MINIMIZING THE LAYOUT AREA OF

2-INPUT LOOK UP TABLES

by

Nafiul Hyder

Bachelor of Engineering in Electrical Engineering, Ryerson University, 2015

A thesis presented to Ryerson University in partial fulfillment of the requirements for the degree of Master of Applied Science in the program of Electrical & Computer Engineering

Toronto, Ontario, Canada, 2017

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

MINIMIZING THE LAYOUT AREA OF 2-INPUT LOOK UP TABLES

©Nafiul Hyder, 2017

Master of Applied Science

Department of Electrical & Computer Engineering

Ryerson University

Abstract

This work investigates the minimum layout area of multiplexers, a fundamental building block of Field-Programmable Gate Arrays (FPGAs). In particular, we investigate the minimum layout area of 4:1 multiplexers, which are the building blocks of 2-input Look-Up Tables (LUTs) and can be recursively used to build higher order LUTs and multiplexer-based routing switches. We observe that previous work routes all four data inputs of 4:1 multiplexers on a single metal layer resulting in a wiring-area-dominated layout. In this work, we explore the various transistor-level placement options for implementing the 4:1 multiplexers while routing multiplexer data inputs through multiple metal layers in order to reduce wiring area. Feasible placement options with their corresponding data input distributions are then routed using an automated maze router and the routing results are then further manually refined. Through this systematic approach, we identified three 4:1 multiplexer layouts that are smaller than the previously proposed layouts by 30% to 35%. In particular, two larger layouts of the three are only 33% to 45% larger than layout area predicted by the two widely used active area models from previous FPGA architectural studies, and the smallest of the three layouts is 1% to 11% larger than the layout area predicted by these models.

Acknowledgements

I would like to thank my supervisor, Professor Andy G. Ye for his technical advice and support. It would have been impossible to complete this work without his support.

I would like to thank Farheen Fatima Khan for laying the ground work by implementing several 2-LUT layouts. I would also like to thank her for all the Magic VLSI software support that she provided.

I would also like to thank Jim Koch and James Naughton for their technical support and advice.

I am extremely thankful to my family, especially to my parents for their support.

Finally, I would like to thank all Professors and colleagues whom I worked with in Ryerson University for the past six years, for creating a great learning environment.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction:

Field-Programmable Gate Arrays (FPGAs) are one of the most widely used programmable platforms for implementing digital computation. The flexibility of being able to program the chip after manufacturing gives FPGAs significant cost and ease-of-use advantages over Application Specific Integrated Circuits (ASICs) [1,2]. The fine-grain programmability of FPGAs, on the other hand, gives them a significant performance advantage over CPUs [1,2].

When designing a FPGA, one needs to quickly and accurately estimate the layout area of a given set of FPGA architectural parameters in order to select the best FPGA architecture for a given set of target applications [3]. To accurately estimate the layout area of a FPGA, one must accurately estimate the layout area of its fundamental building blocks, and two fundamental building blocks of FPGAs are Look-Up Tables (LUTs) that are used to implement FPGA logic and multiplexer-based routing switches that are used to provide programmable interconnect between the LUTs. Both LUTs and multiplexer-based routing switches are built upon multiplexers and they only differ in how the multiplexers are connected to inputs and configuration memory. In particular, as shown in Figure 1, a k-input LUT consists of a 2^k multiplexer whose data inputs are connected to the configuration memory while the select lines are connected to the LUT inputs. A multiplexer-based routing switch, on the other hand, uses a multiplexer whose select lines are connected to the configuration memory and whose data inputs are connected to the various routing tracks and/or LUT outputs.

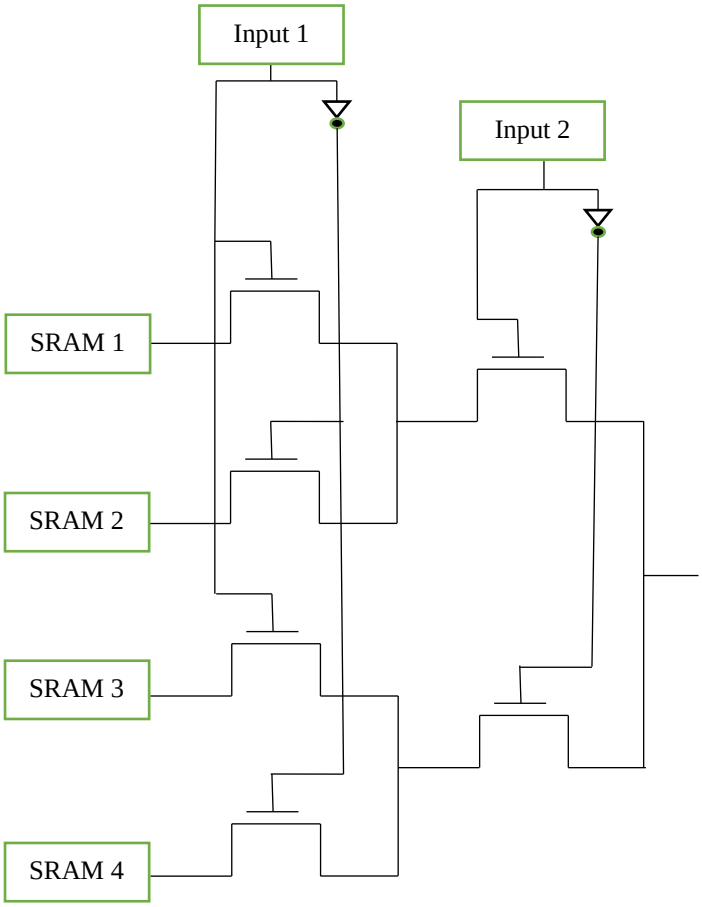


Figure 1: A 2-Input Look Up Table

Due to their multiplexer-based structures, higher order LUTs and multiplexer-based routing switches can be built simply by recursively connecting multiple smaller lower order circuits together. For example, a 4-input LUT, which utilizes a 16:1 multiplexer to select the configuration stored in one of the 16 bits of SRAM cells, can be built using two 3-input LUTs where each LUT is constructed out of an 8:1 multiplexer. Similarly, a 3-input LUT can be built using two 2-input LUTs by decomposing the 8:1 multiplexer into two 4:1 multiplexers. Therefore, the layout area of a 4:1 multiplexers can significant impact on the layout area of higher order LUTs and multiplexers.

We observe that when all four data inputs of a 4:1 multiplexer are transported on the same metal layer, the layout area of the 4:1 multiplexer is significantly dominated by its wiring area. Consequently, previous 2-input LUT layouts [4] are significantly larger than the layout area estimated by the traditional active-area-based area models [3][5]. As a result, the main goal of this work is to minimize the layout area of 4:1 multiplexers by significantly reducing their wiring area. Due to the significant use of LUTs and multiplexer-based routing switches in modern FPGAs [6], a successful reduction in 4:1 multiplexer layout area can have a significant impact on the overall layout area of FPGAs.

Note that there are a number of previous researches [3][4][5] that attempt to estimate the minimum layout area of LUTs and routing switches. In particular, the VPR area model [3] and the COFFE area model [5] are two widely used [7][8][9] models, which are described by simple equations. However, these models do not have any physical layout implementations. Therefore, it is not clear that how many metal layers are required to achieve the area given by the models. A more recent study in [4] implemented multiple LUT layouts and compare the layouts to these equation-based models. However, the 4:1 multiplexers used in the study transport all data inputs of the multiplexer on the same metal layer resulting in excessive wiring area. Moreover, the work has only explored a small number of transistor-level placement options for the 4:1 multiplexer. This work also has the limitation of only using three metal layers, whereas increasing the number of metal layers can potentially minimize the layout area.

Therefore, the purpose of this study is to provide an accurate estimation of the minimum layout area for 4:1 multiplexers by systematically search their transistor-level placements while matching

the pitch of multiplexer data inputs to the pitch of their active area through the use of infinite number of metal layers. In this process, an automated routing tool is used whose results are then manually refined. The minimum layout area achieved is then compared to the area prediction of previous area models and previous minimum area layouts.

The final result of this investigation, does not only give a layout area but also presents physical layouts of each placement discussed to verify how this area can be achieved. Moreover, it also states the number of metal layers needed to achieve this area. Furthermore, this work also implements physical layout of 2- input LUTs with larger transistors to show how the area changes with the change of the size of the transistors. Finally, we use 2-input LUT to build multiple higher order LUTs and present the physical layouts. These layouts not only verify the area but also denotes the number of metal layers required to achieve this area, which by far was not done by any of the previous models and layout work.

The remainder of this thesis is organized as follows: Chapter 2 presents a background review on FPGA architectures and also describes the previous related area estimation work in detail. Chapter 3 systematically searches the transistor-level placements for 4:1 multiplexers and picks the most suitable placements for further investigation. Chapter 4 describes the routing algorithm used in this study. Chapter 5 presents experimental results obtained by utilizing the automatic routing tool from Chapter 4 to route feasible transistor-level placement options identified in Chapter 3. Finally, chapter 6 concludes.

Chapter 2

Background and Motivation:

FPGA are a key digital circuit implementation media. They are pre-fabricated silicon devices that can be electrically programmed to carry out almost any digital logic. The designs of these circuit contain a mix of reconfigurable fabric and fix logic to maximize performance and minimize power consumption of their target applications. All FPGAs contain both programmable logic blocks and programmable routing. The nature of the logic blocks strongly influences a FPGA's speed and density [11]. Programmable logic block with its supporting programmable routing resources create a FPGA tile and these tiles are put in a matrix format to create a complete FPGA. This chapter initially reviews the general architecture of a FPGA and describes the functionality of different components within it. Then it moves on to discussing the importance of routing multiplexers and LUTs in FPGA architecture. Finally, it presents previous work that has been done to estimate the minimum layout area of LUT and routing multiplexers, which form the backbone of any FPGA architecture.

2.1Basic components of a FPGA:

FPGAs usually consist of an array of logic blocks which includes logic blocks, which contains a set of tightly connected LUTs, memory blocks, which provide distributed memory and multipliers, which provide arithmetic acceleration. These blocks are then connected by routing tracks. The entire array is then surrounded by input/output blocks which connects the chip with the outside

world [3]. Logic blocks are the most numerous amongst the three type of logic resources on FPGA. Consequently, in this work, we focus on the design of a homogeneous FPGA architecture which contain only logic blocks and the corresponding routing resources.

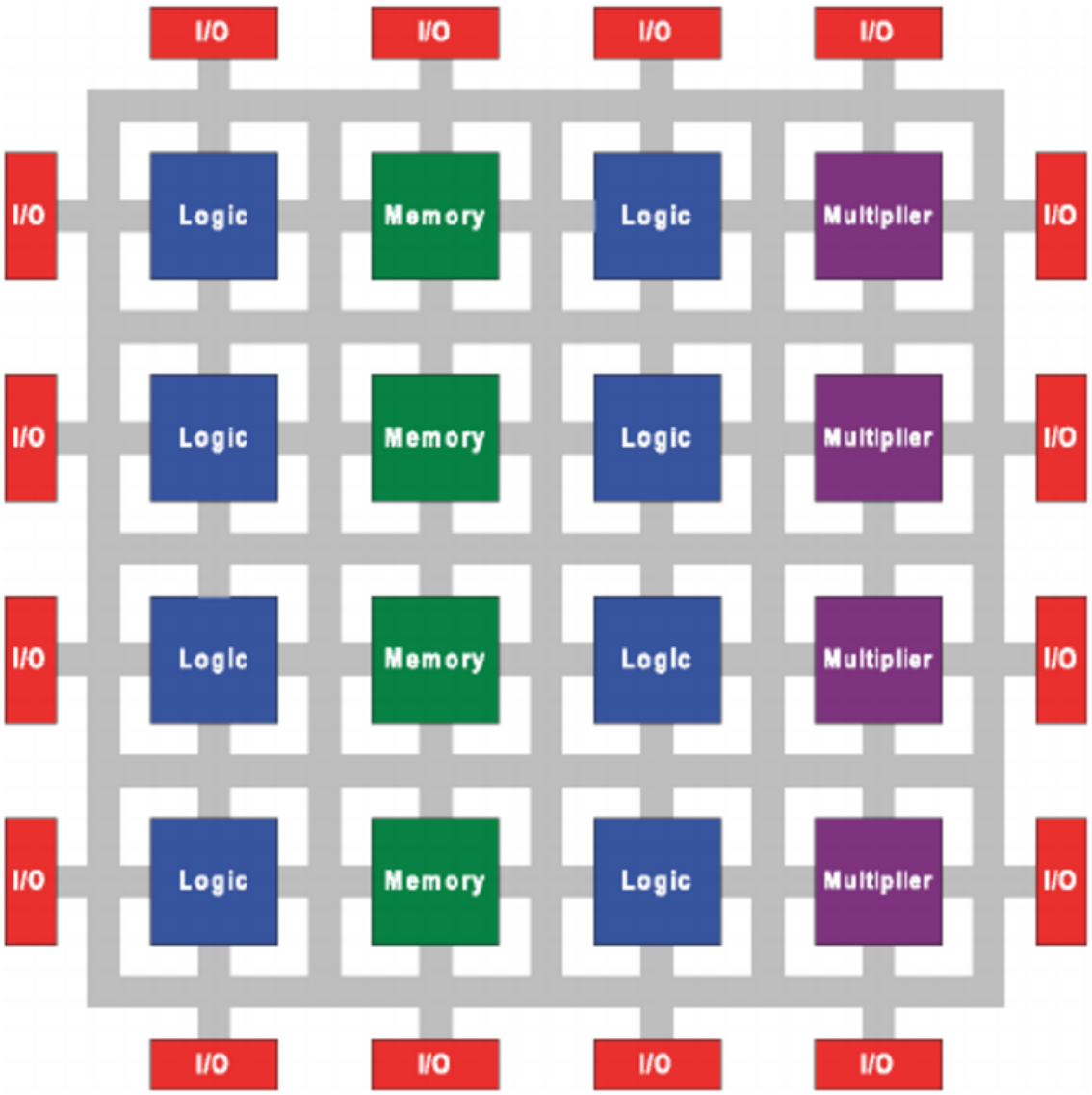


Figure 2: Basic FPGA Structure [3]

2.2 Logic Block Architecture:

The logic block architecture is extremely important because it has a dramatic effect on how much programmable routing is required [3]. The purpose of the logic block of a FPGA is to provide the basic computation and storage elements to be used in the digital logic system. The most simple and conventional way of providing these capabilities is to use a transistor as the most basic logic element and build the required gates and storage element using it from there on [3]. This kind of very fine-grained logic block, however, requires the use of large amounts of programmable interconnect to create any typical logic function [3]. It will generate a FPGA that will be area inefficient because it will have a lot of programmable routing which will cost a lot of area. Moreover, it will be very low in performance because there must be an individual routing track for each transistor which will make the number of routing tracks higher and therefore the signal propagation slower. Furthermore, FPGAs containing this kind of logic blocks will also consume very high static and dynamic power due to the excessive number of transistors that are required to implement a unit of logic. Therefore, FPGA architectures within the last two decades are logic block based which contain a combination of local routing multiplexers [12] and LUTs [10]. In addition to a basic logic block, many modern FPGAs also contain a heterogeneous mixture of different blocks, which are only be used for very specific functions, such as dedicated memory blocks or multipliers [3].

The most common approach that is being used in almost all the industrial FPGAs these days is to use clusters of LUTs and flip-flops to build a logic block. As shown by Figure 3(a) each LUT is first grouped with a flip-flop to form a Basic Logic Element (BLE). BLEs are then grouped together and connected by a local interconnect structure that are connected by multiplexer based routing switches. In general, there are fewer inputs to the logic cluster from the external inter-

cluster routing than the total number of inputs to the basic logic elements inside the logic cluster. This reduction is possible because each cluster input signals are often used as inputs to multiple BLEs within the cluster [3]. Figure 3(b) illustrates the basic structure of a logic cluster, where N is the number of BLEs in a cluster and I is the number of logic cluster inputs from inter-cluster routing.

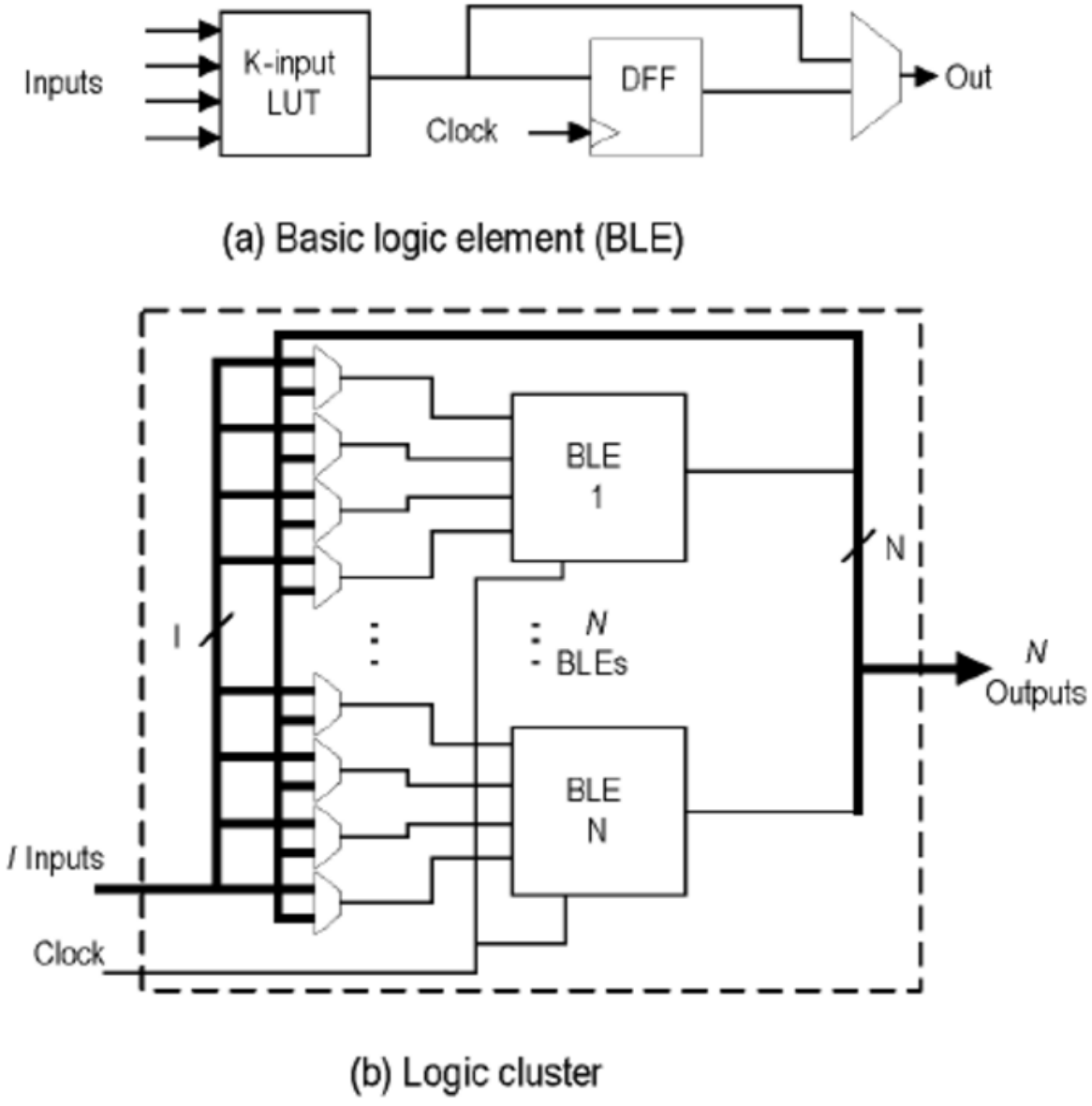


Figure 3: FPGA Basic Logic Elements and Cluster [3]

In general, published research on logic block architecture tends to model and explore relatively simple basic logic elements, such as the pure K-input lookup table. In contrast, commercial logic blocks have undergone an evolution that typically has led to the development of more complex blocks to gain more functionality [3].

For example, one of the earliest FPGAs, the Xilinx XC3000 FPGA [17] employed a complex logic block, which is a 5-input lookup table that uses an additional multiplexer, but it is augmented to allow the creation of two 4-input look up tables that share most of the inputs, as shown in Figure 4

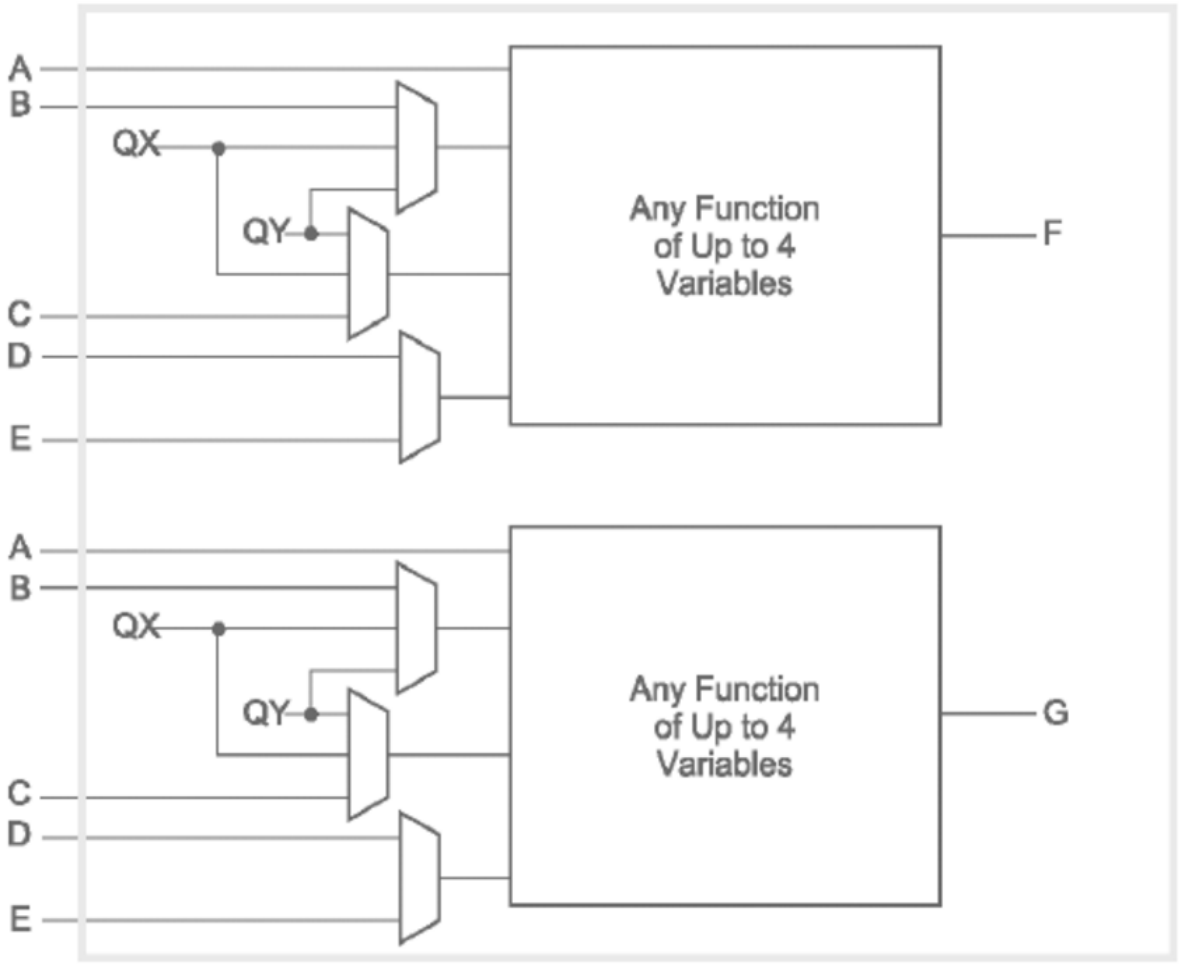


Figure 4: One possible configuration of a Xilinx 3000 logic block [3]

Similarly, the Altera Stratix family of FPGAs uses adaptive logic models as shown in Figure 5, which can programmably adapt its basic logic structure from a collection of 6-input LUTs to a collection of 3-input LUTs. Nevertheless, as shown in Figure 4 & 5 this programable adaptability of FPGA LUT sizes are also supported by a network of multiplexers. Consequently, the minimum layout area of multiplexers are extremely important in estimating the layout area of FPGA logic resources.

