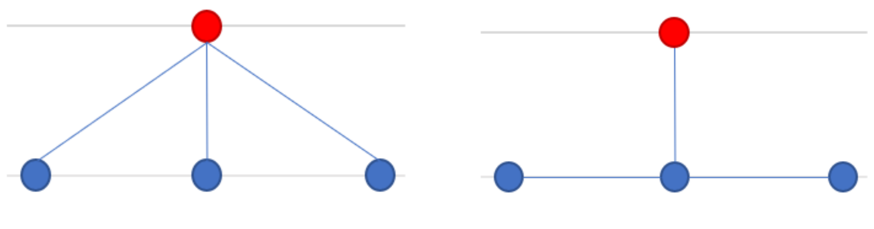
5.1 Benchmark

The algorithm is executed on four initial 2D IC: B09, B12, B14, CNN. These circuits are characterized by quite different number of elements. In this way is possible to understand how the algorithm behave with several graph size and interconnection complexity. Furthermore, the number of TSVs and the size of the silicon area required are expected to change as the initial circuit variate, so it becomes important understand how this affects the result. The tests start from B09, the less complex circuit characterized by 96 elements, and end with CNN that count 20362 nodes.

	B09	**B12**	**B14**	**CNN**
Nodes	96	620	4088	20362
Edges	486	3424	20240	102126

Table 5.1: Attributes of the tested circuits

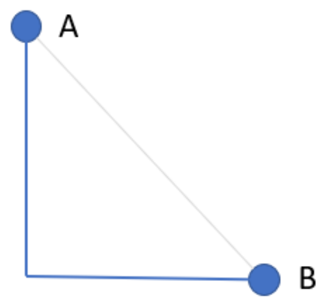
For each circuit, different tests are carried out in which the initial parameters and positioning strategies are changed. In this way, in addition to examining how the algorithm behaves on different types of IC, we can also understand how the variables and the used approach affects the results obtained on the same circuit. In particular, starting form a 2D IC, the algorithm is executed several times with an increasing number of tiers defined for the 3D architecture. The initial number of layers considered is equal to one and represent the 2D architecture. The maximum number of tiers tested is five, this quantity has been chosen because the construction of a circuit composed by a higher number of layers is not feasible due to thermal and architectural reasons. In some case is not even possible use five tiers due to the presence of long TSV. The long TSV are inter-tiers connections that bond together two units that belong to non-adjacent layers. For example, in a 3-tiers architecture, an edge that connect a node belonging to the first layer to a node that belong to the third layer is a long TSV. This kind of interconnection is illegal in the 3D architecture because each layer allowing the communication only with its adjacent tiers. This limitation derived from the TSVs structure that is based on physical holes in the silicon between two tiers. If the algorithm is applied with a high number of tiers to a 2D circuit characterized by a huge interconnections' density, sometime become hard or even impossible to avoiding the presence of long TSV. Consequently, the maximum number of tiers that can be reached will always be less than six but can be further reduced by the characteristics of the initial circuit used. In addition to the variation in the number of layers, the algorithm is executed using the two 2D placement strategies explained in the previous chapter: Centroid strategy and Vertical strategy. These two approaches have been defined to manage the placement of nodes on a layer considering possible interconnections with the lower tiers. While the Centroid strategy place the node in the centroid coordinates of the connected elements, the Vertical strategy place the node above the connected element closest to the centroid coordinates. By comparing these two approaches it will be possible to understand when one reach better performance than the other. For each combination of tiers quantity and placement strategy several attributes are computed from the resulting 3D PDD file. Each of which allows to identify the main properties of the new 3D circuit. All the attributes computed for each run are explained and listed below. The Area of the circuit represents the size of the silicon surface on which the elements are placed. It is computed from the maximum and minimum coordinates defined by the PDD file. The value of the area is defined in number of cells, which can contain a maximum of one component each. Depending on the technology used, the cells can assume different edge sizes which can assume value vary between 7nm and 130nm. Knowing how many cells are needed to build a layer allows us to compare the area of different circuits without having to define the technology used in advance. The Nodes and the Edges represent respectively the number of elements that make up the circuit (units and pins) and the number of interconnections between these components. The #TSVs is the real number of TSVs physically needed for managing all the inter-tiers connections. Its value is computed next the 2D placement phase, when the nodes are allocated according to their interconnections with the below tiers. If an element is connected to three units belonging to the layer below it, the number of edges based on TSV are three, but once the node is positioned the connection with the lower level will be made with only one TSV, while the connections with the three nodes will be managed through horizontal vias placed on the below plane.