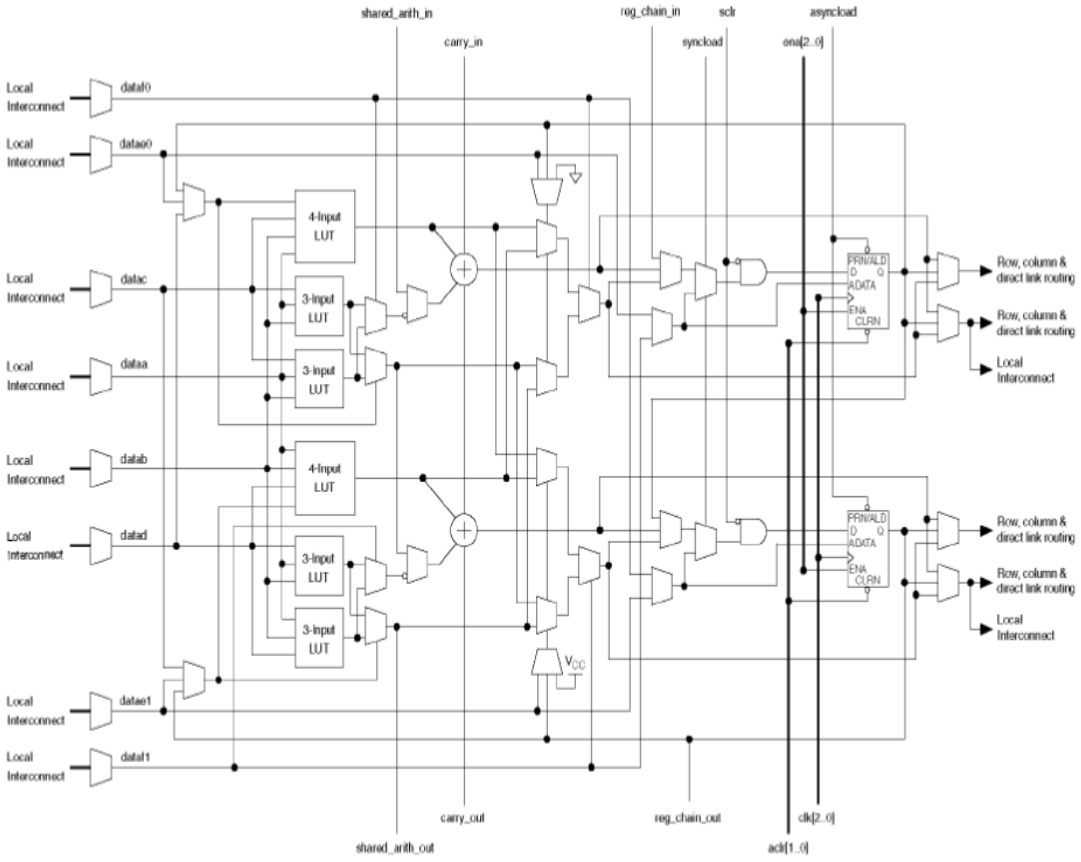


Figure 5: The Altera Stratix II adaptive logic module [16]

2.3 Routing Architecture:

The programmable routing in a FPGA provides connections among logic blocks and I/O blocks to complete a user designed circuit. It consists of wires and programmable switches that form the desired connections [3].

The routing architecture of a FPGA defines the relative position of routing channels in relation to the positioning of logic blocks, how each channel connects to other channels, and the number of wires in each channel. The detailed routing architecture specifies the lengths of the wires, and the specific switching quantity and patterns between and among wires and logic block pins [3]. FPGA routing architectures can be characterized as either hierarchical [17] or island-style [3,18].

Hierarchical routing architectures separate FPGA logic blocks into distinct groups [17,19]. Connections between logic blocks within a group can be made using wire segments at the lowest level of the routing hierarchy. Connections between logic blocks in distant groups require the traversal of one or more levels of the hierarchy of routing segments. Only one level of routing directly connects to the logic blocks. Programmable connections are represented with crosses and circles. Generally, the width of routing channels is widest at levels furthest from the logic blocks. [3]. A simple visual representation of this routing architecture is shown in Figure 6:

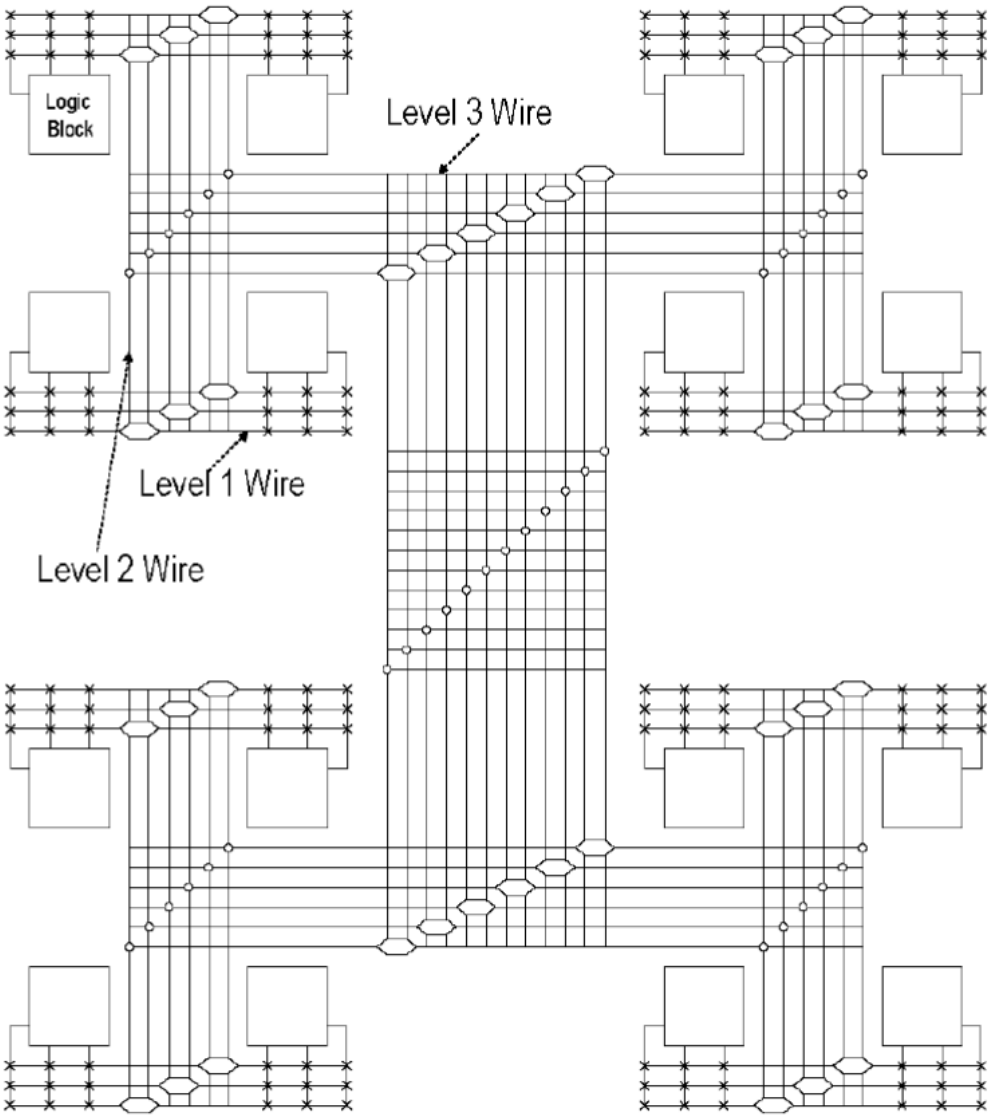


Figure 6: Example of Hierarchical FPGA [19]

In an island-style FPGA, logic blocks are arranged in a two-dimensional matrix with routing resources evenly distributed throughout the matrix. An island-style routing architecture typically has routing channels on all four sides of the logic blocks. The number of wires contained in a channel is pre-set during fabrication, and is one of the key choices made by the architect. Island-style routing architectures generally employ wire segments of different lengths in each channel in

an attempt to provide the most appropriate length for each given connection. They also typically stagger the starting point of the wire segments so that each logic block has a chance of connecting at the beginning of a wire of the most appropriate length. Currently, most commercial SRAM-based FPGA architectures [5,13, 20,21,22] use island-style architectures.

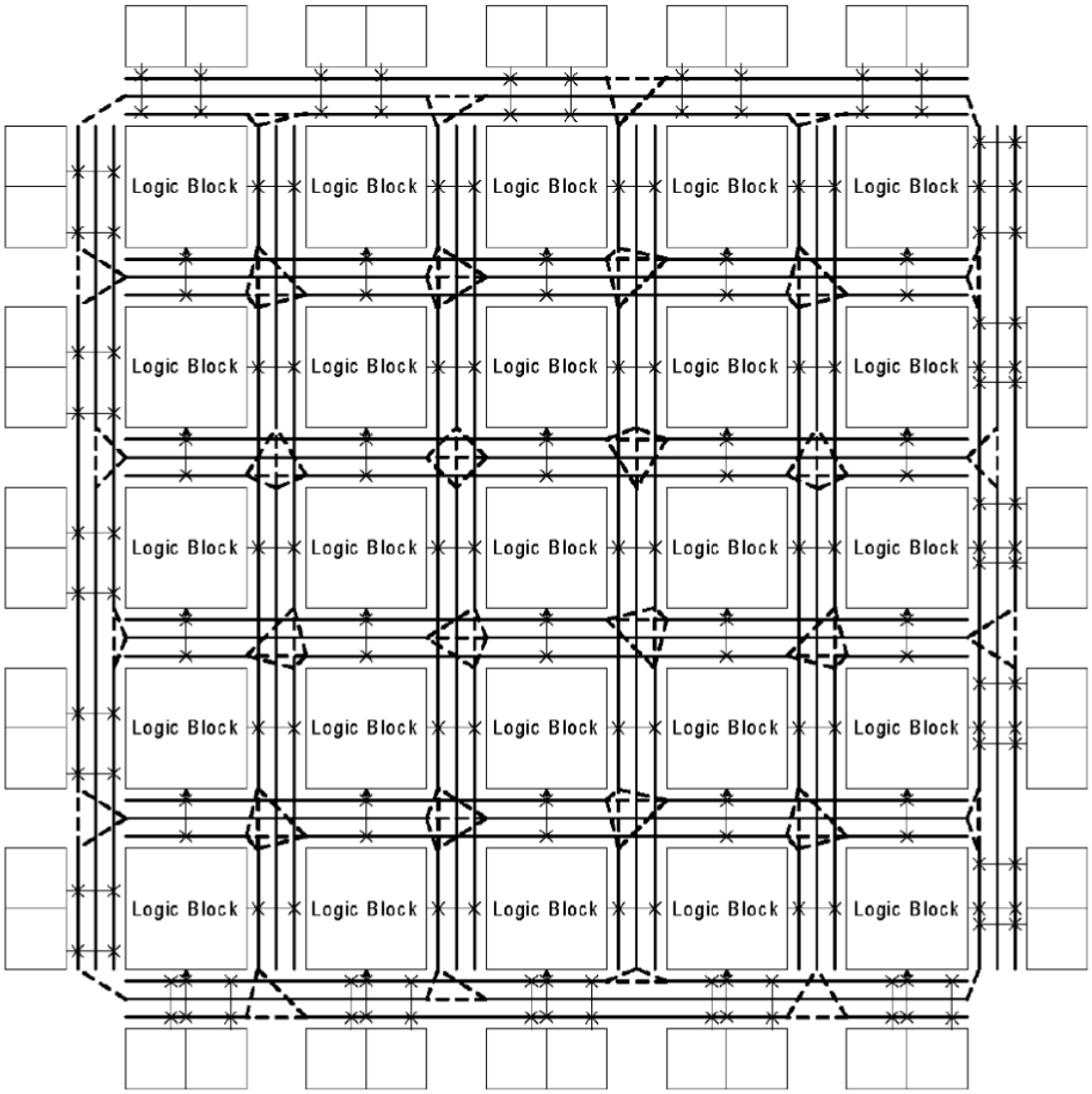


Figure 7: Island Style FPGA [3]

Switch boxes are the most essential component for both hierarchical and island-style routing architectures. Switch boxes can select different routing track or logic block outputs from neighboring logic blocks within the routing network depending on which two logic blocks are going to be electrically connected. The conventional way of designing a switch box is to use multiple multiplexers to select from several routing tracks. The number or the size of the multiplexer increases with the increasing number of routing tracks. However, multiplexers are always needed to implement the switch boxes. Figure 8 shows how one signal has to connect with multiple signals, depending on the requirement of the logic operation and using multiplexers is the easiest way of fulfilling this requirement.

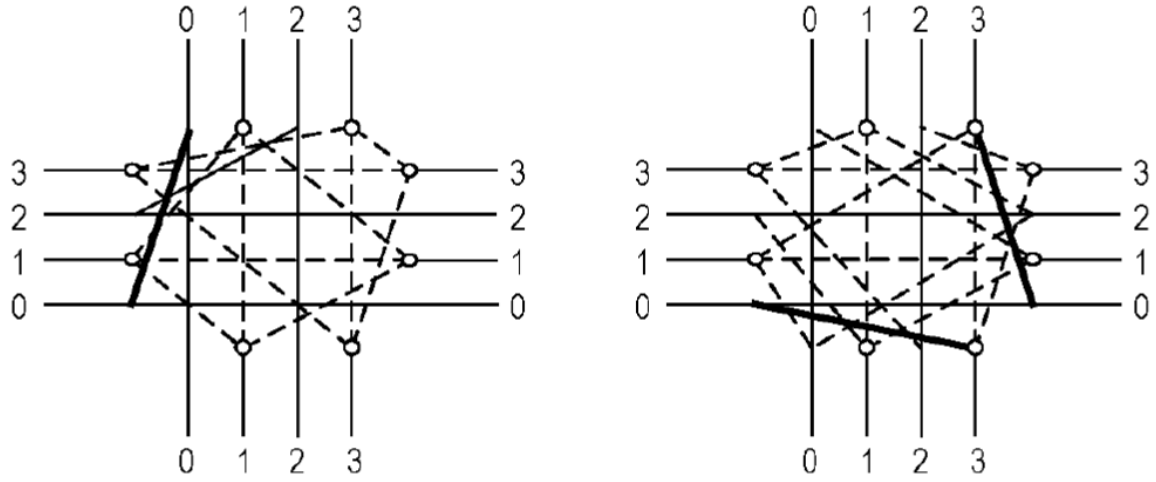


Figure 8: Different types of switch boxes [3]

The building characteristics of multiplexers are very similar to the LUT's. Any size of multiplexer can be created using multiple 4:1 multiplexer. Moreover, 4:1 multiplexer is identical to a 2-input LUT in terms of schematic. That is why, this investigation emphasizes heavily on estimating the layout area of a 2-input LUT.

2.4 Previous Work:

Previous work [3,4,7] on estimating the layout area of FPGA can be classified into two types. The work from [3] and [5] creates equation based models that are based on the theoretical minimum layout area for a single minimum width transistor [3,7]. The work in [2] is empirical based where, multiple layouts of LUTs are created and compared to the equation based models. Nevertheless, none of the previous work has systematically investigated the minimum layout area of FPGA LUTs and multiplexers.

The minimum width transistor area model [3,7] defines one unit of layout area as the area required to layout the smallest transistor. These transistors contain one contact for each source and drain diffusion area. Based on the lambda based rules scalable to a typical CMOS process [23], this layout area corresponds to a two-dimensional area, where the length of the transistor is calculated as 16λ and the width is 10λ. As the width of transistor, x, increases, this layout area increases based on the Equation 1[3], where x is the drive strength between the wider transistor and the minimum width transistor. The height, width and the number of metal layers that is required to achieve the predicted layout area are not considered [4].

$$Area\ (x) = 0.5 + 0.5x \qquad \textit{Equation 1}$$

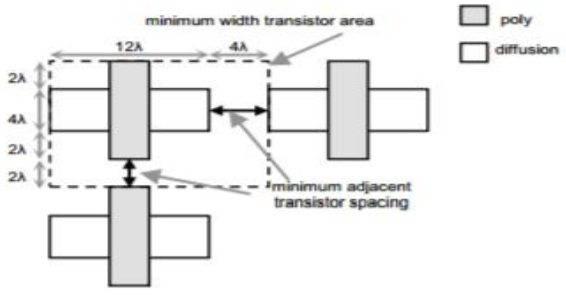


Figure 9: Minimum width transistor area model [3][4

]

A newer area model was described in [5], based on the individual transistor sizing of layouts. Equation 2 presents the area of the newer model. This area model includes different coefficients and the calculated area is based on the layout work of transistors with varying sizes. This model predicts smaller area when compared to the previous one. The minimum width transistor area for this area model corresponds to $154.5\lambda^2$ based on lambda based rules [26].

$$Area(x) = 0.447 + 0.128x + 0.391\sqrt{x} \quad \textit{Equation 2 [23]}$$

Both of these equation-based area models focuses on the layout area of a single transistor with minimum spacing between adjacent transistors. None of these models account for the wiring area which is required to connect adjacent transistors. However, determining accurate area estimation of any circuit requires considering of the area consumed by wiring.

A more recent work [4] implements physical layouts of different sizes of LUT multiplexers ranging from 2-input LUTs up to 6-input LUTs and compares the layout area with the above equation-based models and the work considers all the connectivity along with the spacing between adjacent transistors, in particular a mirroring technique as shown in Figure 10 is used to build larger LUTs from 2-input LUTs. The result of this investigation shows that the implemented layouts are at least 200% larger than the given theoretical model [4, 25]. From this result, we can conclude that how wiring is implemented can play a vital role in the accurate layout area of multiplexer based FPGA components. The work from [4,25], however, did not systematically explore ways to minimize the wiring area of LUTs and multiplexers. Moreover, the wiring is done manually and based on direct human observation. Therefore, the purpose of the current work is to minimize the layout area of 4:1 multiplexers used in constructing 2-input LUTs by minimizing their wiring area.

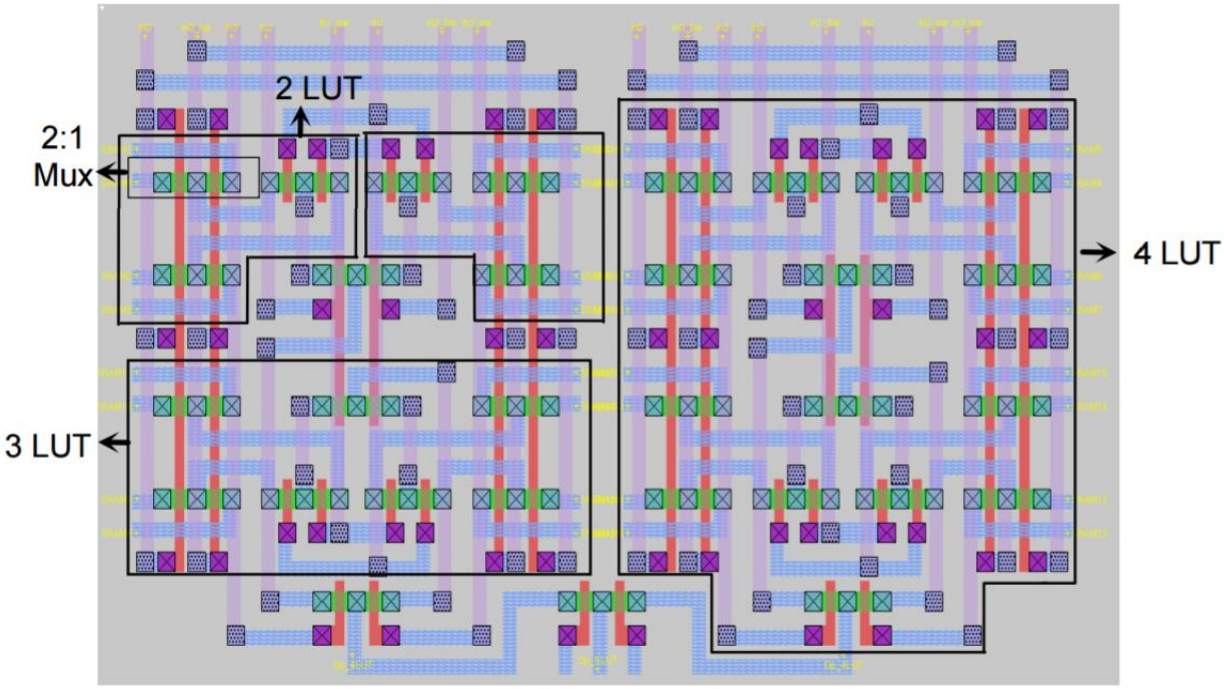


Figure 10: Repetitive use of 2-input LUT to create larger LUT circuits [25]

Chapter 3

Placement Exploration:

A 4:1 multiplexer consists of 6 transistors where the recursive y-topology of the circuit allows the transistors to be divided into group of twos. Within each group transistors can share the diffusion regions to minimize the area [4]. Hence, in this work, we consider the placement of three 2-transistor basic blocks in constructing the layout of 4:1 multiplexers.

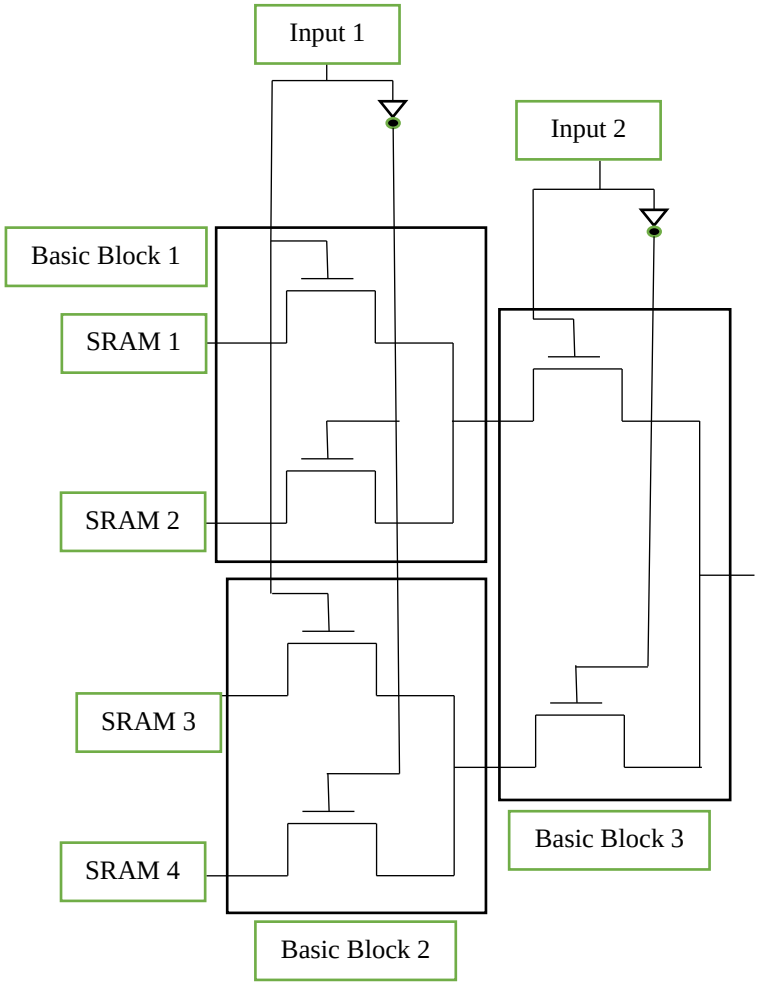


Figure 11: Schematic of a 2 input LUT Multiplexer

Note that, the same 2-transistor basic blocks are also used in chapter 5 to build higher order LUT multiplexers based on the minimum area 4:1 multiplexer layout identified in this work. In this chapter of the thesis, we first define the stick diagram symbol for a basic block from the schematic and compares it to the actual layout of the block. Later on, we exhaustively searches through all possible layout placement options of these basic blocks within a 4:1 multiplexer and eliminate all the redundant placements.

3.1 Conversion from schematic to stick diagram:

Stick diagram is a conventional way of presenting a digital circuit in simpler manner. Before the cell can be constructed from a transistor schematic, it is necessary to develop a strategy for the cell's basic layout. In other words, stick diagrams provide a rough floor plan of any circuit before implementing the layout of that specific circuit. From the previous section, we know that a 4:1 multiplexer is consisted of 3 basic blocks. Hence, it is very important to create the stick diagram of a basic block since we will be using them repeatedly in our discussion. Using the stick diagram instead of the schematic level diagram will simplify the process and make it easier for the readers to understand the layout procedure.

Figure 12 presents a schematic level diagram as well as the stick diagram of a basic block. Different colored crosses ‘x’ represents different nodes from the schematic. As seen in the diagram the red cross represents the left diffusion region of the top transistor. Similarly, the blue cross in the stick diagram refers to the right diffusion region of the bottom transistor. Since the transistors are connected with each other through their right diffusion regions, these regions are shorted into the same node. This node is marked by the green cross in the stick diagram.

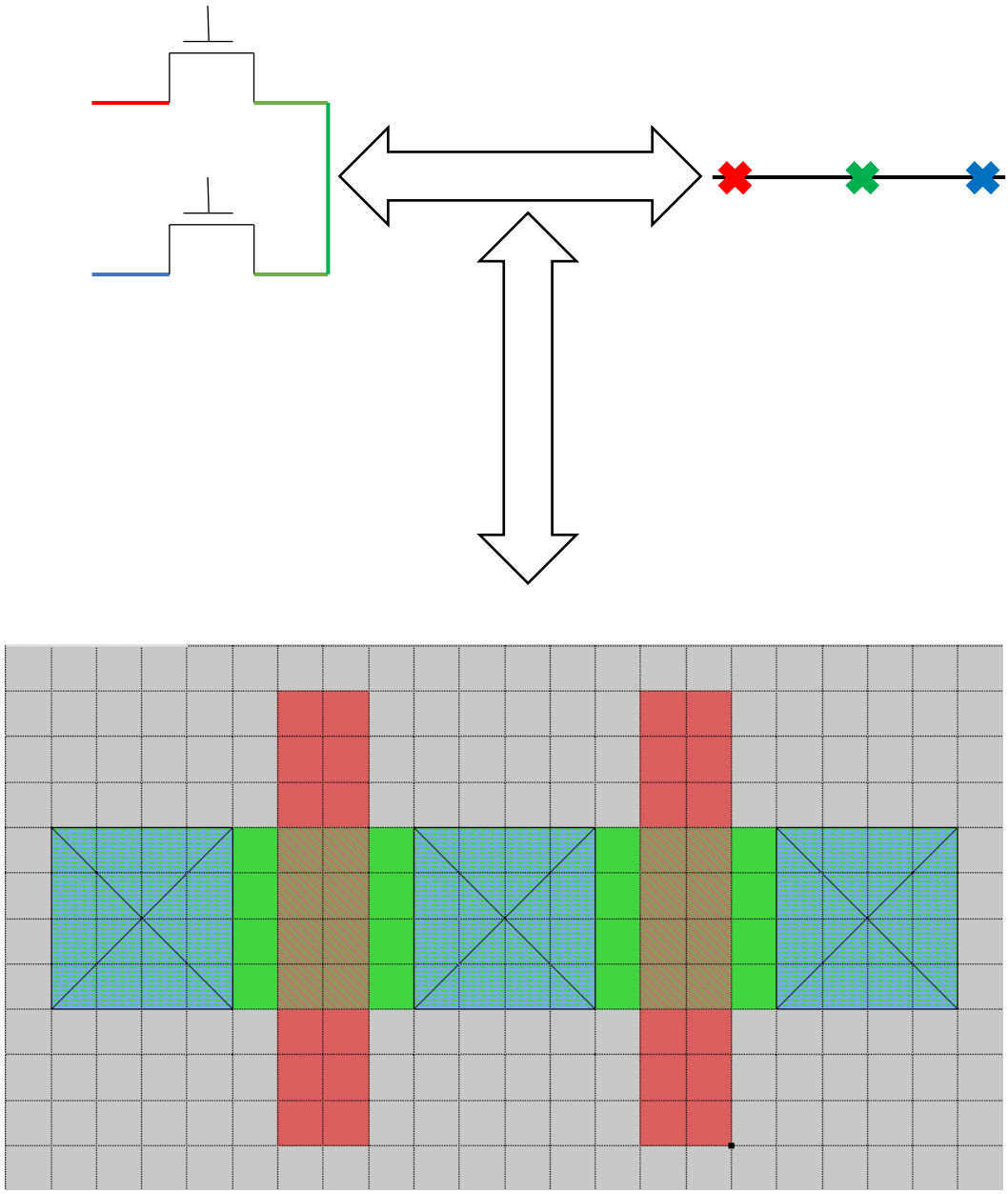


Figure 12: Layout, Schematic and Stick Diagram of a Basic Block

3.2 Placement of the basic blocks:

Two variables play the most important role in achieving the smallest layout area. They are: Placement and routing.

This section analyzes all the placement possibilities to layout a 4:1 multiplexer for a 2-input LUT. The routing issues and possibilities are then analyzed in the next chapter. Finally, the combination of the best outcome of these chapters will give the best possible 2-input LUT layout in terms of area. Then, the result will be compared with the previous work in chapter 5.

Figure 10 shows that, 3 basic blocks are required to create a 4:1 multiplexer. Since the number of basic blocks required is very low, the idea is to do an exhaustive search of all possible placements of these three basic components and Figure out the most suitable one for routing. According to the design rule for SCMOS [23], poly-silicon cannot be both vertical and horizontal with in the same layout. Therefore, we cannot explore or consider the placements where any of the basic block stick diagram is perpendicular to the other ones. Hence, all the three-basic block stick diagram must be either vertical or horizontal to be considered as a valid placement option. Consequently, three fundamental placements are available for the 4:1 multiplexer where all the other placement options are either not legal according to the Design Rule Check (DRC) rules or they are just rotations or mirrored images of one of the three fundamental ones. The Figures below present all the fundamental placements with their respective layout and stick diagrams. However, these Figures don’t show the routing which will be done in the following chapters.

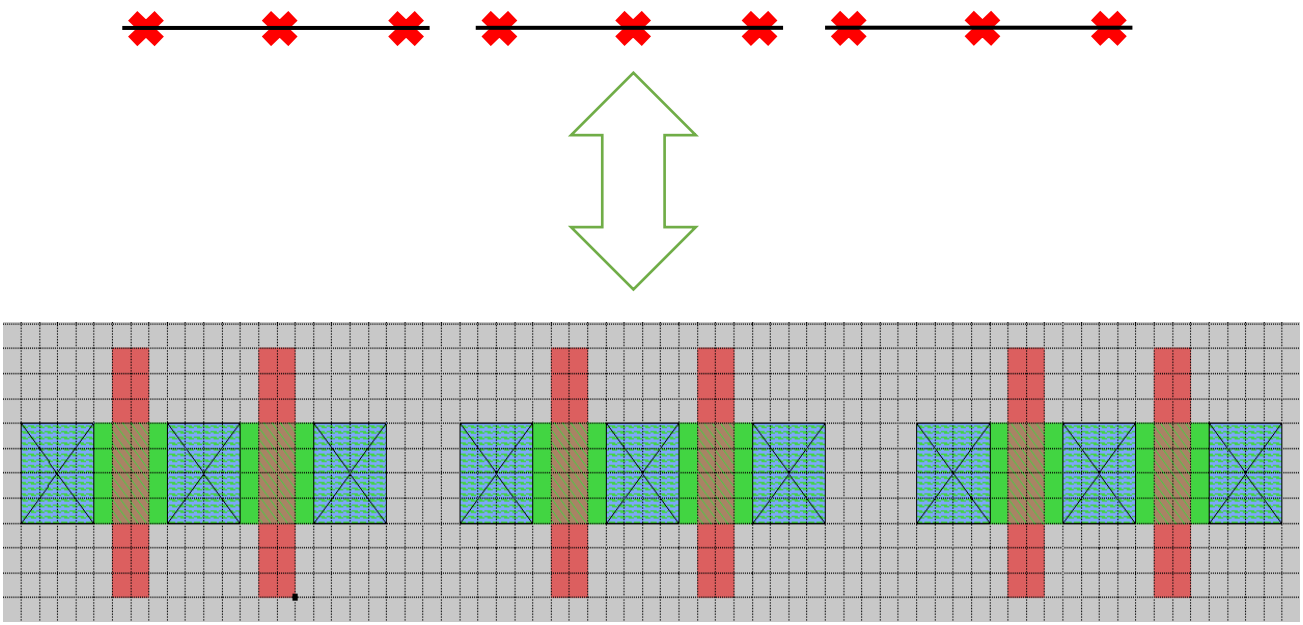


Figure 13: Valid placement 1

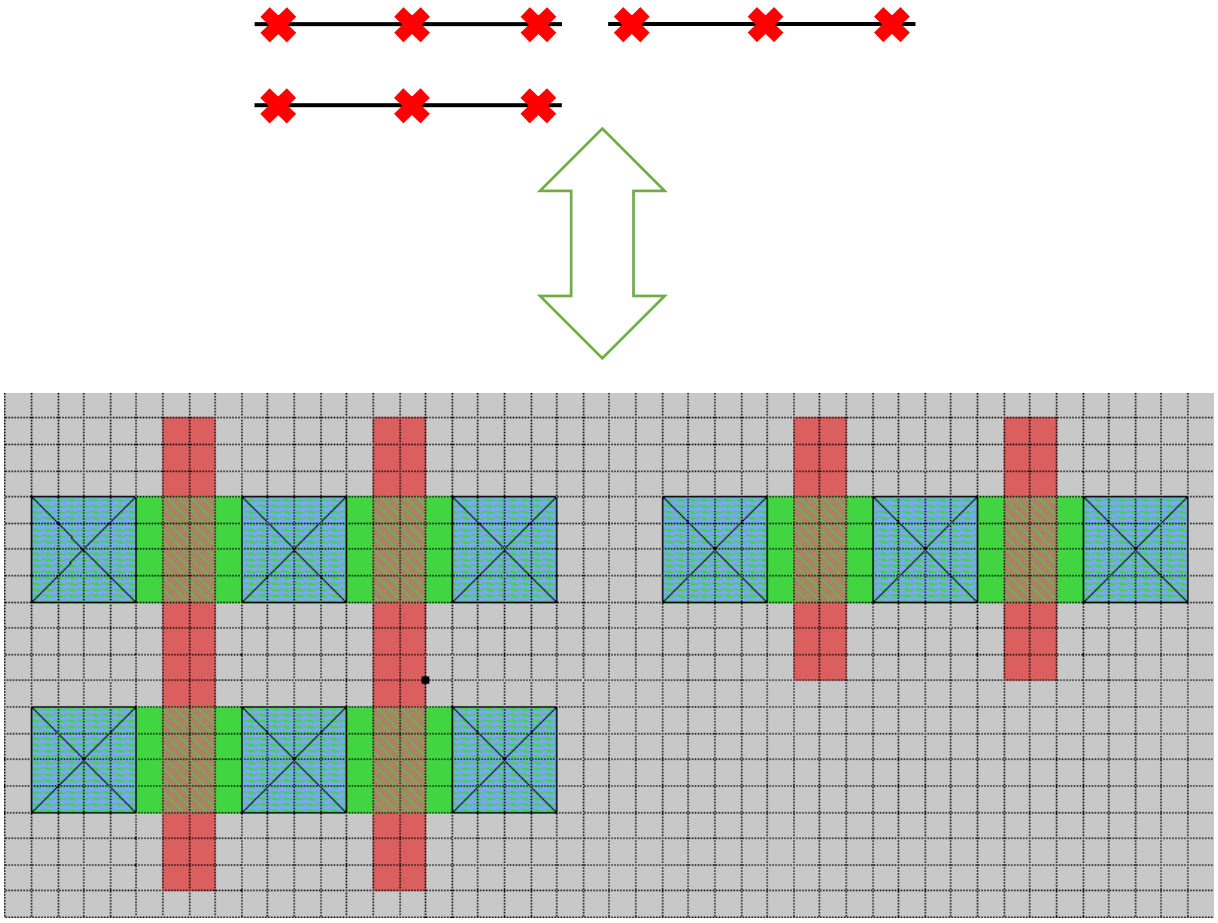


Figure 14: Valid placement 2

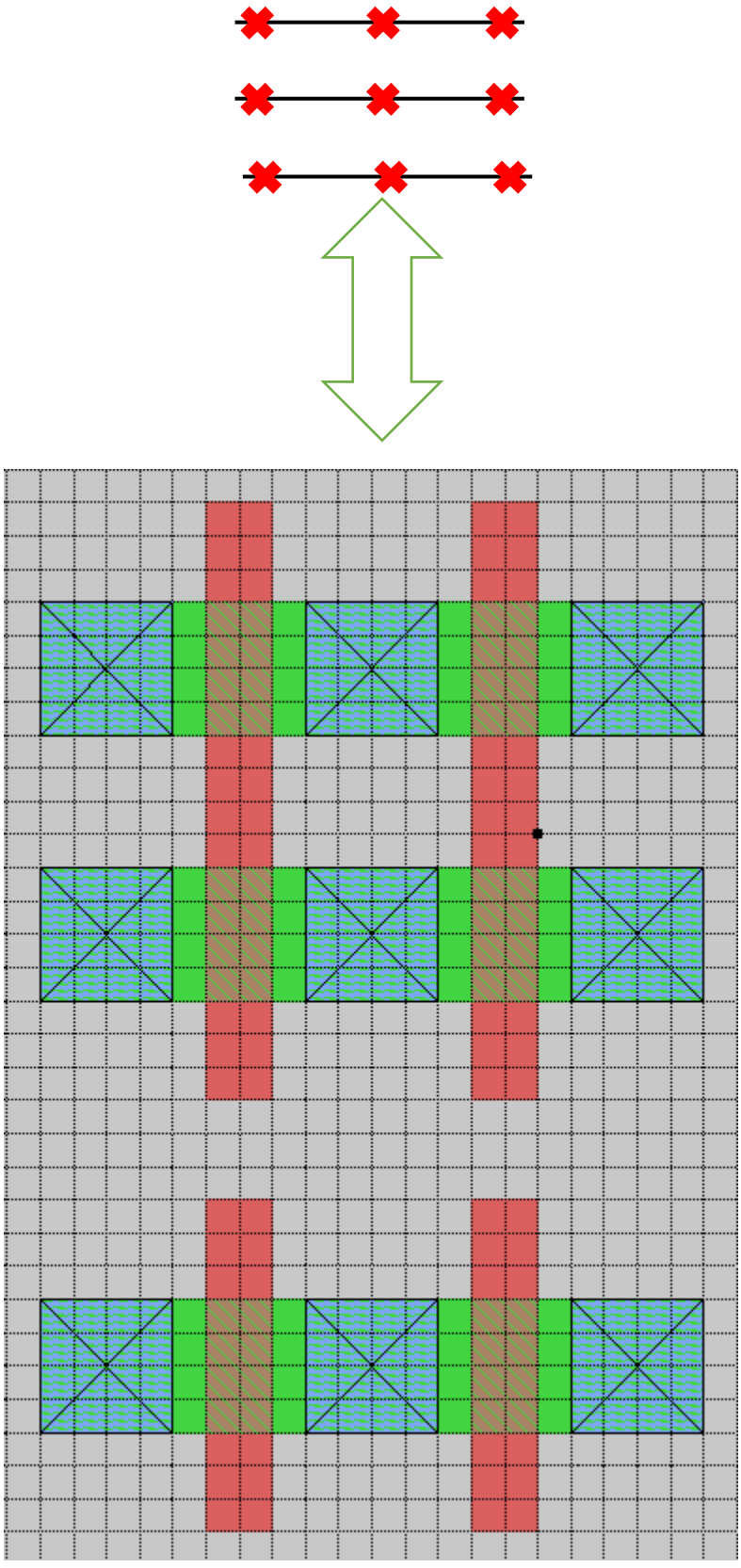


Figure 15: Valid placement 3

At this point of the investigation, it is important to discuss how the area of the 2 LUT multiplexer will be calculated from the stick diagram. It is also worth mentioning that the layout area will be calculated based on λ, where λ refers to half of the channel length of a transistor. The area calculation will also take into consideration the minimum spacing required between metal contacts and metal layers according to CMOS DRC rules, note that, the stick diagram presented in Figure 12 has three crosses which refers to three different terminals or connection points of the transistors. Since, connections are established using metal layers in silicon level designs, each connection point contains a metal contact to connect the metal wire with the desired diffusion region on the transistor. As a result, a basic block will have three metal contacts as shown in Figure 12. Based on the λ-based scalable DRC rules from [23], the area taken by each metal contact is 4 λ. The minimum spacing between two contacts has to be another 4 λ. Moreover, no other component can be placed with in the distance of 4λ from the contact at any direction.

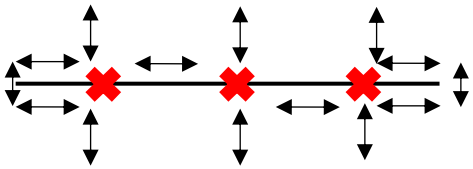


Figure 16: Spacing requirement for each stick diagram

As shown in Figure 16, where each double ended arrow segment represents 4λ of spacing, therefore, total area required to implement one basic block layout without any routing is 28λ x 12λ.

Chapter 4

Routing Algorithm:

This part of the report describes routing algorithm used in this work. The routing used in this work is a maze router designed based on the Lee algorithm [27] which is also known as the grid based Dijkstra's algorithm. The Lee algorithm is very efficient, arguably the best to solve any path finding problem with in a grid set up. Two salient features of this algorithm are, it will always find a path if one exists and the found path will always have the minimum possible cost [27]. The steps for Lee path connecting search algorithm is given below:

1. Place the cell(s) that represents the origin of a path in a priority list.
2. Find the lowest cost cell 'C' in the list. If it is a target cell, go to Step 12.
3. If 'C' was previously expanded, skip to Step 9.
4. Otherwise generate the first neighbor of 'C'.
5. If the neighbor was previously expanded into or it is an obstacle cell, skip to Step 7.
6. If the neighbor is the destination go to step 13.
7. Record the cost and predecessor direction of the neighbor and add it to the list.
8. If any more neighbors of 'C' exist, repeat from Step 5 for the next neighbor.
9. Record the occupancy status of 'C' in the cell matrix.
10. Delete 'C' from the priority list.
11. If any more cells exist in the priority list, repeat from Step 2.
12. No path exists. Stop.
13. Trace back along the path to its origin. Done. [28]

The remainder of this chapter describes how the standard Lee's algorithm is modified in this work to solve the routing problem for 4:1 multiplexers while creating routing solutions that obey the DRC rules.

4.1 Problem Specifications & Customizing Lee algorithm:

The first part of the specification is to figure out how many source and destination pairs are there to be routed. A 4:1 multiplexer has four SRAM inputs, two poly-silicon inputs and three basic building blocks constructed out of six transistors. Since the input signals are carried by poly-silicon, no metal routing or connection is required for the input signals. That leaves us with seven different source and destination pairs. Figure 17 shows each of the pairs in different colors to present the problem in a simple manner:

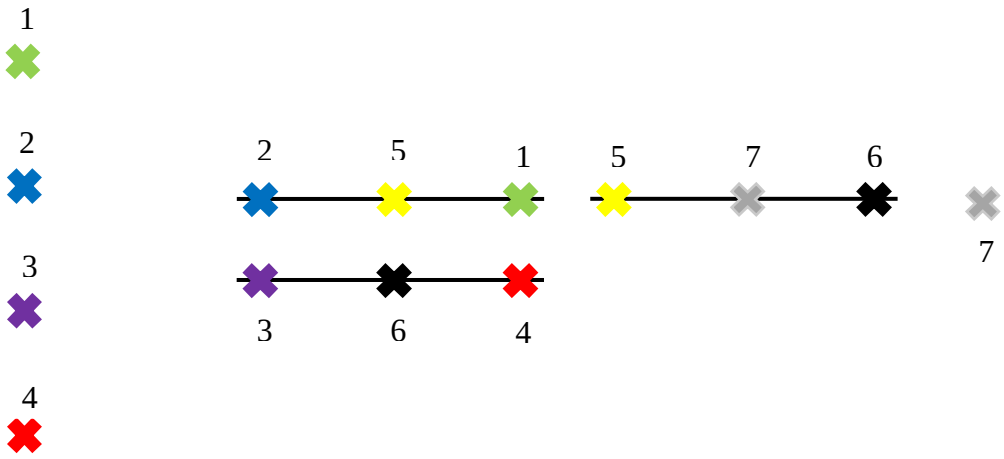


Figure 17: Stick Diagram of 2 LUT multiplexer showing individual pairs to be routed

The Lee algorithm produces the best result in a grid based environment. This fits well with industrial and academic DRCs which also can be approximated by a grid based system, since the

metal wires can only bend 90º at a time in this layouts [26]. Therefore, we set up a grid based platform for our layout process. In particular, we expand the 2-D lee algorithm to a 3-D process to accommodate multiple metal layers.

Each node in our 3-D grid system is given a unique identification number based on their x, y and z positions in the grid. Then each of the crosses from

7														
6		✖												
5														
4		✖		✖		✖		✖		✖		✖		✖
3														
2		✖		✖		✖		✖				✖		
1														
0		✖												
y/x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**

Figure 18: Mapping the Source-Destination pairs in the grid

the stick diagram from Figure 14 was placed in individual grid nodes where the z direction indicates the metal layer that a signal is routed on. Here metal layer one is assigned z position of 0 and higher metal layers are assigned consequently higher z positions. In particular, Figure 18 shows the mapping of the stick diagram from Figure 14 in our 3-D grid system where all the nodes shown in the stick diagram are mapped onto metal layer 1; and the detailed grid assignment is shown in table 1. Note that each grid in our 3-D grid system has an X × Y dimension of 4λ × 4λ. This is based on the fact that each contact in a metal to metal or metal to diffusion via under the

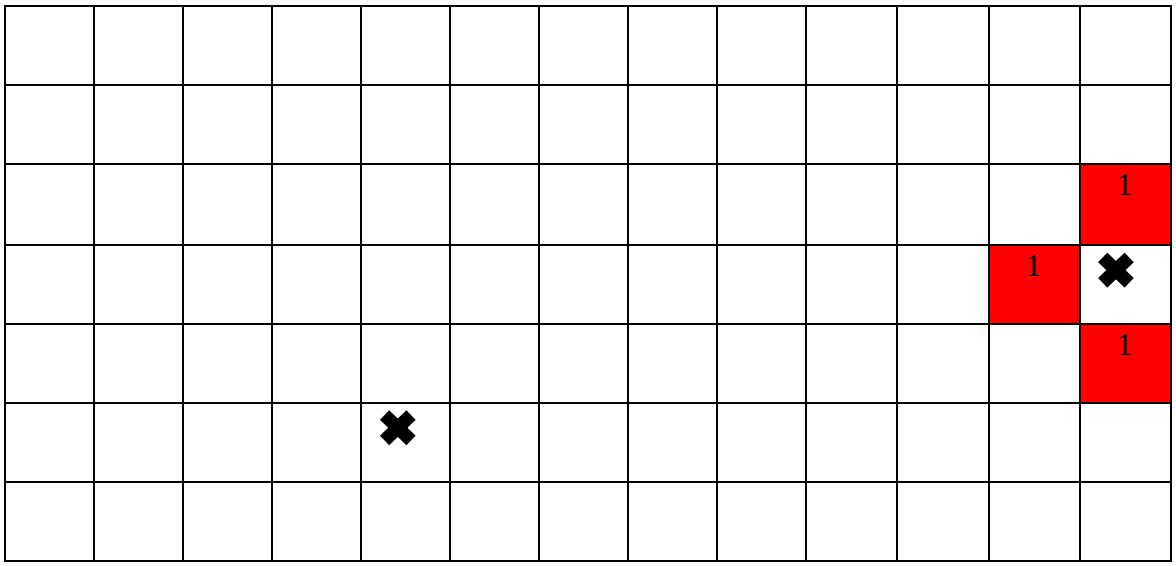
scalable DRC [23] has a minimum size requirement of 4λ × 4λ each and metal wires also have a minimum width requirement of 4λ.