(a) Edges based on TSV. **(b)** Number of TSVs.

Figure 5.1: Difference between edges base on TSV and number of TSVs

The Vertical wirelength represent the total length of the TSVs required for building the 3D IC. The computation of this variable derived by the length of a single TSV. The TSV can assume several sizes according to the used technology. In our case the TSV length is fixed to 10µm, but in particular case it can also reach 100µm[24]. The Vertical wirelength is computed by multiplying the #TSVs with the TSV's length. The Horizontal wirelength represent the total length of those vias that connect nodes belonging to the same tier. Are also considered the horizontal wire that connect a unit to a TSV that will be used to communicate with another tier. The Horizontal wirelength computation is based on the Manhattan distance, a geometric concept according to which the distance between two points is the sum of the absolute value of the differences of their coordinates. This value defines how close two connected nodes are, and so how good their placement is. The Avg wirelength define the typical length of a connections between two nodes, considering both the vertical and horizontal vias. It is computed by summing the vertical and horizontal wirelength and dividing the result by the number of edges. The Max wirelength represent the longest connection between two elements, it is very useful because can tell us which are the critical path that can become the bottleneck of our 3D IC. To understand the distribution of the Horizontal wirelength between the layers used for the verticalization of the circuit, a table show and compare the lengths of the cables of each tier for all the performed tests.



$$D(A,B) = |X_A - X_B| + |Y_A - Y_B|$$

Figure 5.2: Formula and graphical representation of the Manhattan distance

5.2 Results

The results obtained from the tests performed on the different initial circuits are reported in this paragraph. The tables below allow to make a direct comparison between the performances achieved on the various circuits and how these are affected by the initial parameters.

The B09 is the less complex circuit, is composed by 96 nodes that create a network of 486 interconnections. The silicon area of the 2D architecture is composed by 221 cells. As expected, by increasing the number of tiers for the 3D architecture, the required area decrease. Is equal to 49 cells with two layers and is reduced to 25 cells with four layers. It means that the required area can be reduced of 88% compared to the initial size. The #TSVs value increase with the number of tiers, this leads to a greater length of the vertical cable. This rise is foreseeable because the presence of more levels also involves the increase of interconnections between nodes not belonging to the same layer. The Horizontal wirelength of the B09 IC, developed on a single tier, is equal to 23874µm. Using only one additional tier this value drops drastically to 2834µm for the Centroid strategy and 2822µm for the Vertical strategy. The Max wirelength drop from 131µm to 19µm for both the approaches. The reduction of these distances derives from the small size of the tiers which force the nodes to stay close together. Increasing the number of layers shows a reduction in the Horizontal wirelength and the Max wirelength for both strategies. With circuits characterized by many interconnections, the presence of more tiers entails the fixing of the positions of many nodes due to the links with lower layers. Furthermore, having more TSVs leads to an increase in the total length of the cables used. Despite this, the reduction of the area of the tiers brings such a high advantage that the average wirelength decreases with the increase of the layers. The Centroid strategy and the Vertical strategy, with B09 IC, lead to similar solutions due to the simplicity of the circuit. The distribution of wirelength on the different tiers, as expected, is higher in the first layers. This behaviours derived from the fact that the BFS algorithm place the modules with higher interconnections density starting form the first layer. The B12 IC is composed by 620 nodes that create a network of 3424 edges. The initial area of the circuit, considering one tier, is equal to 3068 cells. By increasing the number of layers, like the previous case, the required area decreases from 361 cells with two tiers, to 144 cells with five layers. It means that by executing the algorithm the initial area can be reduced from the 88% to the 95%. Obviously, this big improvement is achieved with a price. The number of TSVs increase more respect to the B09 IC because now the total number of interconnections to manage are further. The ratio of #TSVs to the number of edges is greater for B09 than for B12. The initial Horizontal wirelength is equal to 117138µm with a maximum of 182µm. With two tiers their values become, respectively, 31820µm and 30µm for the Centroid strategy and 32490µm and 32µm for the Vertical strategy. A reduction not as drastic as that of B09 but still quite impressive. Also in this case, increasing the number of tiers lead to a lesser Horizontal wirelength, that remain always lower than the 2D initial value. For B12, the Vertical strategy lead all the time to a Horizontal wirelength greater than the one obtained with the Centroid strategy, while the Max wirelength still quite similar for both the approaches. The distribution of the wirelength, in this case, does not follow a decreasing trend when more than two layers are used. Although the first tier remain the one characterized by a greater wirelength, due to the high number of connections, is possible to have lower levels characterized by a total length of the cables shorter than the higher ones.

The B14 IC is composed by 4088 nodes that create a network of 20240 edges. Unlike the previous circuits B14 is more complex, and its initial area is equal to 23478 cells, almost eight times the initial size of B12. By increasing the number of tiers that are required for building the circuit, the size of the silicon area decrease. With two layers the area needed is equal to 2304 cells (more than a tenth of the initial one), while with four layers the area assumes the value of 1156 cells. With a greater number of edges, the #TSVs value increase compared to the previous result. Like the prior circuits, the value of the Horizontal wirelength decreases with the number of layers. The only difference is that the reduction brought about by adding each tier is greater than the previous cases. This is initially equal to 451096µm for Centroid strategy and 457126µm for Vertical strategy and drops respectively to 364216µm and 365602µm. This improvement derives from the fact that, in the presence of a large quantity of nodes, a high number of tiers does not affect the dispersion in a heavy way as with the two previous circuits. In addition, the number of nodes that must be blocked due to connections with lower layers does not negatively affect the positioning of the others, because the area of the tiers is now larger and offers a greater quantity of positions. For this reason, even if the number of TSVs increase and affect in a negative way the total length of the interconnections, the reduction of the area involves a greater advantage and therefore the Horizontal wirelength improve. Having many Interconnections help also to understand the main difference between the Centroid and Vertical strategies. In all the B14 tests, the Vertical strategy leads to a greater Avg wirelength than the Centroid strategy, even if sometimes the Max wirelength has a opposite trend. It means that critical paths are managed better using the centroids coordinates. Like the B09 IC, the wirelength distribution on the different layer follow a decreasing behaviour with the increase of the number of tiers. Moreover, the maximum wirelength reach by a single layer is lower when many levels are used for the verticalization of the IC respect to cases with few tiers. The last and most complex IC tested is CNN which is constituted of 20362 nodes and 102126 edges. The number of TSVs is quite high and reach a maximum of 9424 with five tiers. Although the circuit is composed of a number of elements that is five times that of B14, its initial area is quite small, equal to 24576 cells. For this reason, it is possible to note that the Horizontal wirelength computed with a number of tiers higher than one is worse than that of the equivalent circuit in 2D. The Horizontal wirelength is initially equal to 2703646µm and become 4230306µm for the Centroid strategy and 4254206µm for the Vertical strategy, with two tiers. The small initial area results in a low wirelength which gets worse when the circuit is divided into several tiers because node that initially were close can be placed into different layer. By increasing the number of tiers, the Horizontal wirelength value decrease, thanks to more restricted area, reaching a minimum of 3237038µm with five layers. Although Horizontal wirelength worsens compared to the 2D circuit, its maximum value is mitigated, albeit slightly. Also in this case, the Centroid strategy achieves better average wirelength than the Vertical strategy. Like the B12 IC, in CNN IC is possible to find below tiers with a total wirelength lower than upper tiers.