Pair Name	Color of the Pair	Identity of the node	X Position	Y Position	Z Position
I1		Source	1	6	0
		Destination	7	4	0
I2		Source	1	4	0
		Destination	3	4	0
I3		Source	1	2	0
		Destination	3	2	0
I4		Source	1	0	0
		Destination	7	2	0
S5		Source	5	4	0
		Destination	9	4	0
S6		Source	5	2	0
		Destination	13	2	0
O7		Source	11	4	0
		Destination	11	2	0

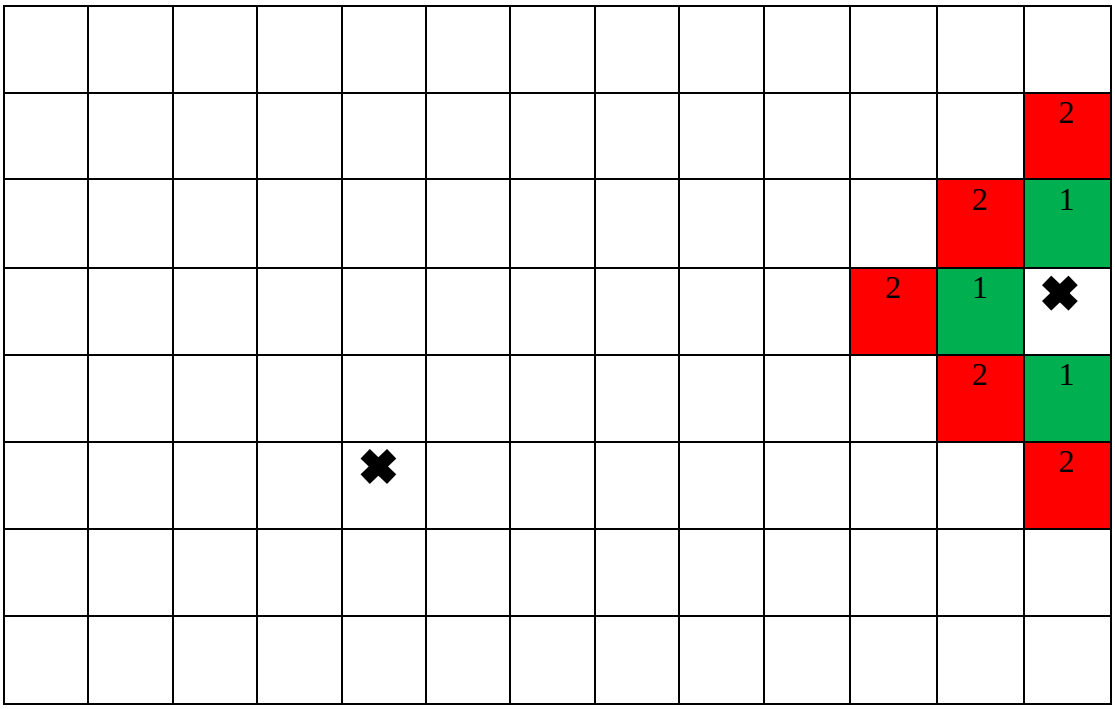
Table 1: Position of each contact of the stick diagram in the grid

In the remainder of this section, we will use the placement shown in Figure 14 to illustrate the inner workings of our 3-D Lee's algorithm implementation. Our algorithm routes one source and destination pair at a time. Once the first pair is routed, the same process will be applied to rest of

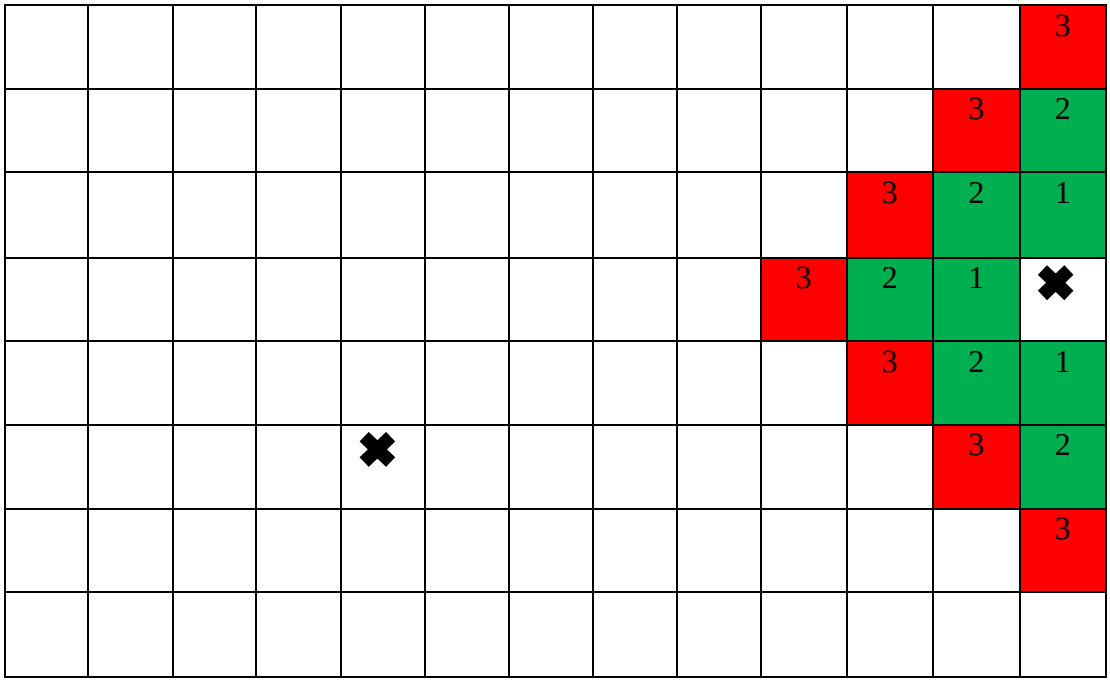
the pairs to establish the shortest possible connection between them. For example routing signal S6 from its source to destination consists of the following steps.. First of all, a field called value is declared for each and every node in the grid. Then source and destination node is recognized and the value of these nodes are set to '0'. All the other nodes of the grid are set to '-1'. The source then expands to it immediate neighbors, and update the node values to "previous node value +1". Note that this process does not allow diagonal expansion. Then the expanded nodes expand to their immediate neighbors and update the node values accordingly. This process continues until the one of the expanded node is the destination or there are no more nodes to be expanded in the grid as shown by Figure 19(a) to Figure 19(j)



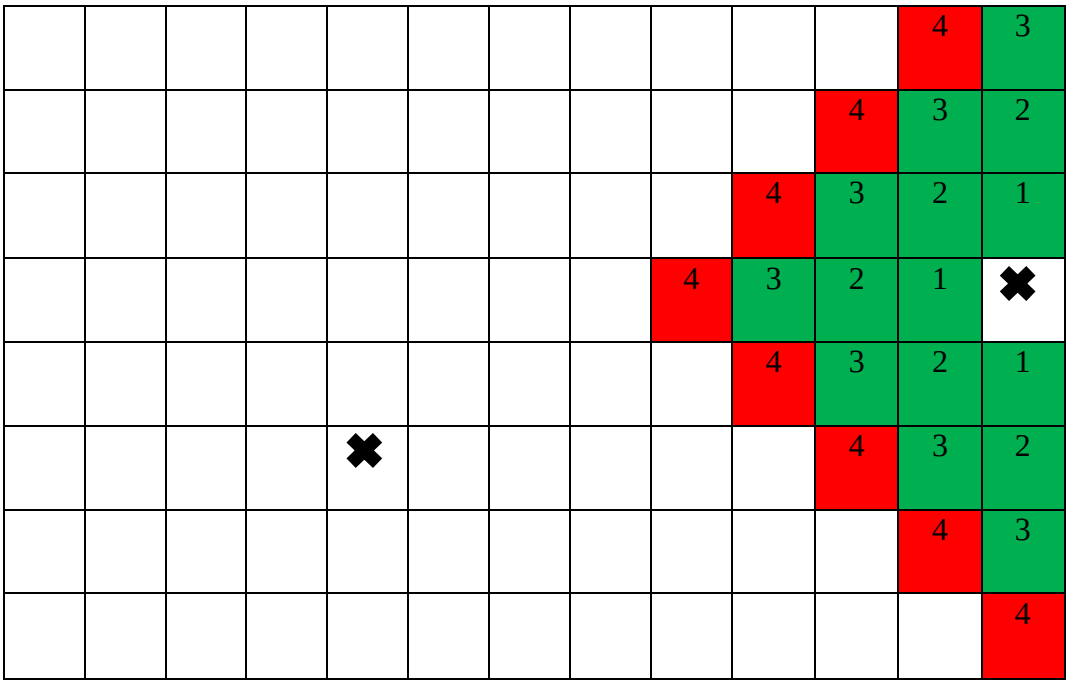
(a)



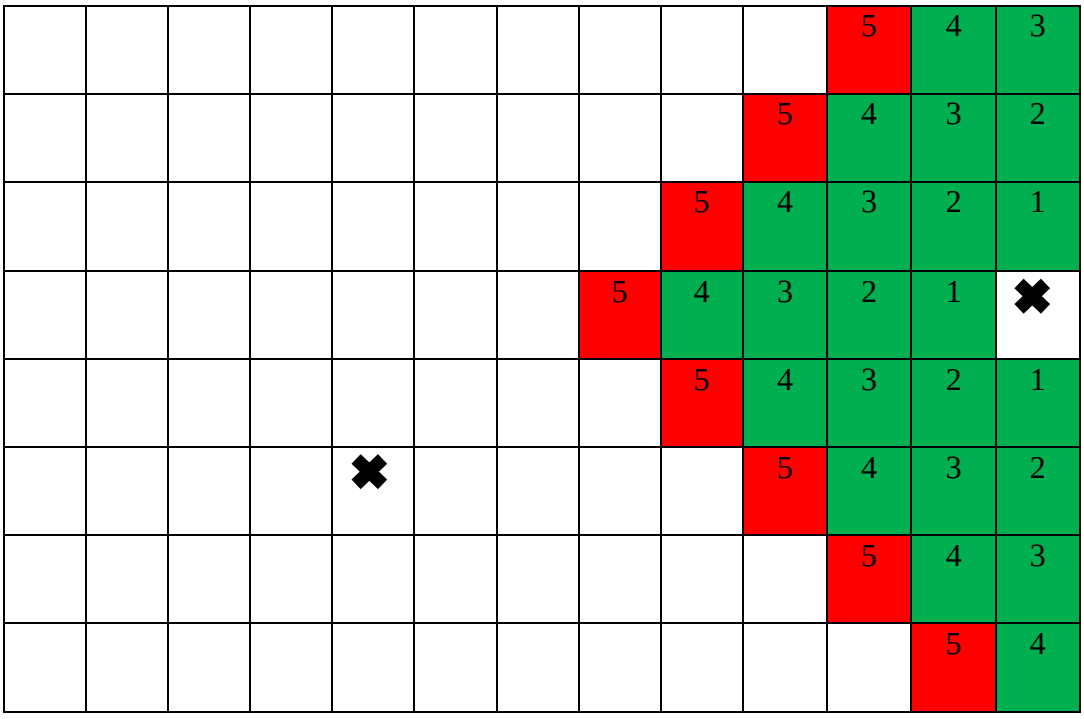
(b)



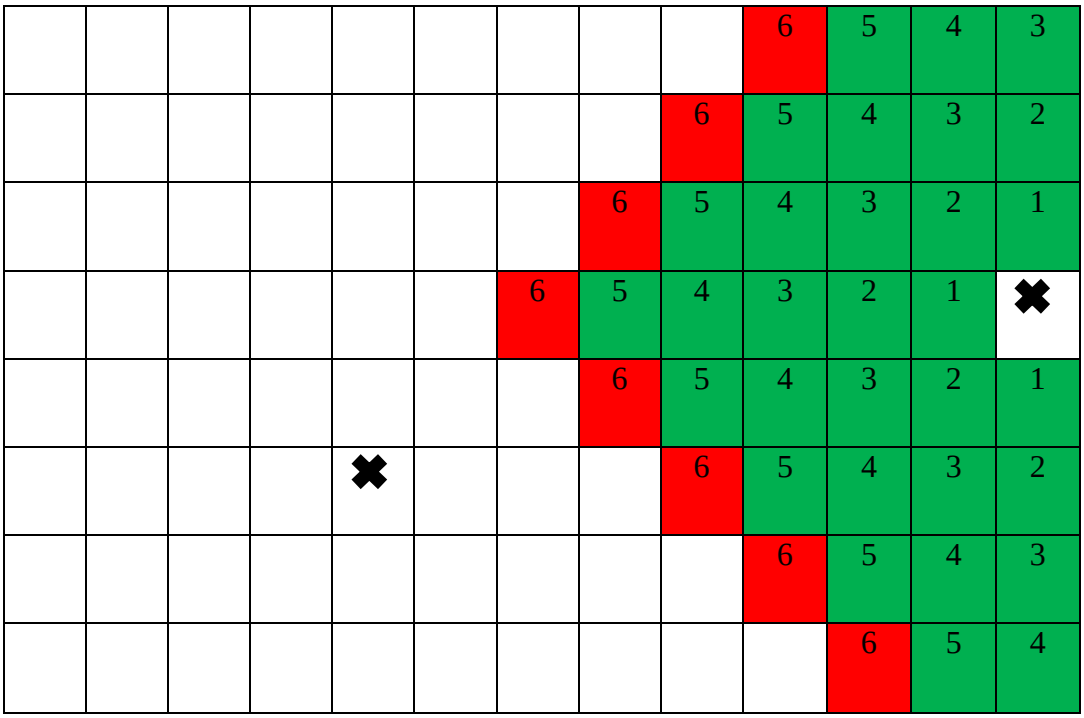
(c)


4 3
4 3 2
4 3 2 1
4 3 2 1
4 3 2 1
4 3 2
4 3
4

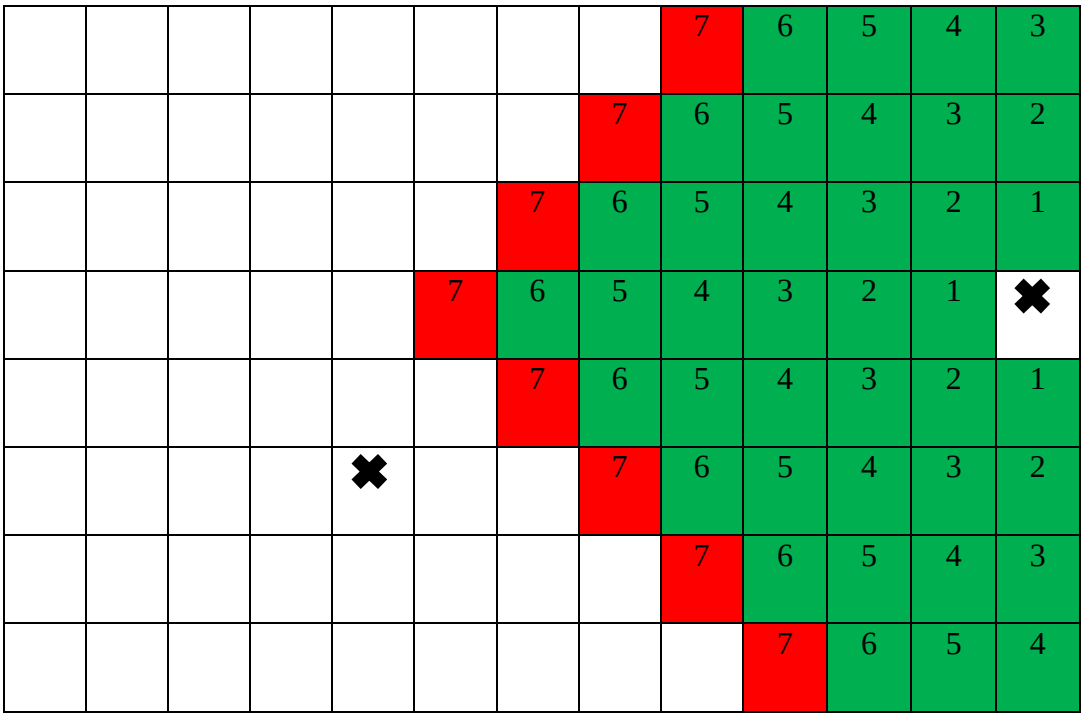
(d)


5 4 3
5 4 3 2
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2
5 4 3
5 4

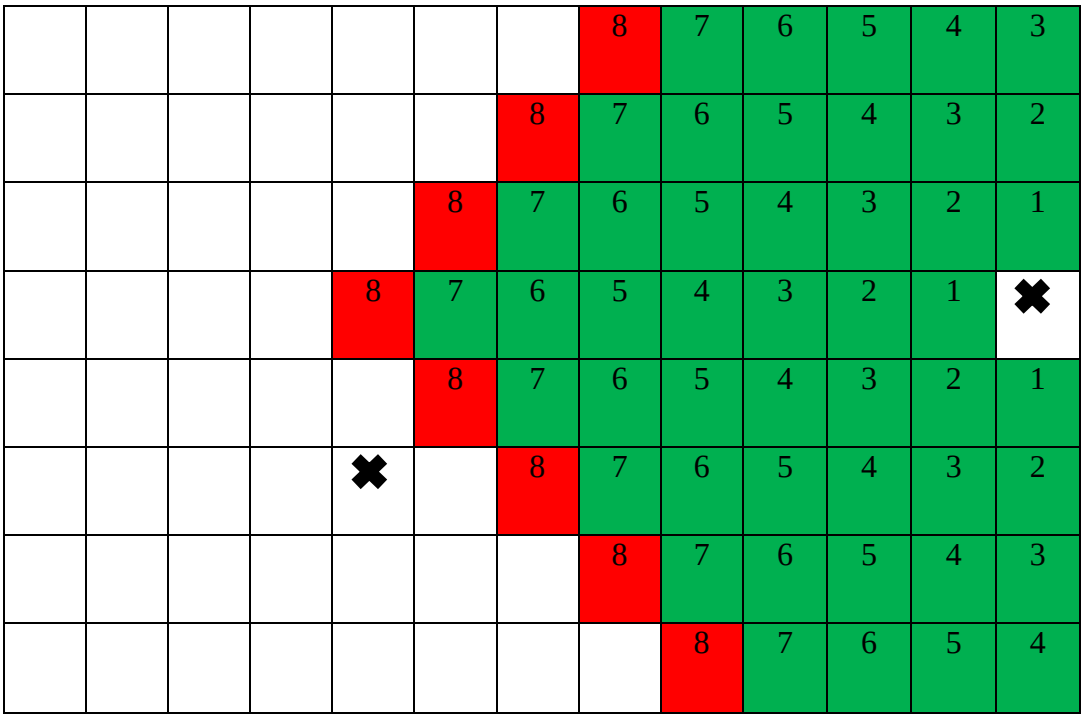
(e)


6 5 4 3
6 5 4 3 2
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2 1
6 5 4 3 2
6 5 4 3
6 5 4

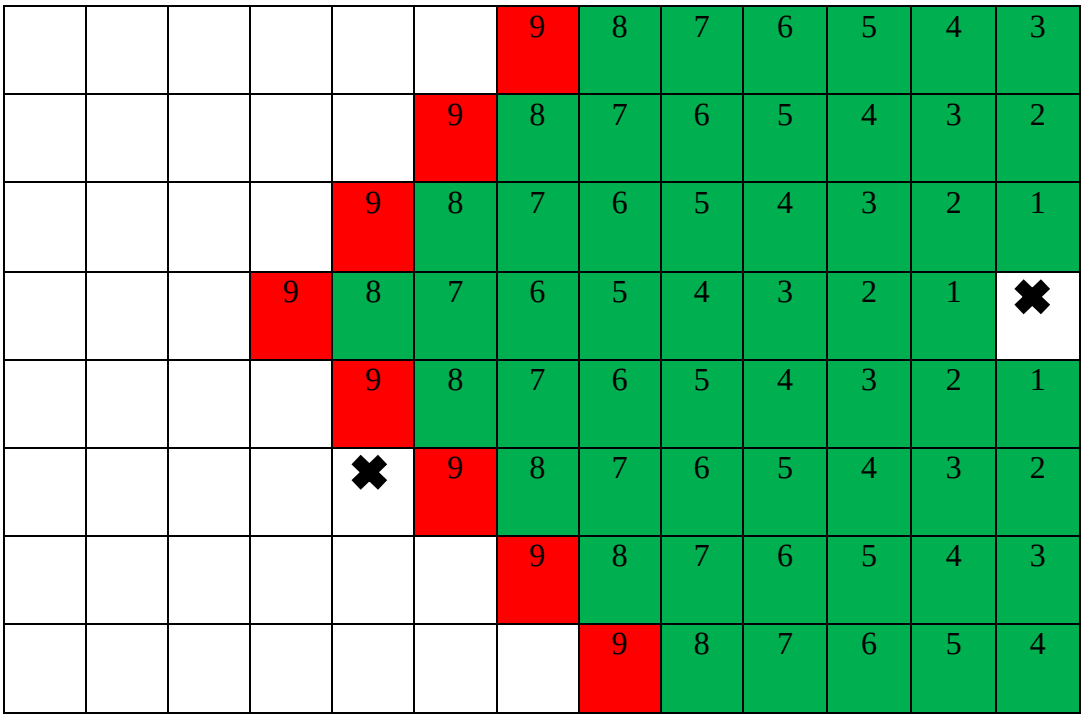
(f)


7 6 5 4 3
7 6 5 4 3 2
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2 1
7 6 5 4 3 2
7 6 5 4 3
7 6 5 4

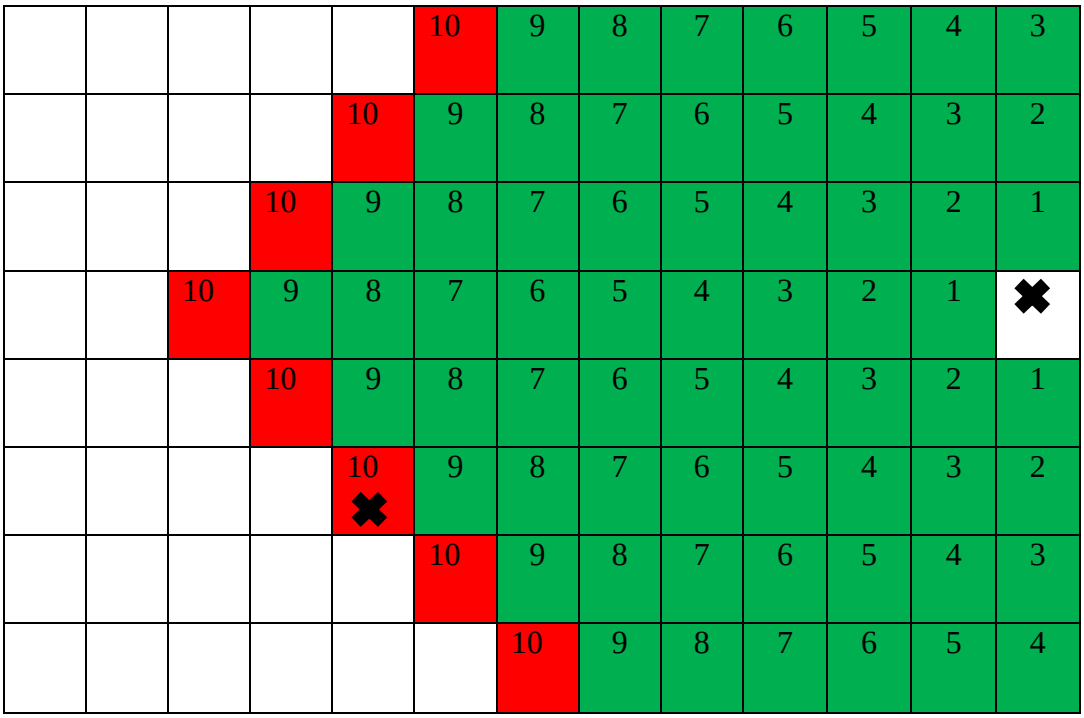
(g)


8 7 6 5 4 3
8 7 6 5 4 3 2
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2
8 7 6 5 4 3
8 7 6 5 4

(h)


9 8 7 6 5 4 3
9 8 7 6 5 4 3 2
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2
9 8 7 6 5 4 3
9 8 7 6 5 4

(i)



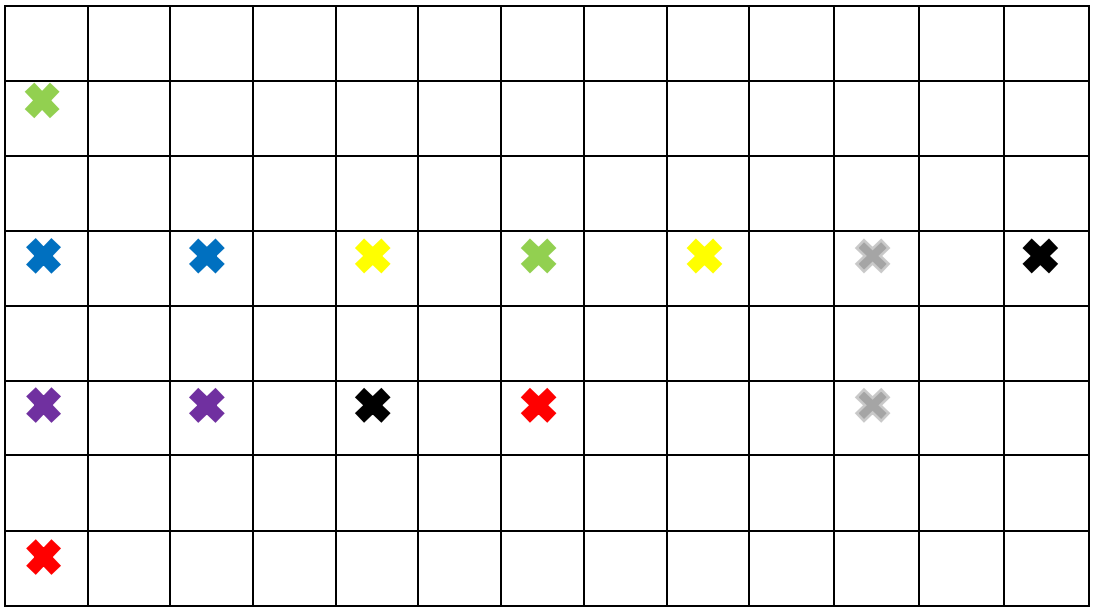
(j)

Figure 19: Complete expansion process for S6

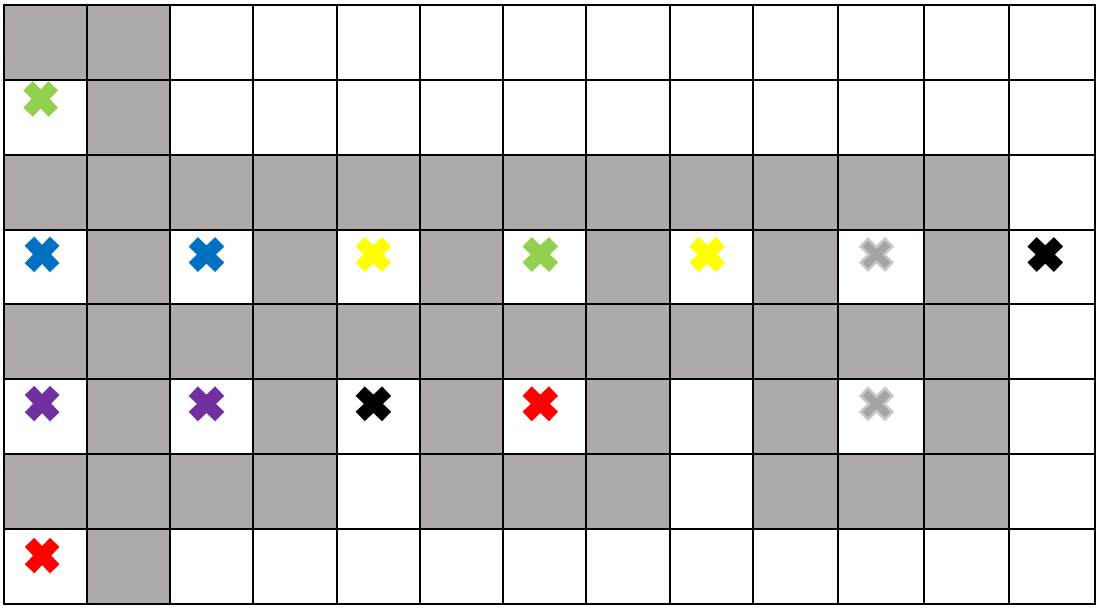
We terminate the expansion as soon as the destination is reached. In the Figure, the nodes that are already expanded to are marked by the color green. Nodes that are being expanded at current wave front are marked with red. Nodes that are never been expanded to are marked with white. In this case, it took ten iterations for the source to reach the destination. Note that, even though, the Figures portraits a two-dimensional (x and y) expansion, the actual program does it in all three (x, y and z) direction. However, the expansion gets a little more complicated when there are multiple sources and destinations.

Source and destination pairs are routed one at a time, therefore while routing one pair it had to be make sure that the expansion does not go through the rest of the sources or destinations. A concept we call padding was introduced to get rid of this problem. According to this concept, whenever a source is expanding toward its destination, all the other sources and destinations mark all their surrounding nodes as occupied by setting the flag ‘po’ from 0 to 1. ‘po’ stands for pre-occupancy

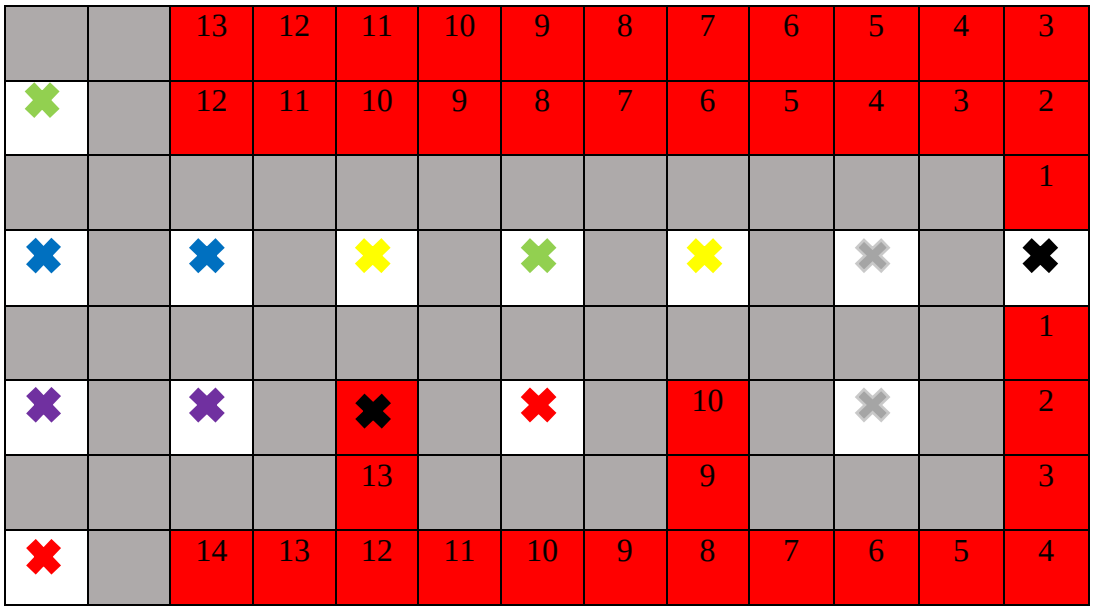
and sends the main function the occupancy status of the requested node. The pre-occupied nodes do not allow other nodes to expand through them, hence keeps the respective source and destination value as ‘0’. This procedure is known and handled by a function called pre-padding, within the program. There is another type of padding called post-padding, which will be discussed later in this section.



(a)



(b)

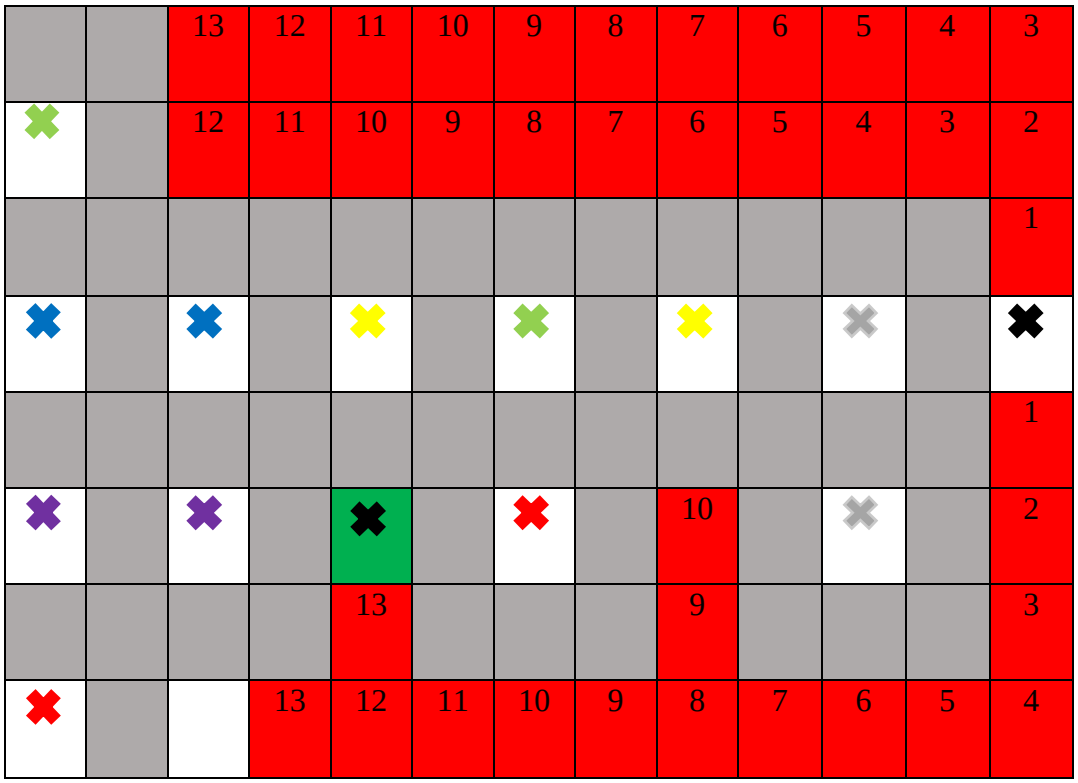


(c)

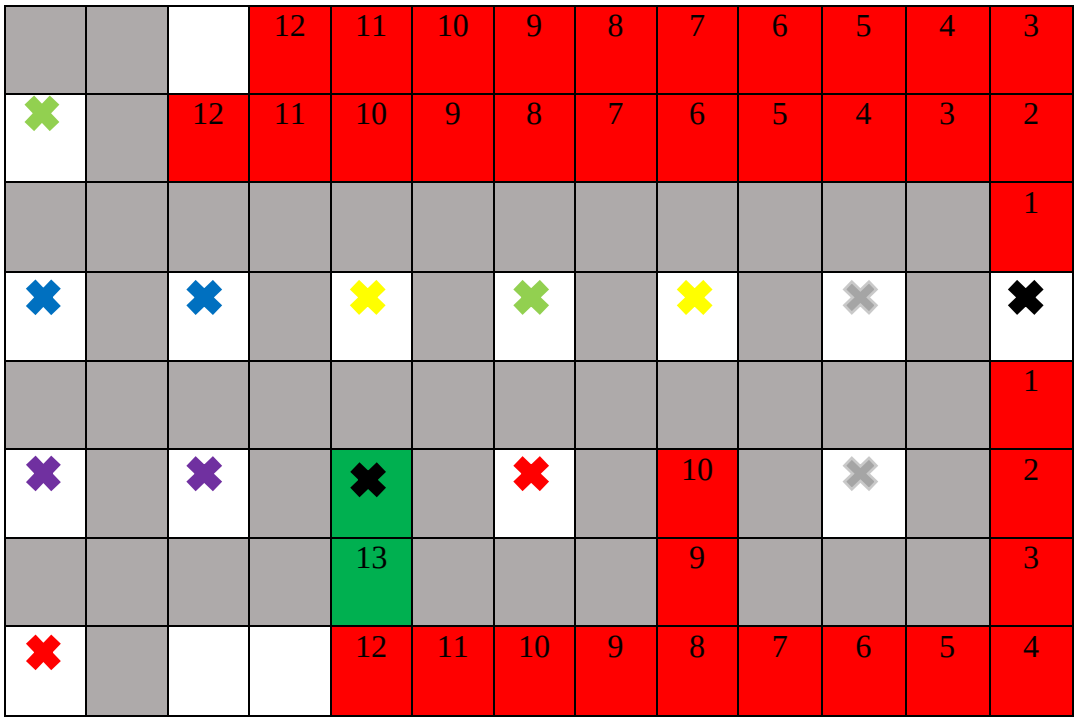
Figure 20: Pre-padding of the source and destinations

Figure 20(a) shows all the source and destination pairs without any pre-padding. However, when we decide to expand from source 6 to destination 6 all the other sources and destination is pre-padded as presented in Figure 20 (b). Figure 20(c) shows how the pre-padding affects the expansion process. Previously without any pre-padding it took 10 iterations for the source to reach the destination (Figure 19(j)), which has changed to 14 iteration with a complete different expansion direction with the pre-padding (Figure 20(c)).

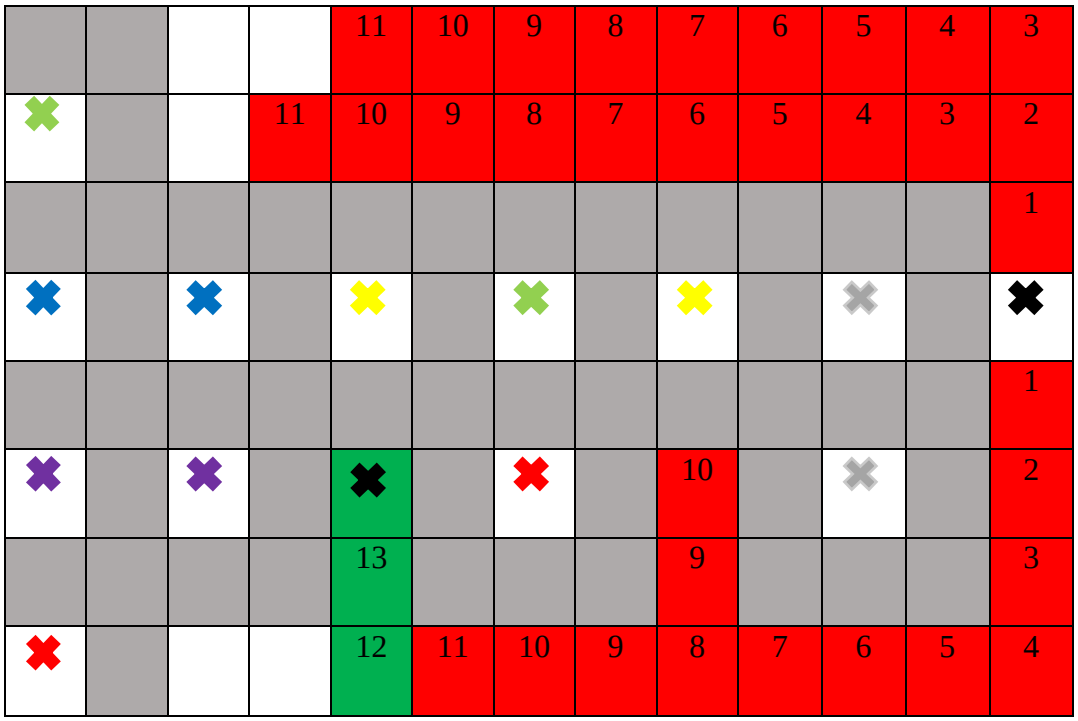
The next step is to specify the exact nodes that will be used to establish the routing connection and set the value for rest of the expanded nodes to '0', to make them available for other pairs to rout through. A function called "Trackback" was implemented to accomplish this task. This function initiates from the destination node and looks for the node it was expanded from. After finding the node, it was expanded from, it reapplies the concept to that node and keep repeating it until it reaches the source. This process gives the shortest path from destination to source. Again, a single source and destination will be used to explain the concept visually in Figure 21.


13 12 11 10 9 8 7 6 5 4 3
12 11 10 9 8 7 6 5 4 3 2
1
1
10 2
13 9 3
13 12 11 10 9 8 7 6 5 4

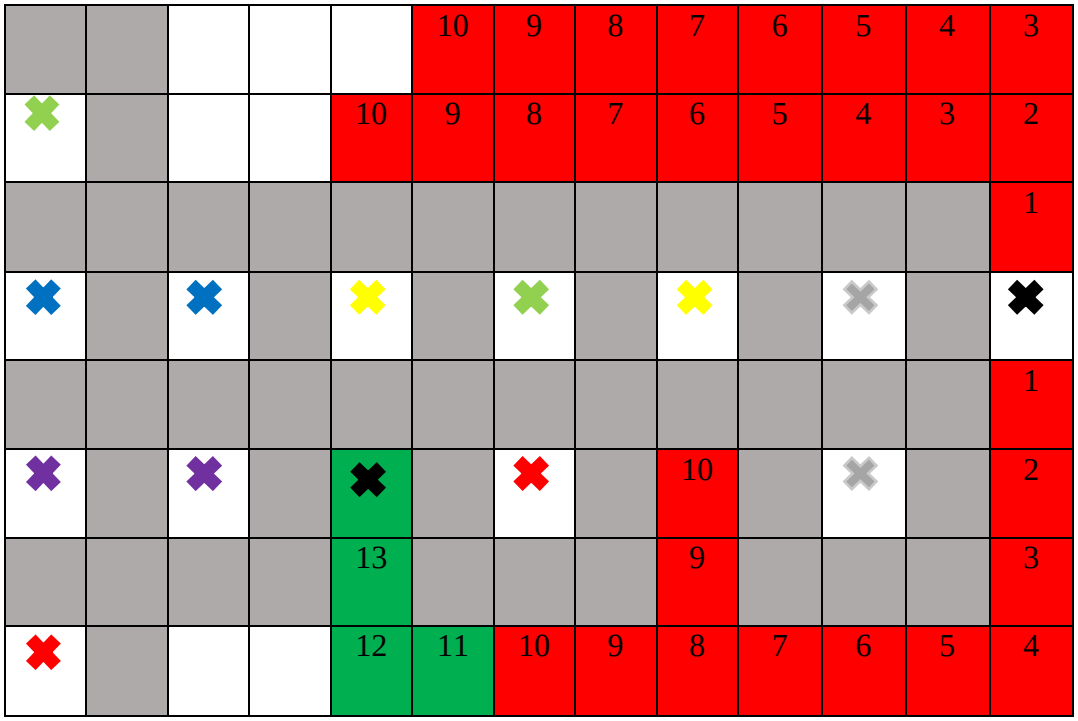
(a)


12 11 10 9 8 7 6 5 4 3
12 11 10 9 8 7 6 5 4 3 2
1
1
10 2
13 9 3
12 11 10 9 8 7 6 5 4

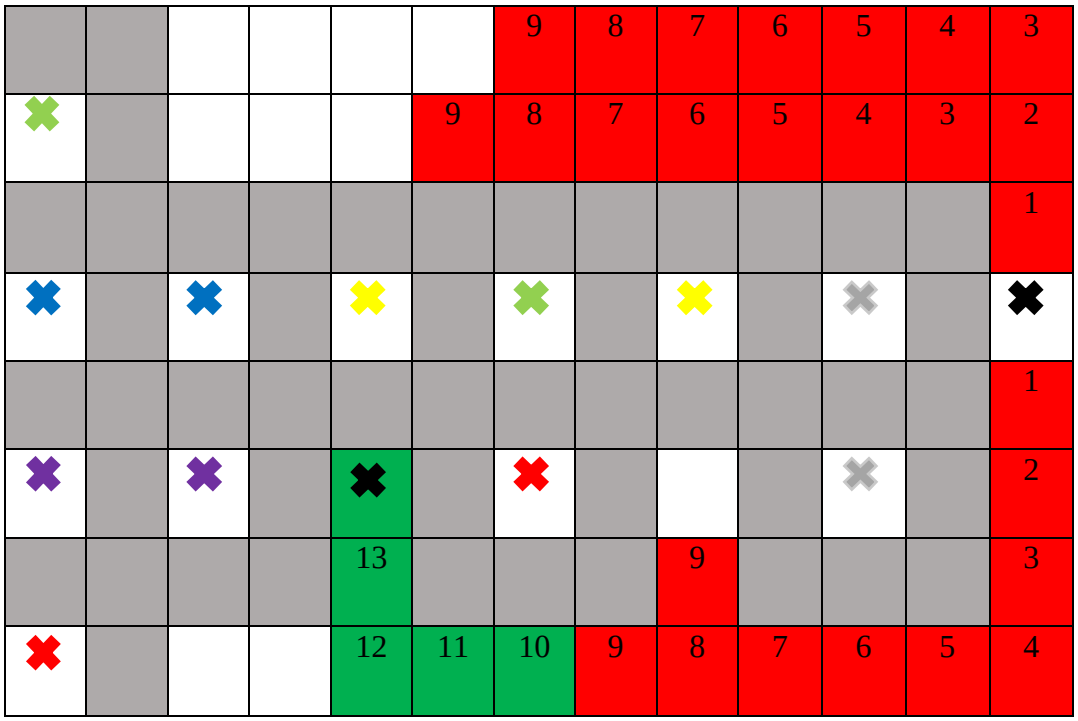
(b)