B09 IC Results											
						Centroid Strategy			Vertical Strategy		
Tiers	Area [cells]	Nodes	Edges	TSVs [#]	Vertical wirelength [μm]	Horizontal wirelength [μm]	Avg wire-length [μm]	Max wire-length [μm]	Horizontal wirelength [μm]	Avg wire-length [μm]	Max wire-length [μm]
1	221	96	486	0	0	23874	49.12	131	23874	49.12	131
2	49	96	486	50	500	2834	6.86	19	2822	6.84	19
3	36	96	486	80	800	2500	6.79	17	2510	6.81	17
4	25	96	486	104	1040	2146	6.56	16	2264	6.80	14

Table 5.2: Results obtained by running the algorithm on B09 IC

B09 IC Tiers' Horizontal Wire-length								
	Centroid Strategy				Vertical Strategy			
Tiers	Tier 1 [μm]	Tier 2 [μm]	Tier 3 [μm]	Tier 4 [μm]	Tier 1 [μm]	Tier 2 [μm]	Tier 3 [μm]	Tier 4 [μm]
1	23874	0	0	0	23874	0	0	0
2	2314	520	0	0	2310	512	0	0
3	2076	388	36	0	2100	374	36	0
4	1760	276	104	6	1856	270	128	10

Table 5.3: Comparison between the wire-lengths of the tiers used on B09 IC

B12 IC Results											
						Centroid Strategy			Vertical Strategy		
Tiers	Area [cells]	Nodes	Edges	TSVs [#]	Vertical wirelength [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]
1	3068	620	3424	0	0	117138	34.21	182	117138	34.21	182
2	361	620	3424	131	1310	31820	9.68	30	32490	9.87	32
3	256	620	3424	293	2930	27540	8.90	30	27882	9.00	28
4	196	620	3424	446	4460	25108	8.64	25	25568	8.77	24
5	144	620	3424	556	5560	22004	8.05	23	23182	8.39	25

Table 5.4: Results obtained by running the algorithm on B12 IC

B12 IC Tiers' Horizontal Wire-length										
	Centroid Strategy					Vertical Strategy				
Tiers	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]	Tier 5 [µm]	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]	Tier 5 [µm]
1	117138	0	0	0	0	117138	0	0	0	0
2	18246	13574	0	0	0	18376	14114	0	0	0
3	16104	8190	3246	0	0	16096	8462	3324	0	0
4	13936	4610	5274	1288	0	13644	5014	5522	1388	0
5	10886	3796	4368	2690	264	11142	4104	4732	2854	350

Table 5.5: Comparison between the wire-lengths of the tiers used on B12 IC

B14 IC Results

						Centroid Strategy			Vertical Strategy		
Tiers	Area [cells]	Nodes	Edges	TSVs [#]	Vertical wirelength [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]
1	23478	4088	20240	0	0	716756	35.41	312	716756	35.41	312
2	2304	4088	20240	1488	14880	451096	23.02	130	457126	23.32	131
3	1521	4088	20240	2670	26700	388596	20.52	112	395882	20.88	109
4	1156	4088	20240	3811	38110	364216	19.88	114	365602	19.95	109

Table 5.6: Results obtained by running the algorithm on B14 IC

B14 IC Tiers' Horizontal Wire-length

	Centroid Strategy				Vertical Strategy			
Tiers	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]
1	716756	0	0	0	716756	0	0	0
2	296814	154282	0	0	296716	160410	0	0
3	212148	142516	33932	0	211568	148048	36266	0
4	178768	105048	68616	11784	178446	107374	66058	13724