11 10 9 8 7 6 5 4 3
11 10 9 8 7 6 5 4 3 2
1
1
10 2
13 9 3
12 11 10 9 8 7 6 5 4

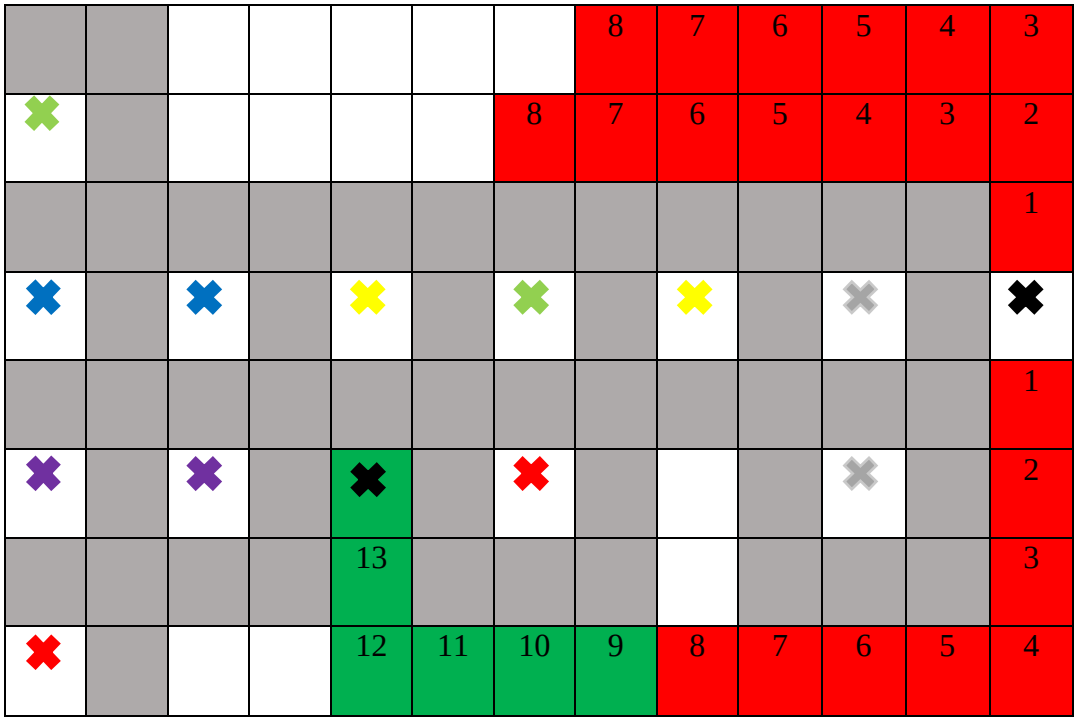
(c)


10 9 8 7 6 5 4 3
10 9 8 7 6 5 4 3 2
1
1
10 2
13 9 3
12 11 10 9 8 7 6 5 4

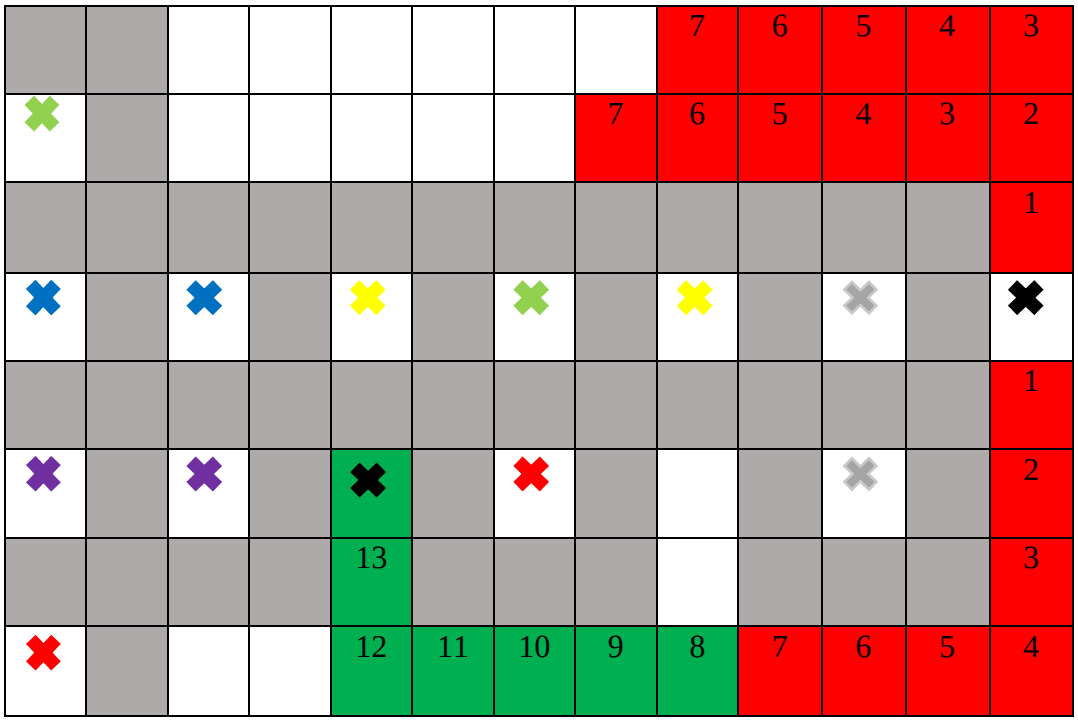
(d)


9 8 7 6 5 4 3
9 8 7 6 5 4 3 2
1
1
2
13 9 3
12 11 10 9 8 7 6 5 4

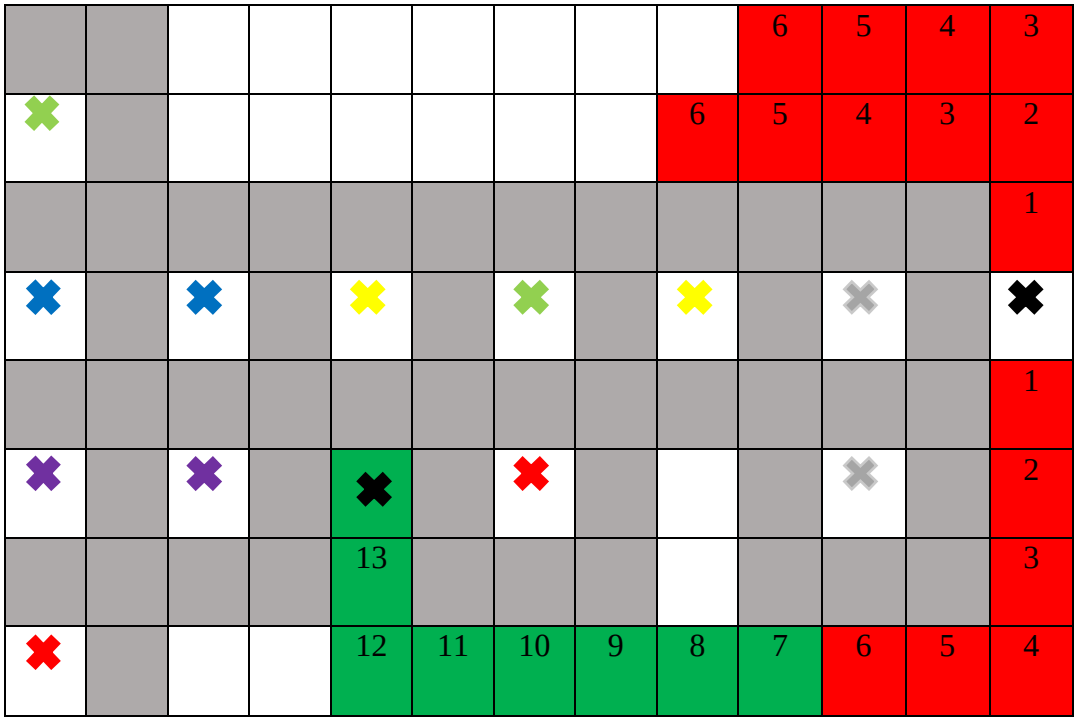
(e)


8 7 6 5 4 3
8 7 6 5 4 3 2
1
1
2
13 3
12 11 10 9 8 7 6 5 4

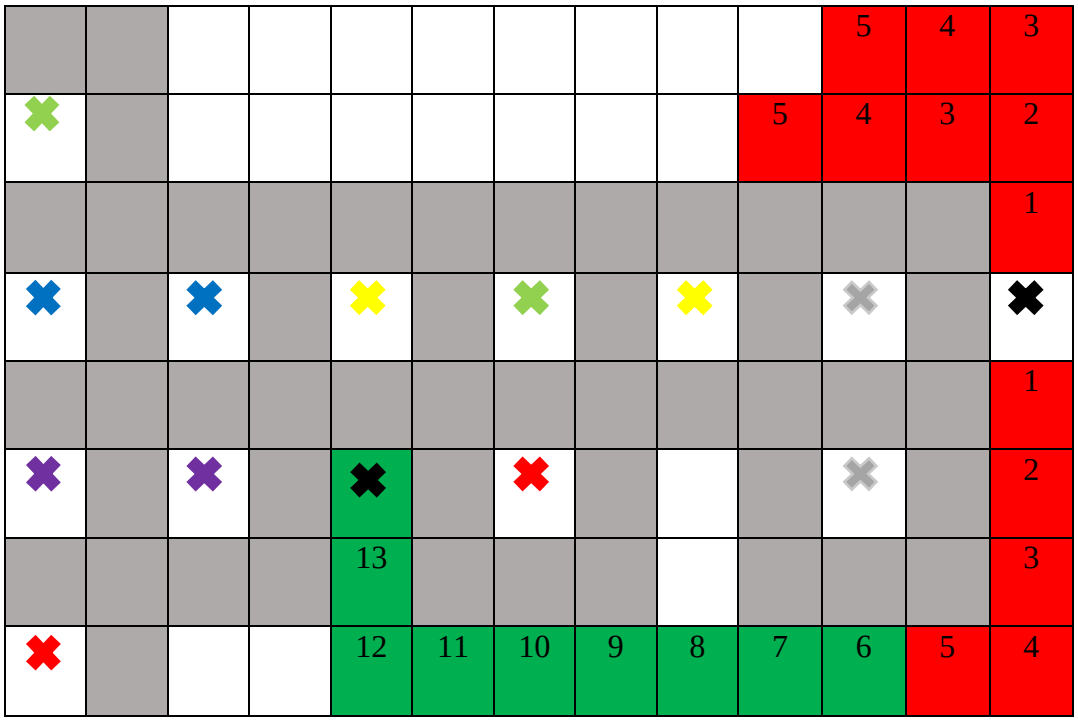
(f)


7 6 5 4 3
7 6 5 4 3 2
1
1
2
3
13
12 11 10 9 8 7 6 5 4

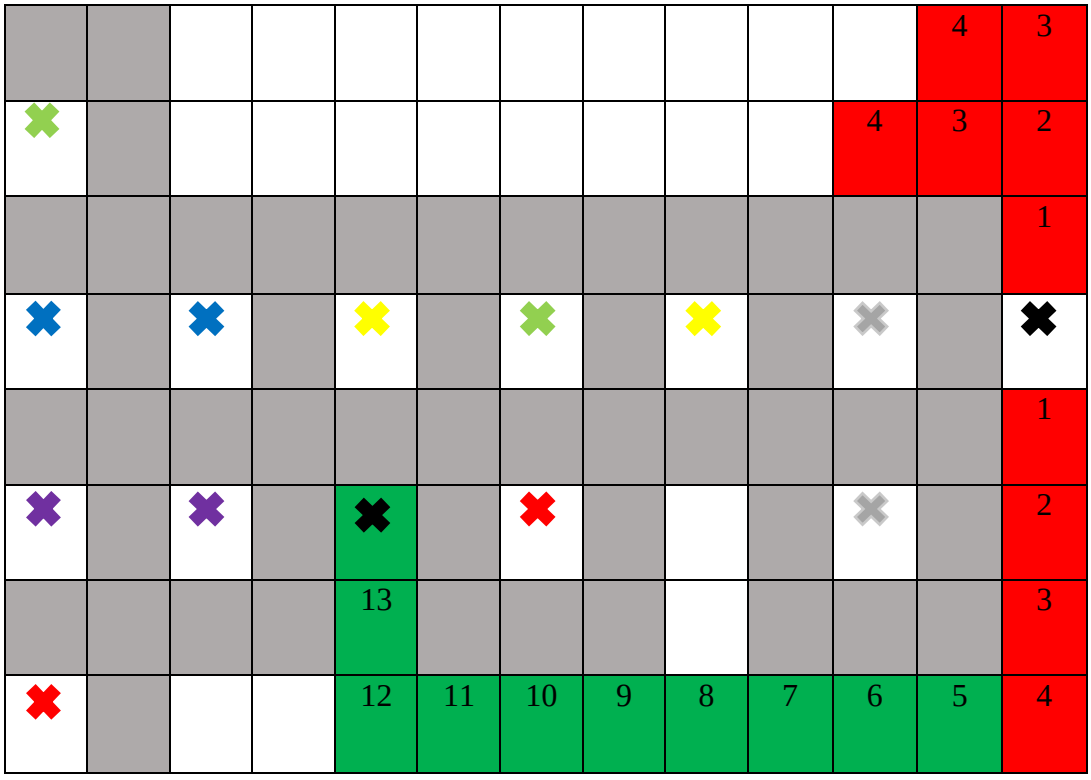
(g)


6 5 4 3
6 5 4 3 2
1
1
2
3
13
12 11 10 9 8 7 6 5 4

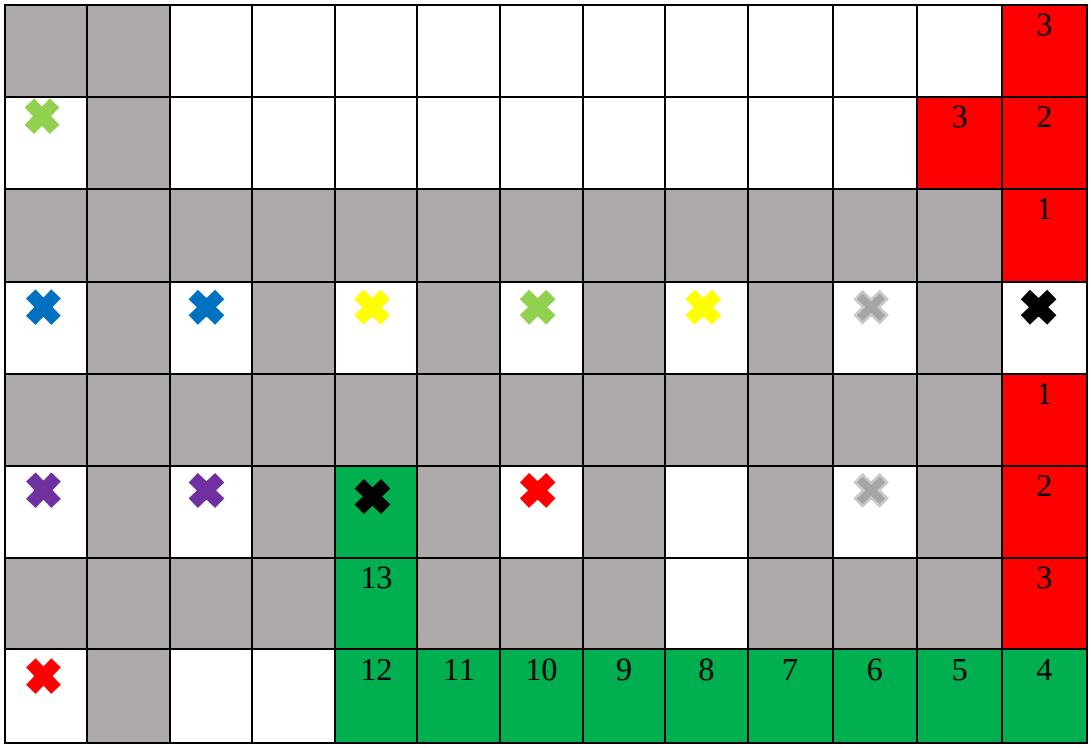
(h)



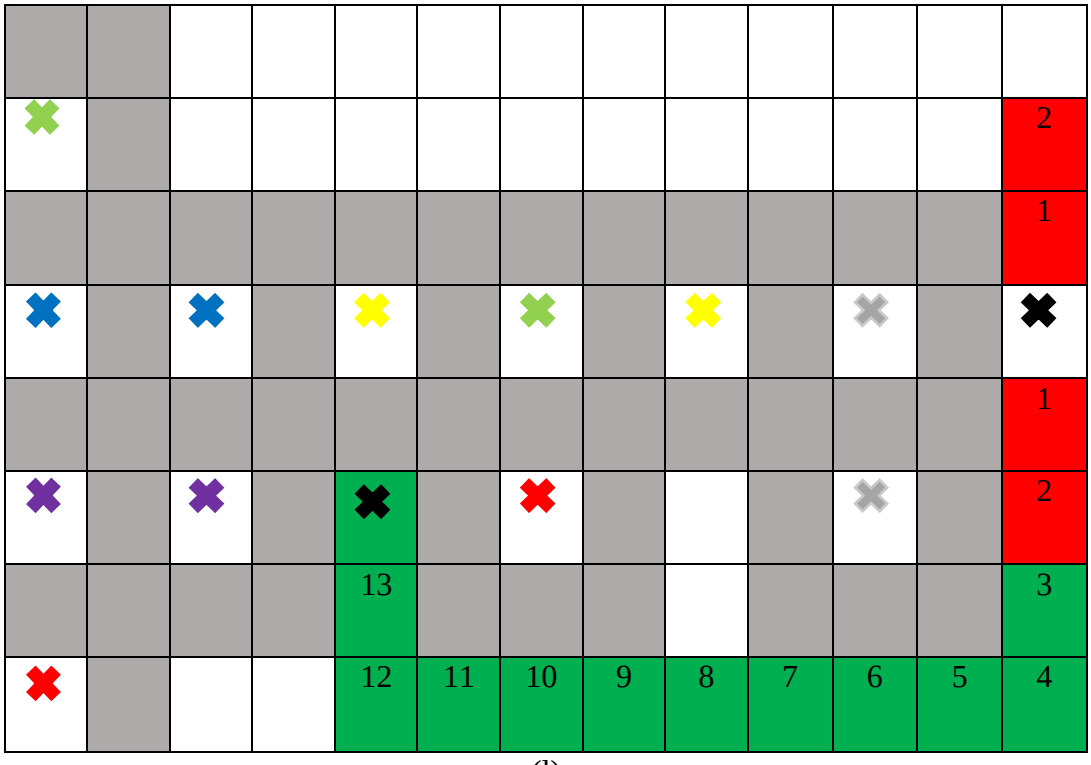
(i)



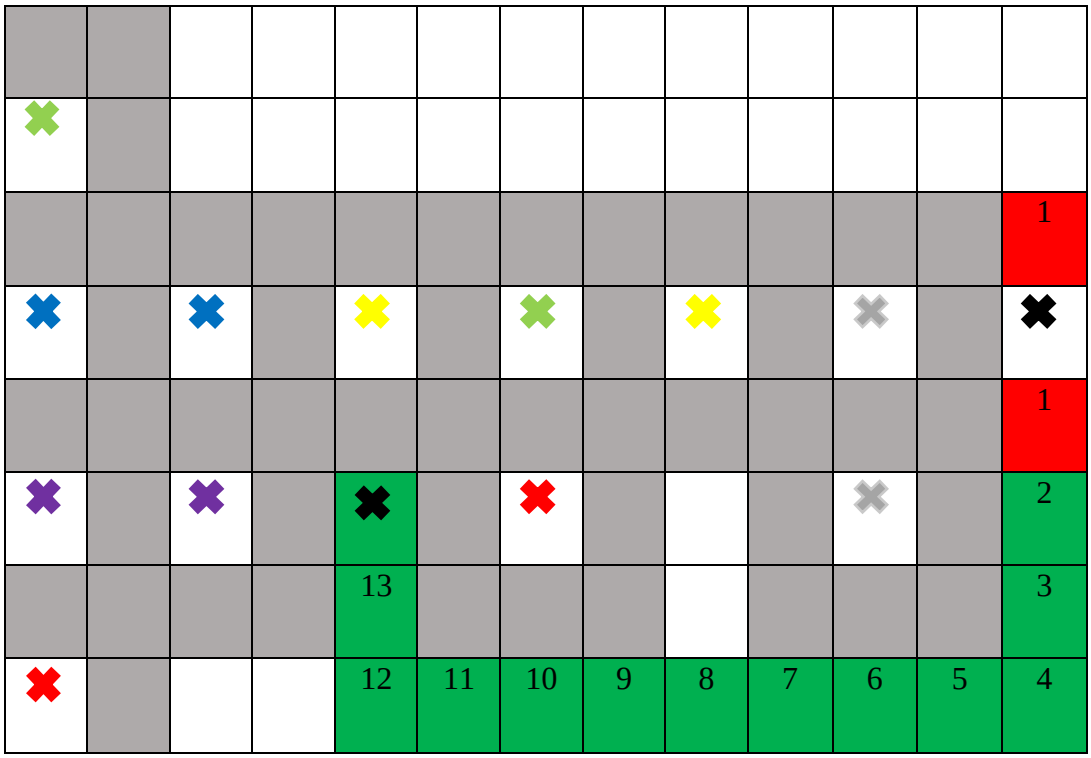
(j)



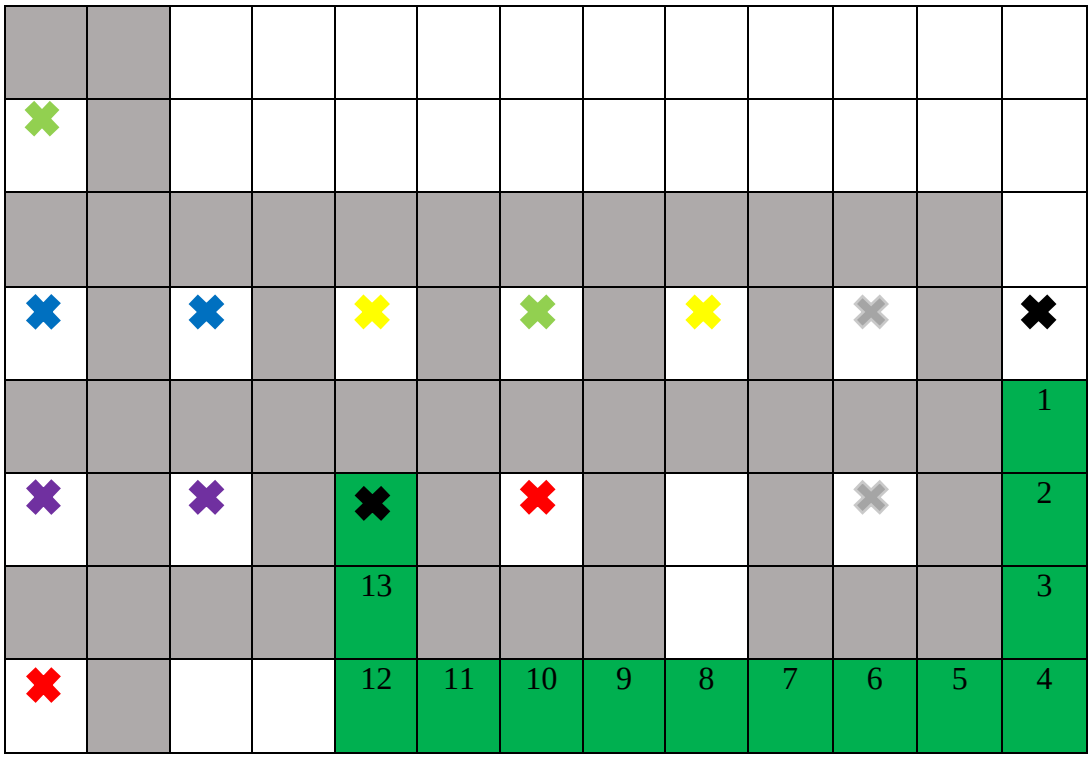
(k)



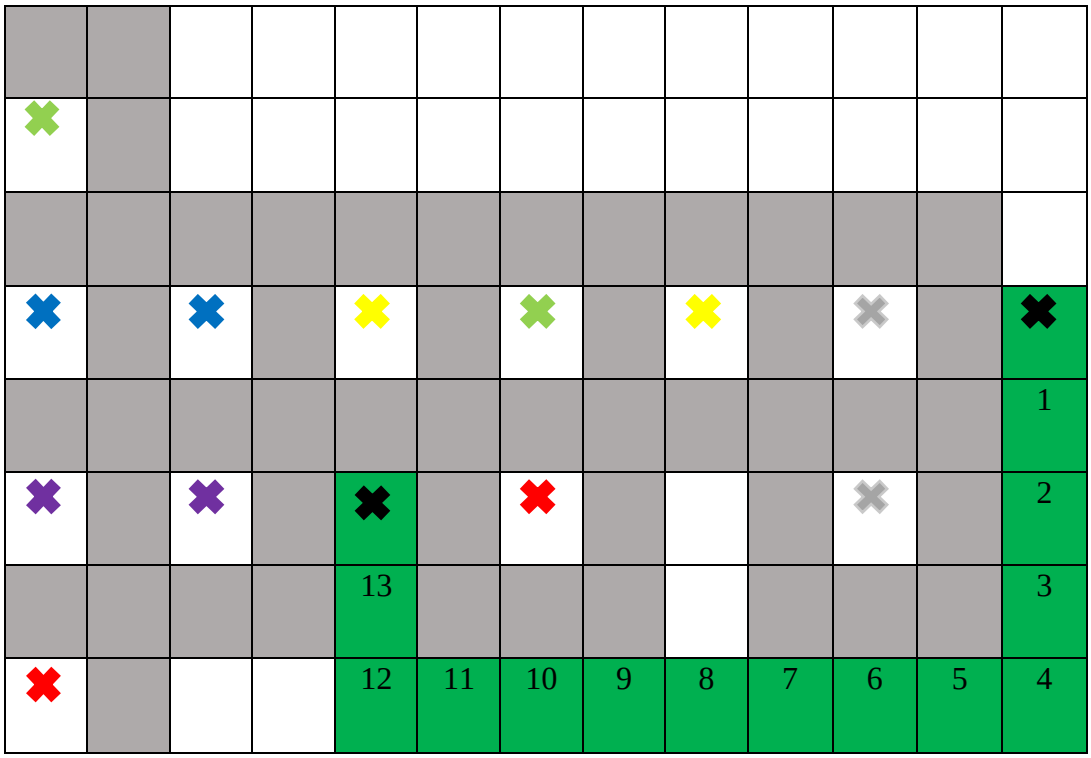
(l)



(m)



(n)

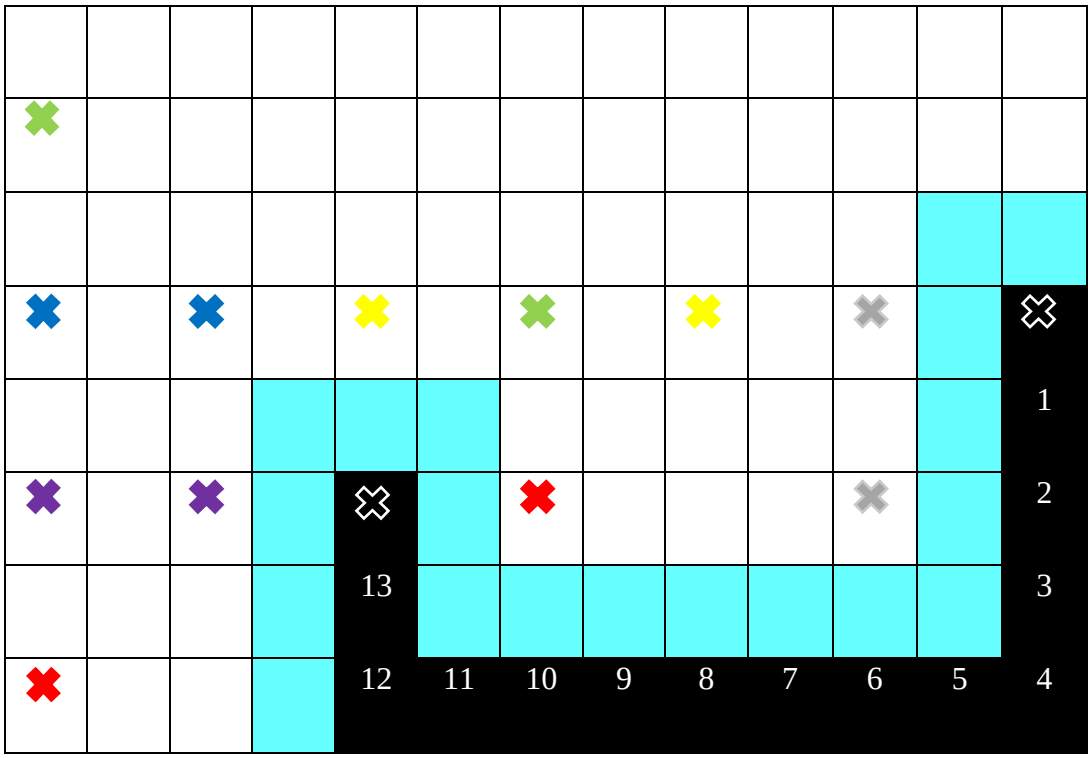


(o)

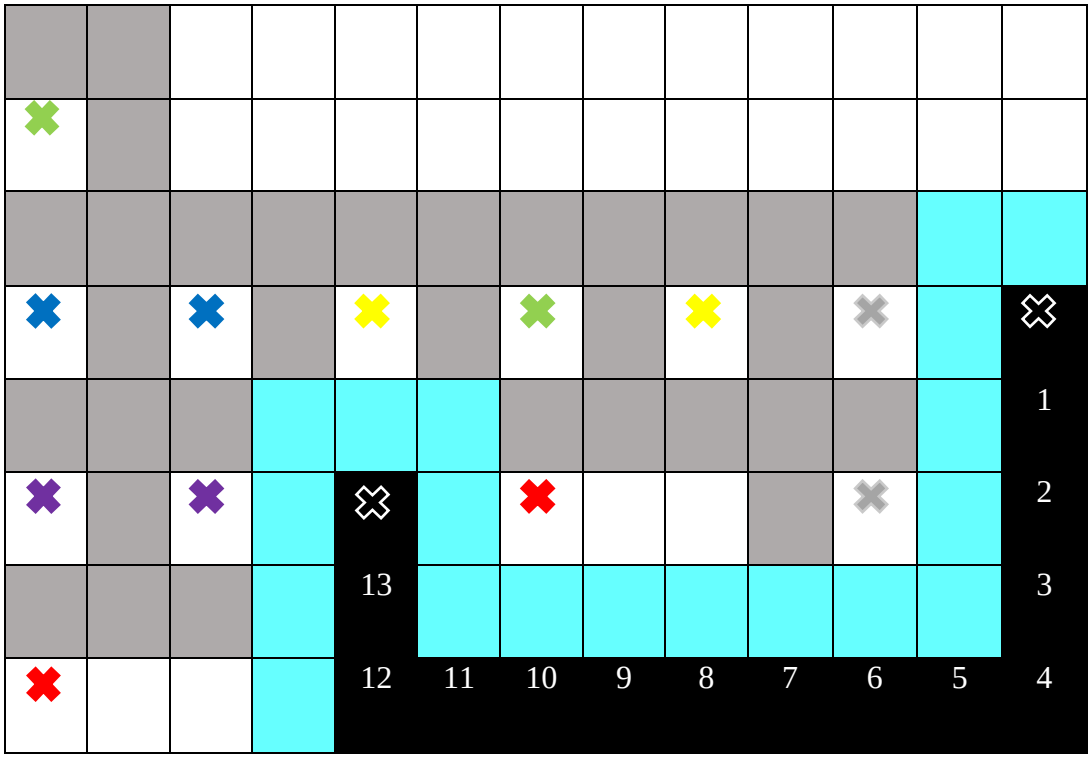
Figure 21: Step by step trackback process for source-destination I6

Figure 21 shows step by step how the trackback process works and the shortest path between the source and destination is found. Once the routing for this pair is completed, the program moves to the next pair and tries to route them. However, before starting to route the next pair, it is essential to un-pad all the source and destination that has been padded previously. It is even more important to pad around the routed path as well as the source and destination, so that the other expansions have at least 4λ distance from the nodes used by S6 based on the scalable DRC rules. This process is referred to as post-padding, where the surrounding nodes for the routed source and destination and also the nodes used to connect them are padded all around to confirm that all the other metal wires or sources and destination nodes are at least 4λ away from these nodes.

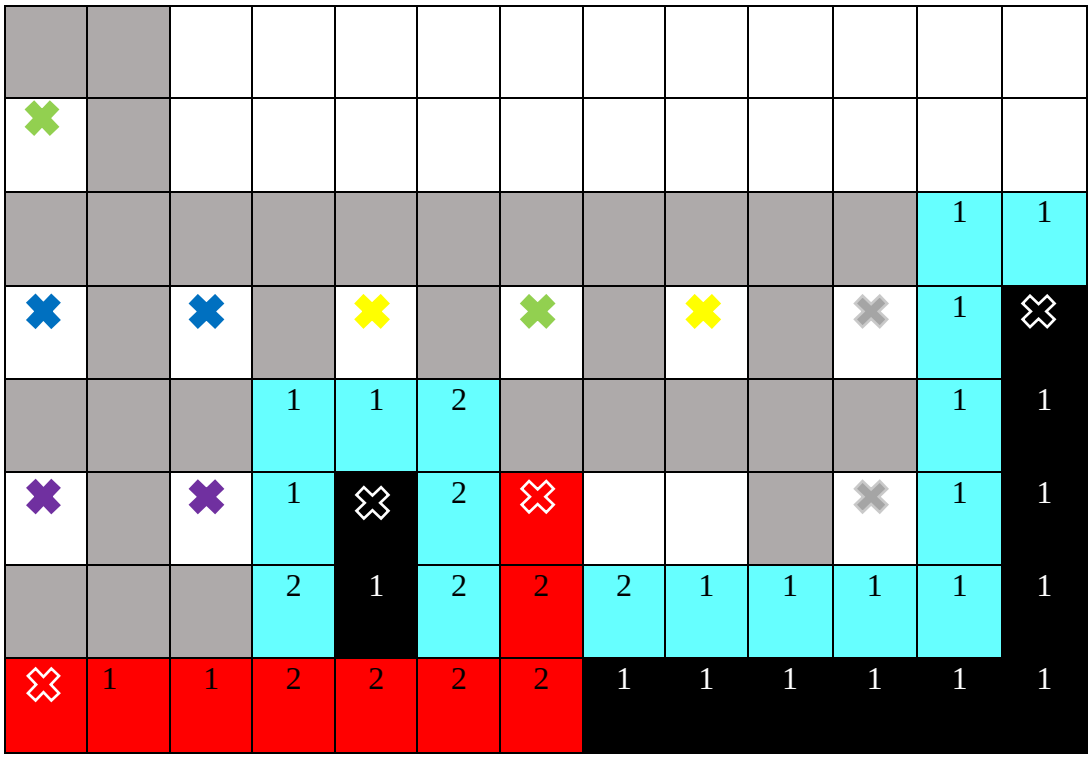
Figure 20(a) illustrates how the routed path along with the source and destination is padded around to fulfill the scalable DRC requirement. In the Figure the black nodes marks the routing path. The



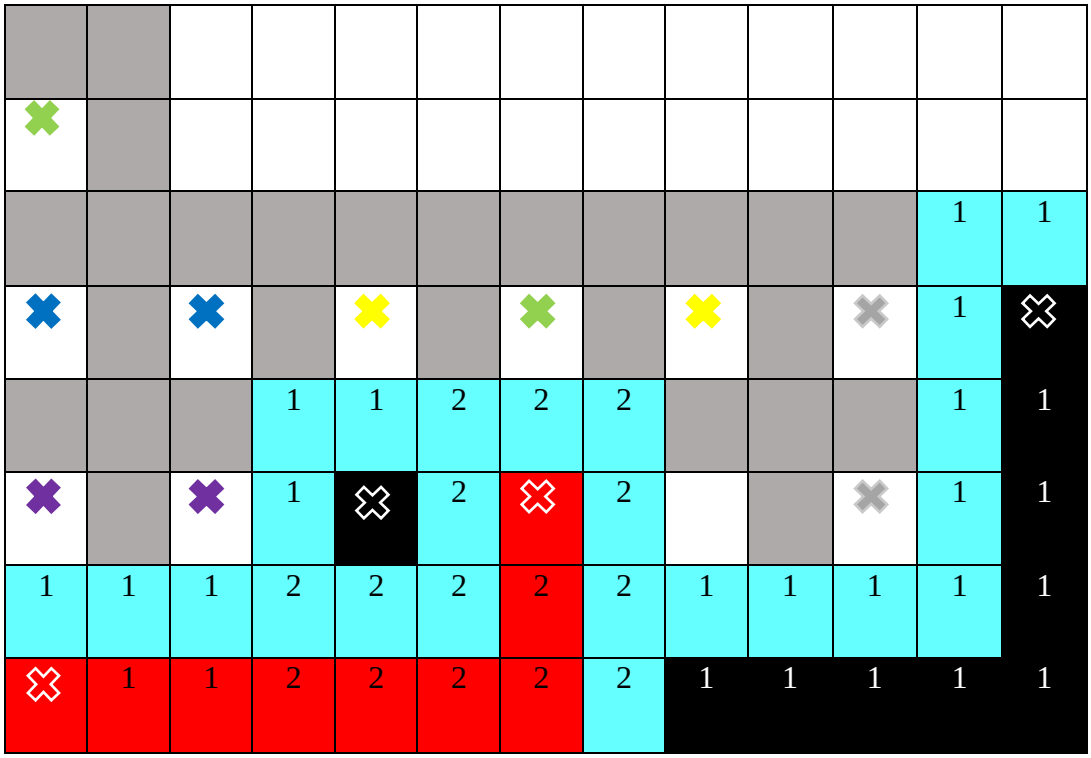
(a)



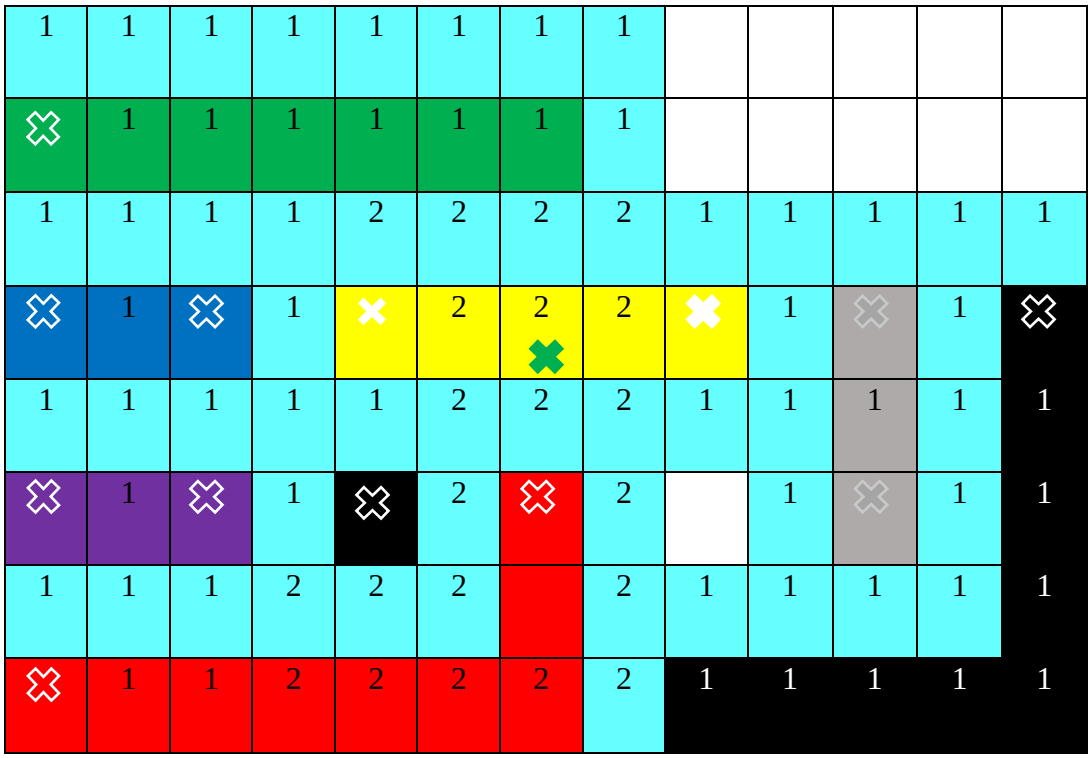
(b)


1 1
1 1
1 1
1 1
2 1 2 2 2 1 1 1 1 1
1 1 1 2 2 2 2 1 1 1 1 1 1

(c)


1 1
1 1
1 1
1 1 1 2 2 2 2 2 1 1 1 1 1
1 1 1 2 2 2 2 2 1 1 1 1 1

(d)



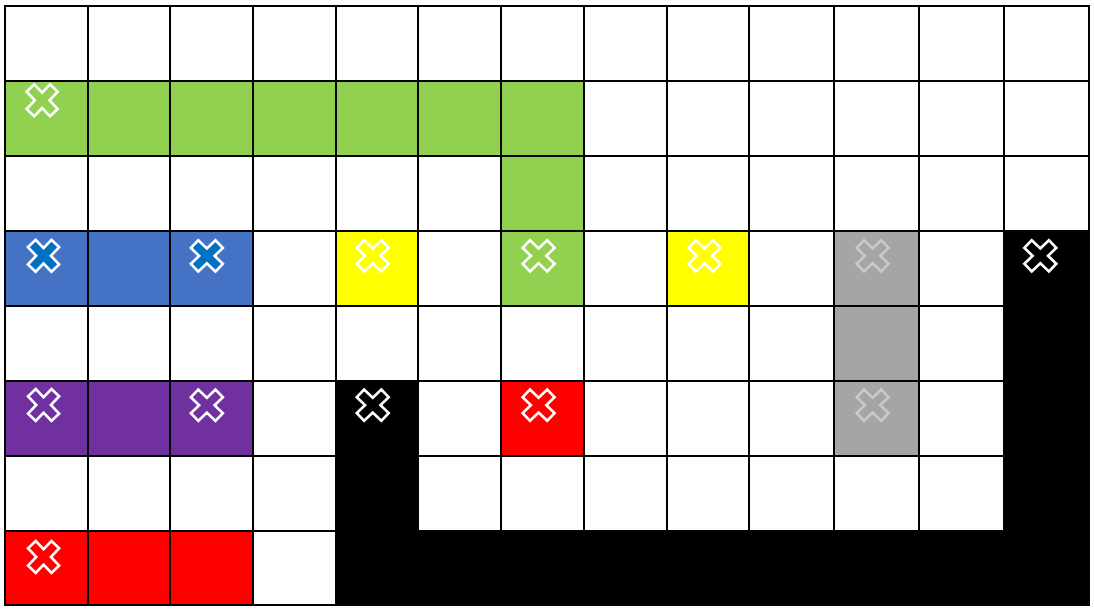
(c)

Figure 22: (a) Post-padding for S6, (b)Pre-padding for I4, (c)Trackback for I4, (d)Post-padding for I4, (e)A top view of the final floor plan showing all the connections

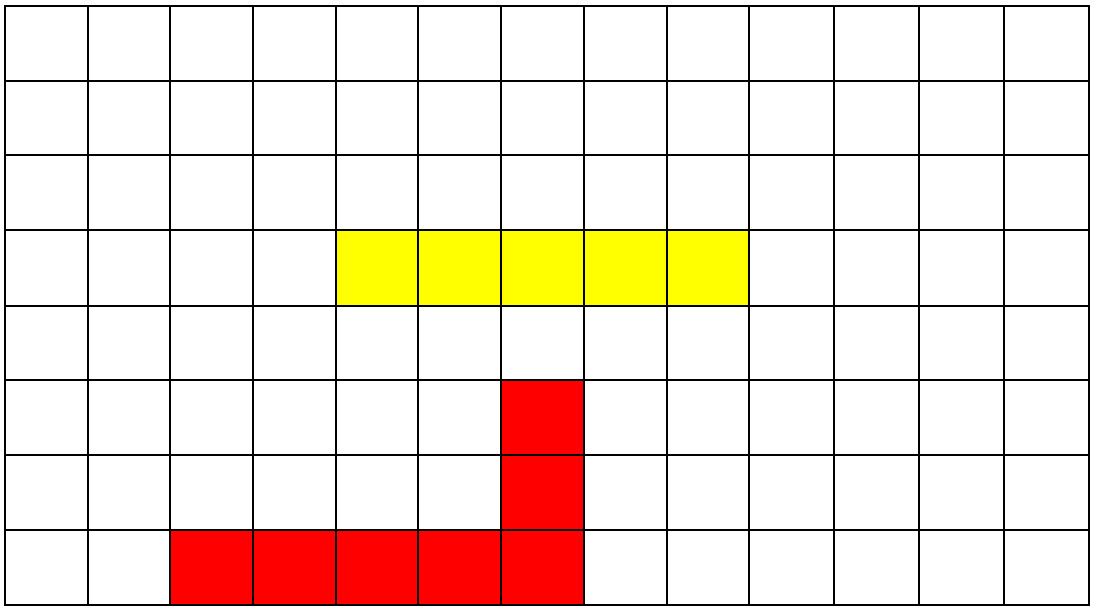
crosses signify the source and the destination node. The indigo nodes are the paddings associated with this pair and will not allow expansion from any other node. In Figure 22(b) the grid is set up for routing I4. Therefore, all the un-routed pairs are pre-padded, which is signified by the grey nodes, except the pair to be routed and the pair(s) previously routed. However, the connection between the source and destination of I4 requires a higher metal layer, since it is impossible to route them using only one metal. Therefore, overlap of metals can be seen in Figure 22(c) which portraits the trackback of I4. To avoid any confusions, each node is marked by a number which denotes the metal level (or the z value) of that node. Figure 22(d) shows the post-padding for both I4 and O7. All the other pairs could be picked one after another and the connection between respective source and destination can be established using the same methodology. Figure 22(e) shows a top view of the final state of the grid after all the connection has been established. The

designated color for each pair marks the routing tracks for that pair in the grid. Indigo represent the post-padded nodes. The number printed on each node represents the maximum z value for that node.

It could be easily seen that, it took two layers of metals to complete the circuit. Figure 23 presents the layer by layer occupancy of the grid since it is easier to understand and visualize compared to Figure 22(e) and in the remainder of this work, a complete routing solution will be presented using the equal numbers of Figures as the metal layers used to successfully route the circuit as Figure 23. Each Figure will illustrate the connections only at one layer starting from the lowest layer to the highest layer. The color designated to each pair will present the connection of that pair in every layer. The padding will not be shown in these Figures in order to make the routing result easily understandable.



(a)



(b)

Figure 23: (a) Routing solution for the given placement at metal layer 1, (b) Routing solution for the given placement at metal layer 2

Given a 3-D bounding box and a given routing order of signals our automated tool outputs the routed paths for all the source-destination pairs, that can be successfully routed. From the output generated by the program, maximum and minimum values on the x and y dimensions of the boundary box are used to calculate the total area. The maximum z value denotes the number of metal layers used to route the circuit. The ordering of the signals to be routed also plays a vital role, in this work, we systematically search through the ordering of the signals to identify successful routing solutions. Below is the final step by step version of the algorithm.

1. Create a 3D grid and provide each node of the grid with a unique position number.
2. Place the source and destination contacts in the grid, based on the placements and input distributions explored in chapter 3.
3. Decide the order in what the pairs will be routed.
4. Pre-pad all the un-routed source and destination pairs except for the one to be routed currently.

5. Expand from the source node and keep expanding until the destination node is reached.
    a) If the destination is reached: Trackback from the destination to the source to figure out the shortest path to establish the connection between them then proceed to 6.
    b) If the destination is not reached: Print no path found and proceed to 7.
6. Post-pad the routing track as well as the routed source and destination.
7. Reset the value of all the unused but expanded nodes to ‘0’.
8. Check if there are any pairs left for routing:
    a) If yes: repeat steps 3-9.
    b) If no: the routing is complete.