Table 5.7: Comparison between the wire-lengths of the tiers used on B14 IC

CNN IC Results											
						Centroid Strategy			Vertical Strategy		
Tiers	Area [cells]	Nodes	Edges	TSVs [#]	Vertical wirelength [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]	Horizontal wirelength [µm]	Avg wire-length [µm]	Max wire-length [µm]
1	24576	20362	102126	0	0	2703646	26.47	355	2703646	26.47	355
2	12100	20362	102126	2749	27490	4230306	41.69	239	4254206	41.92	247
3	8100	20362	102126	4915	49150	3739274	37.10	255	3755970	37.26	255
4	6084	20362	102126	7812	78120	3438880	34.44	223	3439938	34.45	248
5	4900	20362	102126	9424	94240	3237038	32.62	212	3257170	32.82	207

Table 5.8: Results obtained by running the algorithm on CNN IC

CNN IC Tiers' Horizontal Wire-length										
	Centroid Strategy					Vertical Strategy				
Tiers	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]	Tier 5 [µm]	Tier 1 [µm]	Tier 2 [µm]	Tier 3 [µm]	Tier 4 [µm]	Tier 5 [µm]
1	2703646	0	0	0	0	2703646	0	0	0	0
2	2840830	1389476	0	0	0	2848360	1405846	0	0	0
3	1895222	1137198	706854	0	0	1902098	1129574	724298	0	0
4	1429018	797278	909612	302972	0	1427448	795764	907794	308932	0
5	1236984	570676	624292	694434	110652	1243762	580010	626716	690200	116482

Table 5.9: Comparison between the wire-lengths of the tiers used on CNN IC

Chapter 6

Conclusion

From the results obtained it is possible to observe that the Place and Route algorithm achieve different performances depending on the complexity of the 2D IC. The silicon area on which the circuit is built is drastically reduced with the introduction of tiers and this applies to all four initial circuits. The higher the number of layers used, the more the required area can be shrink. The 3D ICs as expected, thanks to the vertical development allow to save space. The #TSVs used to build the networks of interconnections rise with the quantity of edges to manage and with the number of levels used, because both increase the presence of links between two tiers. The partitioning in modules allow to obtain excellent results because the ratio between the #TSVs and the edges is very low, in particular for more complex circuits such as CNN, where assumes value around 10% for five tiers. The reduced number of TSVs that characterize the 3D ICs allows to mitigates all the problems explained in the previous chapters, such as critical paths, performance bottlenecks and excessive vertical wirelength. The Centroid strategy and the Vertical strategy, used in the 2D placement phase to define the position of nodes having inter-tier connections, achieve very similar results with simple circuits. With more complex circuits such as B14 and CNN, even if the best Max wirelength is achieved alternately by both approaches, the Vertical strategy gets a higher Avg wirelength. In principle, it is therefore better to rely on the Centroid strategy which provides a lower average distance between two nodes. By using two tiers, the Horizontal wirelength is reduced compared to the initial 2D IC for all circuits except for CNN. This, being formed by a very small area already in 2D architecture, does not undergo a large reduction in surface compared to the other circuits and consequently its elements are not forced to be close as in the other cases. Therefore, the reduction or increase of the Horizontal wirelength is strictly linked to the characteristics of the initial circuit. Very interesting is the trend of the Horizontal wirelength with respect to the increase of the tiers considered. For simple circuits, such as B09 and B12, an increase in the number of layers entails a decrease in the Horizontal wirelength due to the area shrinking. This improvement is achieved even if, more TSVs mean more node that are placed in base of the connections with other layer, rather than according to the minimization of the total distances. For more complex circuits as B14 and CNN, the trend is exactly the same but the values achieved are higher. With large tiers, even if it is necessary to lock the position of many nodes, it is possible to optimize the wirelength thanks to the presence of more usable positions. Consequently, also in this case, the trend of the Horizontal wirelength will depend on the characteristics of the initial circuit. The wirelength distribution over the used tiers follow a similar behaviour for all ICs. The first layer contains, in all the cases, the higher wirelength value due to the presence of the elements characterized by huge interconnections density. As the number of levels increases, the wirelength distribution can follow two trends. A more linear one where its value decreases tier by tier. A non-linear one where, after the first layer, it is possible to find inferior levels with lower wirelength compared to superior ones. In conclusion, we can be satisfied with the results obtained because they achieve the set objectives: increase the integration density and minimize both the number of TSVs and the total length of the cable.

6.1 Future works

Starting from the Place and Route algorithm it is possible to introduce new strategies and improvements that allow the processes to base nodes allocation and tier choice on different possible objectives. For example, a variant can be defined to optimize energy consumption or to favour thermal dispersion. In this way it will be possible to introduce the 3D ICs to many new applications. Furthermore, the algorithm for now manages the connections between tiers using only the TSVs, but other strategies are emerging, such as MIVs (Monolithic Inter-tier Vias). It may be useful to allow the algorithm, depending on the context, to choose which is the best technology to use for the inter-tier connections. After all, 3D ICs are a new topic that big companies have only recently started working on, so I expect many new technologies and approaches will emerge in the coming years.

Bibliography

[1] Vachan Kumar and Azad Naeemi. «An overview of 3D integrated circuits». In: *2017 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization for RF, Microwave, and Terahertz Applications (NEMO)*. 2017, pp. 311–313. DOI: 10.1109/NEMO.2017.7964270 (cit. on p. 1).

[2] Guojie Luo. «Placement and Design Planning for 3D Integrated Circuits». PhD thesis. Univ. of California, 2011, pp. 1–24. URL: https://cadlab.cs.ucla.edu/news/dissertation_final.pdf (cit. on p. 7).

[3] Sandeep Kumar Samal, Deepak Nayak, Motoi Ichihashi, Srinivasa Banna, and Sung Kyu Lim. «Monolithic 3D IC vs. TSV-based 3D IC in 14nm FinFET technology». In: *2016 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. 2016, pp. 1–2. DOI: 10.1109/S3S.2016.7804405 (cit. on p. 8).

[4] Deepak Kumar Nayak, Srinivasa Banna, Sandeep Kumar Samal, and Sung Kyu Lim. «Power, performance, and cost comparisons of monolithic 3D ICs and TSV-based 3D ICs». In: *2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. 2015, pp. 1–2. DOI: 10.1109/S3S.2015.7333538 (cit. on p. 8).

[5] Max M. Shulaker, Tony F. Wu, Mohamed M. Sabry, Hai Wei, H.-S. Philip Wong, and Subhasish Mitra. «Monolithic 3D integration: A path from concept to reality». In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2015, pp. 1197–1202. DOI: 10.7873/DATE.2015.1111 (cit. on p. 9).

[6] Santo Papaleo. *Mechanical Reliability of open through silicon via structures for integrated circuits.* eng. Wien, 2016 (cit. on p. 11).

[7] Aarohi Desai. «2.5D and 3D ICs: New Paradigms in ASIC». In: *Einfochips Blog* (2017). URL: `https://www.einfochips.com/blog/2-5d-3d-ics-new-paradigms-in-asic` (cit. on p. 12).

[8] S. Muralikrishna and S. Sathyamurthy. «An overview of digital circuit design and PCB design guidelines - An EMC perspective». In: *2008 10th International Conference on Electromagnetic Interference Compatibility.* 2008, pp. 567–573 (cit. on p. 14).

[9] Pete M. Maurer. *Automatic routing of integrated circuit connections: a tutorial.* Tech. rep. Department of Computer Science and Engineering University of South Florida, Jan. 1990. DOI: `10.1109/ICC.1990.117186` (cit. on p. 14).

[10] Stephen J. Bigelow. *single point of failure (SPOF).* URL: `https://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF`. (accessed: 11.06.2021) (cit. on p. 17).

[11] Luca Sterpone. «A New Timing Driven Placement Algorithm for Dependable Circuits on SRAM-Based FPGAs». In: *ACM Trans. Reconfigurable Technol. Syst.* 4.1 (Dec. 2010). ISSN: 1936-7406. DOI: `10.1145/1857927.1857934` (cit. on p. 17).

[12] George Karypis. *Family of Graph and Hypergraph Partitioning Software.* URL: `http://glaros.dtc.umn.edu/gkhome/views/metis`. (accessed: 17.05.2021) (cit. on p. 18).

[13] B. W. Kernighan and S. Lin. «An efficient heuristic procedure for partitioning graphs». In: *The Bell System Technical Journal* 49.2 (1970), pp. 291–307. DOI: `10.1002/j.1538-7305.1970.tb01770.x` (cit. on p. 19).

[14] G. Karypis and V. Kumar. *hMETIS - A Hypergraph Partitioning Package.* Tech. rep. University of Minnesota, Department of Computer Science & Engineering, Nov. 1998 (cit. on p. 20).

[15] J. Cong, G. Luo, J. Wei, and Y. Zhang. «Thermal-Aware 3D IC Placement Via Transformation». In: *2007 Asia and South Pacific Design Automation Conference*. 2007, pp. 780–785. DOI: `10.1109/ASPDAC.2007.358084` (cit. on p. 22).

[16] S. Banerjee, S. Majumder, A. Varma, and D. K. Das. «A placement optimization technique for 3D IC». In: *2017 7th International Symposium on Embedded Computing and System Design (ISED)*. 2017, pp. 1–5. DOI: `10.1109/ISED.2017.8303930` (cit. on p. 25).

[17] V. Sh. Melikyan and A. G. Harutyunyan. «3D integrated circuits multifactor placement». In: *2017 IEEE East-West Design Test Symposium (EWDTS)*. 2017, pp. 1–4. DOI: `10.1109/EWDTS.2017.8110082` (cit. on p. 25).

[18] C. Ababei, Y. Feng, B. Goplen, Hushrav Mogal, T. Zhang, K. Bazargan, and S. Sachin. «Placement and routing in 3D integrated circuits». In: *IEEE Design Test of Computers* 22.6 (2005), pp. 520–531. DOI: `10.1109/MDT.2005.150` (cit. on p. 27).

[19] Microsemi Corporation. *Libero SoC v11.9 and earlier*. URL: `https://www.microsemi.com/product-directory/libero-soc/5507-libero-soc-v11-9-archive`. (accessed: 20.05.2021) (cit. on p. 30).

[20] H. T. Kung et al. «Systolic Building Block for Logic-on-Logic 3D-IC Implementations of Convolutional Neural Networks». In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, pp. 1–5. DOI: `10.1109/ISCAS.2019.8702753` (cit. on p. 32).

[21] Jason Holdsworth. «The Nature of Breadth-First Search». In: (Feb. 1999) (cit. on p. 39).

[22] R. Hentschke, S. Sawicki, M. Johann, and R. Reis. «A method for I/O pins partitioning targeting 3D VLSI circuits». In: vol. 249. Nov. 2007, pp. 259–279. ISBN: 978-0-387-74908-2. DOI: `10.1007/978-0-387-74909-9_15` (cit. on p. 57).

[23] Nikhil Agnihotri. «What are the different types of IC packages?» In: *Engineers-Garage* (2021). URL: `https://www.engineersgarage.com/what_is/ic-packages-types` (cit. on p. 58).

[24] Wen-Wei Shen and Kuan-Neng Chen. «Three-Dimensional Integrated Circuit (3D IC) Key Technology: Through-Silicon Via (TSV)». In: *Nanoscale Research Letters* 12 (Dec. 2017). DOI: `10.1186/s11671-017-1831-4` (cit. on p. 72).