Chapter 5

Experimental Result:

This chapter presents the overall results of our investigation. All three feasible placements identified in chapter 3 are fed to the automated routing tool with various ordering of signals to be routed to identify the smallest feasible routing area. The outcome is then compared with previous work. At this point it is essential to understand how the tool works and how the total area will be calculated from the output of the automated tool. First of all, the contacts will be placed on a grid. The grid size will be as small as possible since it will just be accommodating the contacts with the required spacings. The z value of the grid is set to large number (10), which means it allows the contacts to use as many metal layers as possible to establish a successful routing. All the possible placements explored in chapter 3 are fed into the tool. We iterate through all the permutation of signal ordering of routing the input and internal signals. Once a successful routing is found, the following equations are used to calculate the total area.

$$\text{Length} = \{4 + (\text{maximum\_x\_value\_of\_the\_grid} * 4)\}\,\lambda$$
$$\text{Width} = \{4 + (\text{maximum\_y\_value\_of\_the\_grid} * 4)\}\,\lambda$$
$$\text{Total Area} = (\text{Length} * \text{Width})\,\lambda^2$$

The input distribution system is selected based on the space available in the grid. Same metal layer is preferred most, since it has the highest possibility of a successful routing. However, if the grid size doesn't allow all inputs to be connected on a single metal layer, metal layers are gradually increased from 2 to 4 metal layers to fit all the inputs in the grid.

5.1 Placement1:

First placement that we explore is from Figure 13, where all the basic blocks are placed side by side within the same row. The only input distribution that can provide inputs to this placement is to route the multiplexer data inputs over 4 metal layers without increasing the area is the quadruple metal layer system. Figure 24 shows this input distribution. The Figure contains two main rows and the second column of the bottom row is split into 4 cells to represent use of multiple layers of metal. The top most row and the left most column is there to fulfill the minimum spacing requirements set by DRC. Each cell of the grid is assumed to be 4λ. Therefore, the following setup has a length of 80λ and a width of 8λ, which sums up to a total area of $640\lambda^2$.

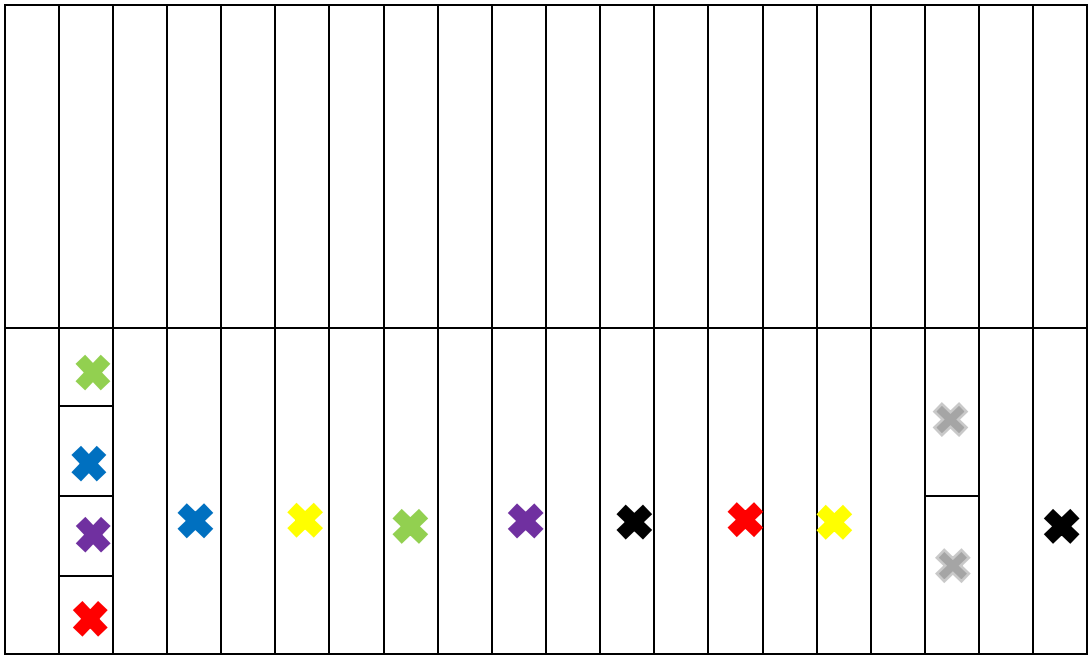


Figure 24:Placement 1 with no extra routing track

The position of all the contacts were fed to the automated tool to figure out if there is any feasible routing scenario for these placements. Several attempts were made by changing the ordering of the pairs to be routed. However, all the attempts failed for this placement. Therefore, we conclude that

there is no possible routing solution for this placement. Next part is to provide this placement with an extra routing track and check for a successful routing. Hence, the scenario shown in Figure 25 was fed to the automatic routing tool. Since, this case has enough space to accommodate input signals from two metal layers, which increases the possibility of successful routing, we use two layers of metal to distribute the data inputs as shown in Figure 25. Even tough, this setup has the same length as the previous one, because of the adding of an extra track with required spacing increases the width by 8λ, which makes the total area $1280\lambda^2$. This placement produced a successful routing solution.

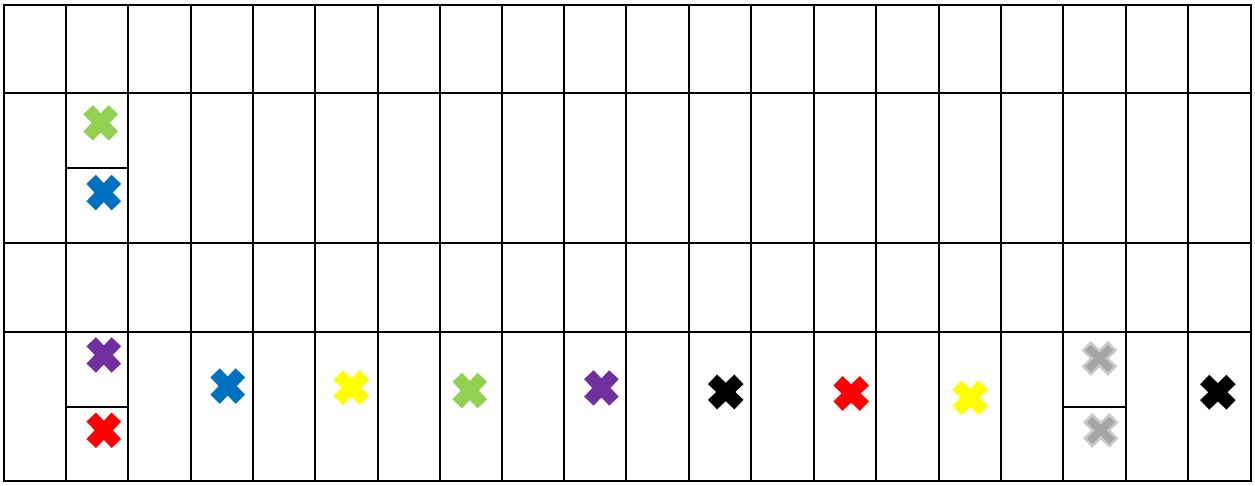
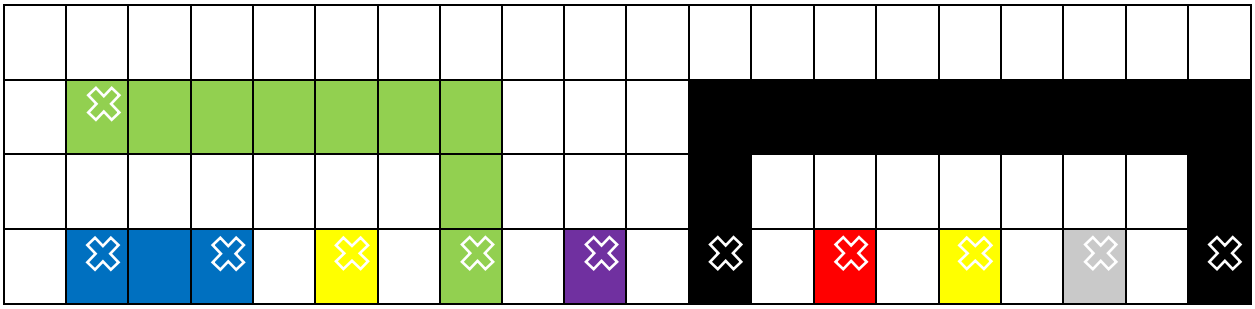
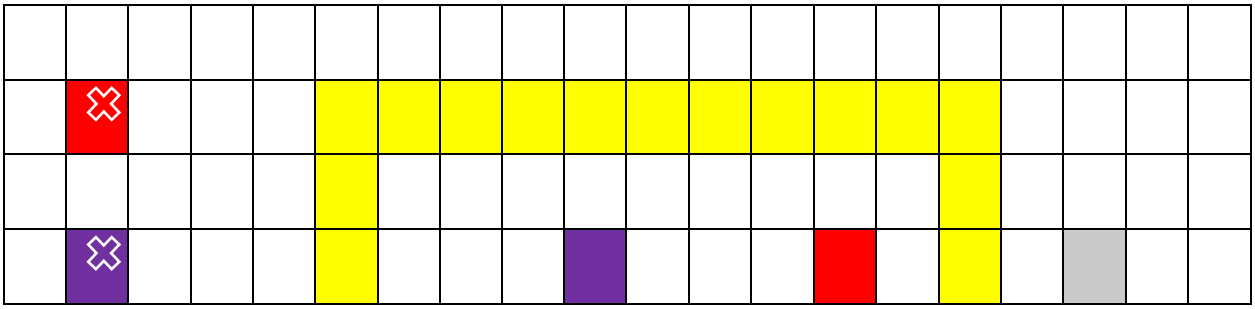


Figure 25: Placement 1 with an extra routing track on top

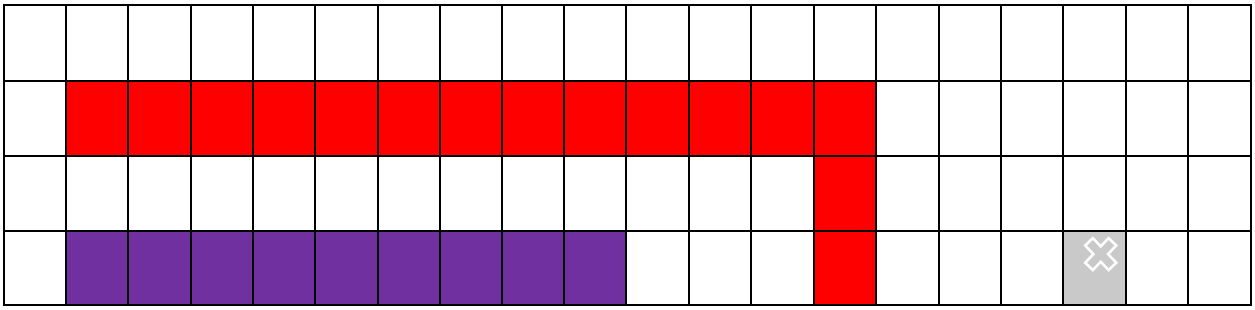
The successful routing is presented in a layer by layer fashion as shown in Figure 26:



(a)



(b)



(c)

Figure 26: Routing result of placement 1 (a) metal layer 1, (b) metal layer 2, (c) metal layer 3

5.2 Placement 2:

Next placement attempted is from Figure 14 of chapter 3. This placement has a length of 56λ and a width of 16λ, which gives it an area of $896\lambda^2$. The width of this placement is wide enough to distribute the multiplexer data inputs on two metal layers as shown in Figure 27. However, even this placement did not produce any successful routing solution after varying the ordering of the signals to be routed. Therefore, a similar approach was taken for this placement as well by adding another routing track at the middle as shown in Figure 28. The routing track was added at the middle to give all the basic blocks

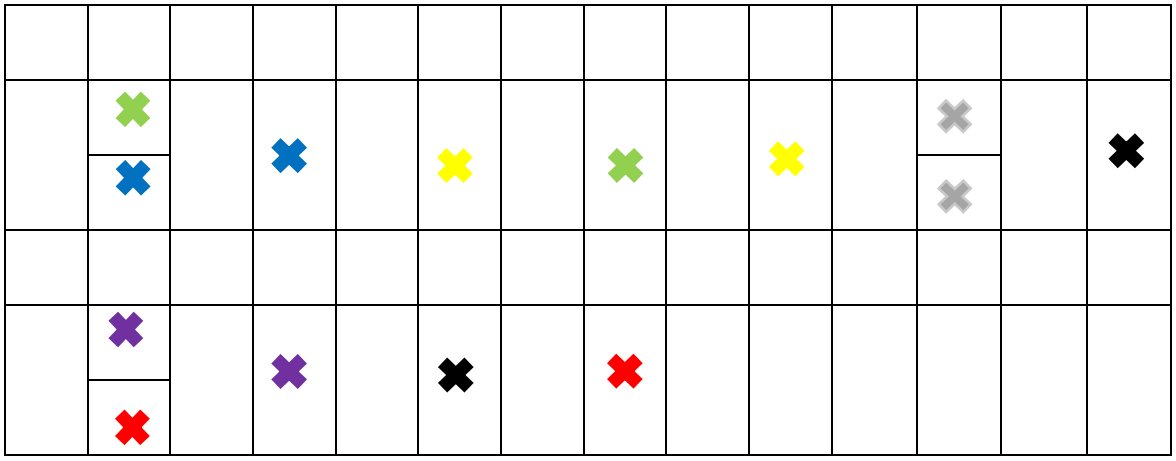


Figure 27: Placement 2 with no extra routing track

equal accessibility to the extra track. The width of the grid increased by 8λ because of this additional track. That changed the total area of the grid from $896\lambda^2$ to $1344\lambda^2$. The bigger placement produced a successful routing when fed to the program. Figure 28 illustrates the updated placement after adding an additional routing track at the middle:

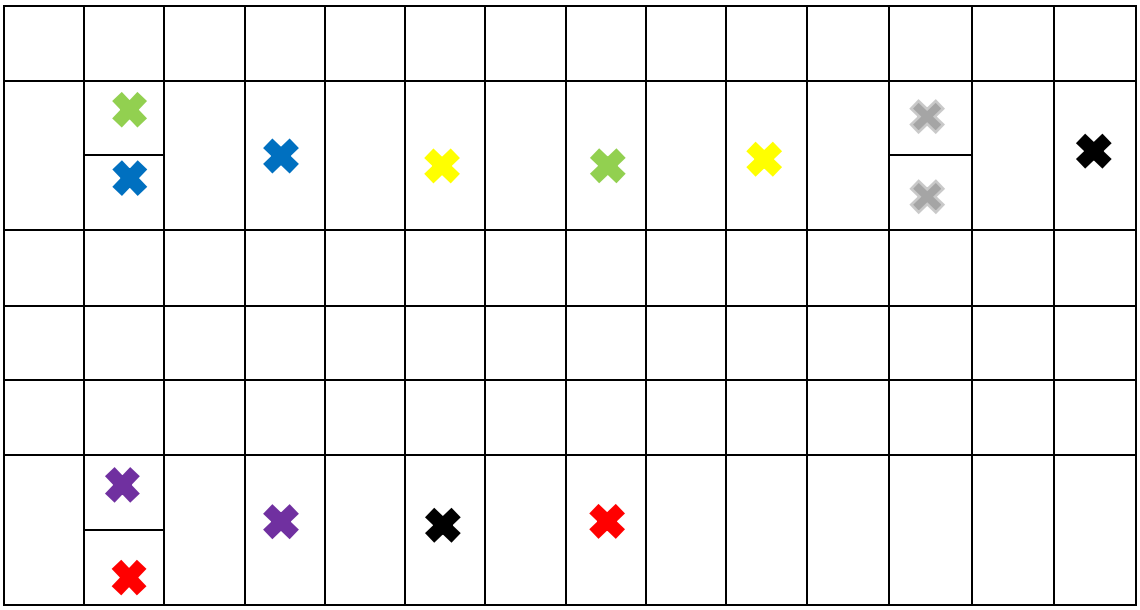
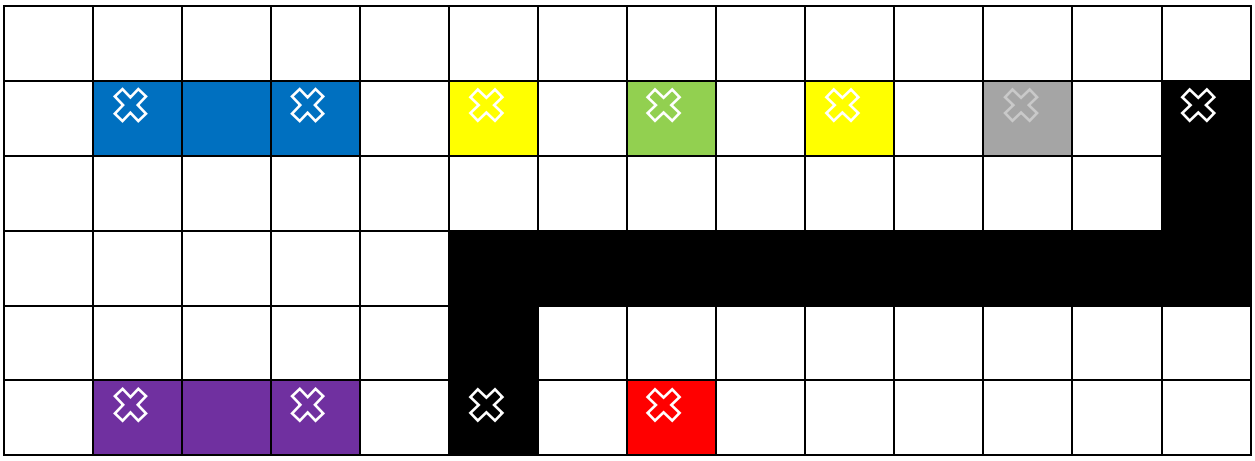
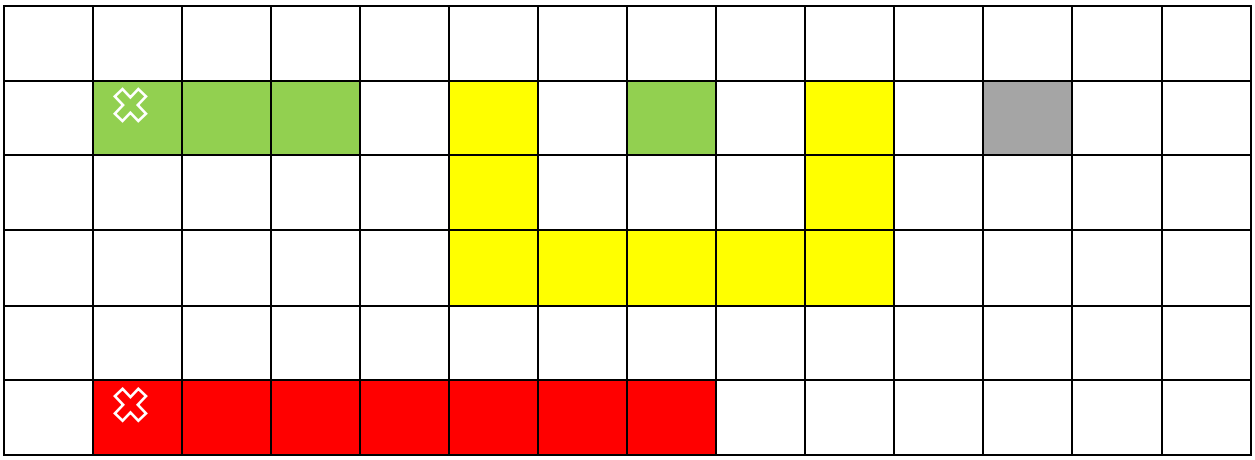


Figure 28: Placement 2 with an addition routing track

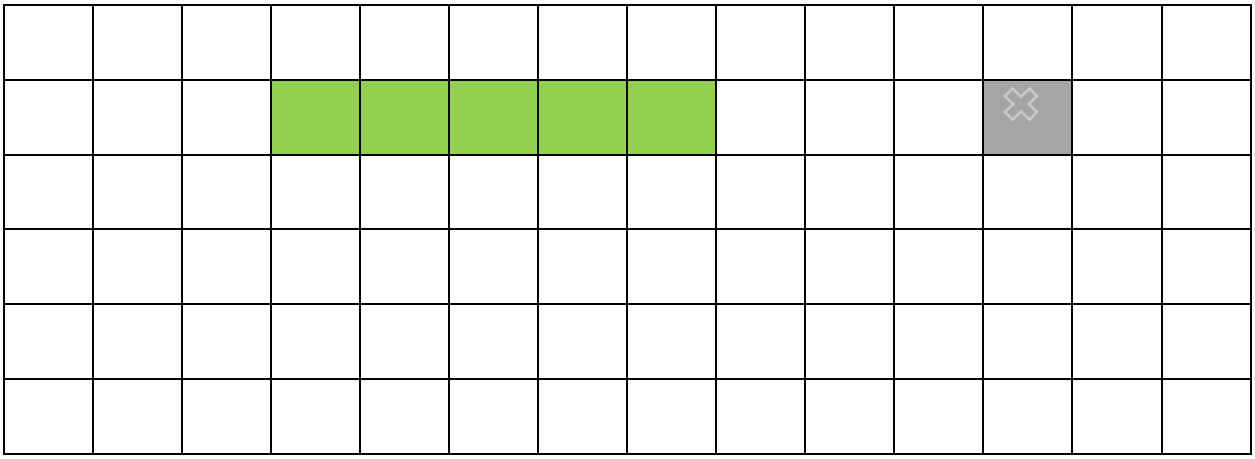
The result for the successful routing of placement 2 is presented below using the layer by layer routing system:



(a)



(b)



(c)

Figure 29: Routing result of placement 2 (a) metal layer 1, (b) metal layer 2, (c) metal layer 3

5.3 Placement 3:

This placement is taken from Figure 15. In this placement, all the basic blocks are placed on top of each other. However, this placement needs an extra 8λ spacing in between basic block 2 and 3 since these two blocks are not sharing the poly-silicon. This placement has just enough width to accommodate distributing multiplexer inputs on two metal layers as shown in Figure 30. After counting in all the required spaces, this placement is 32λ long and 32λ wide, which makes an area of $1024\lambda^2$. This placement produces a successful routing solution and the area is by far the smallest compare to all the others as shown by table 2.

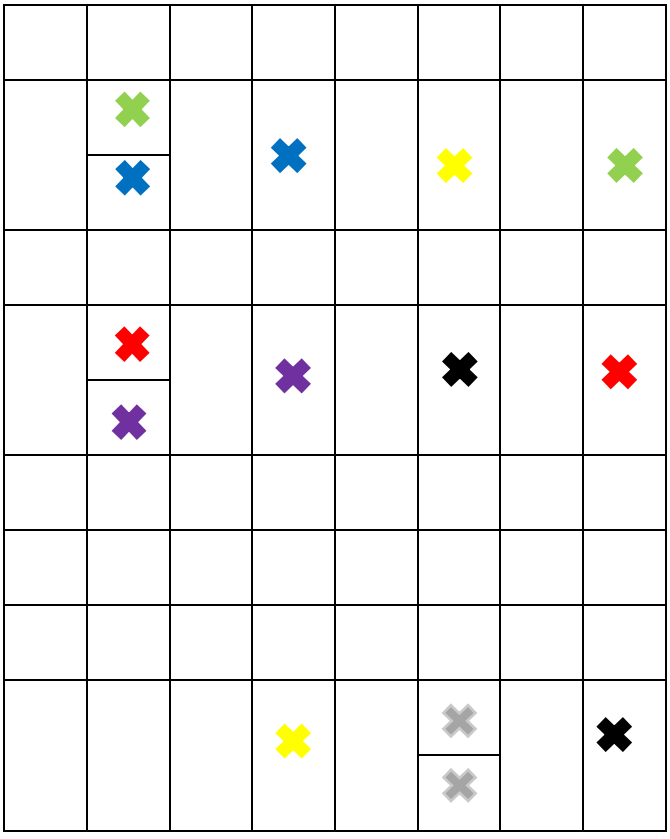
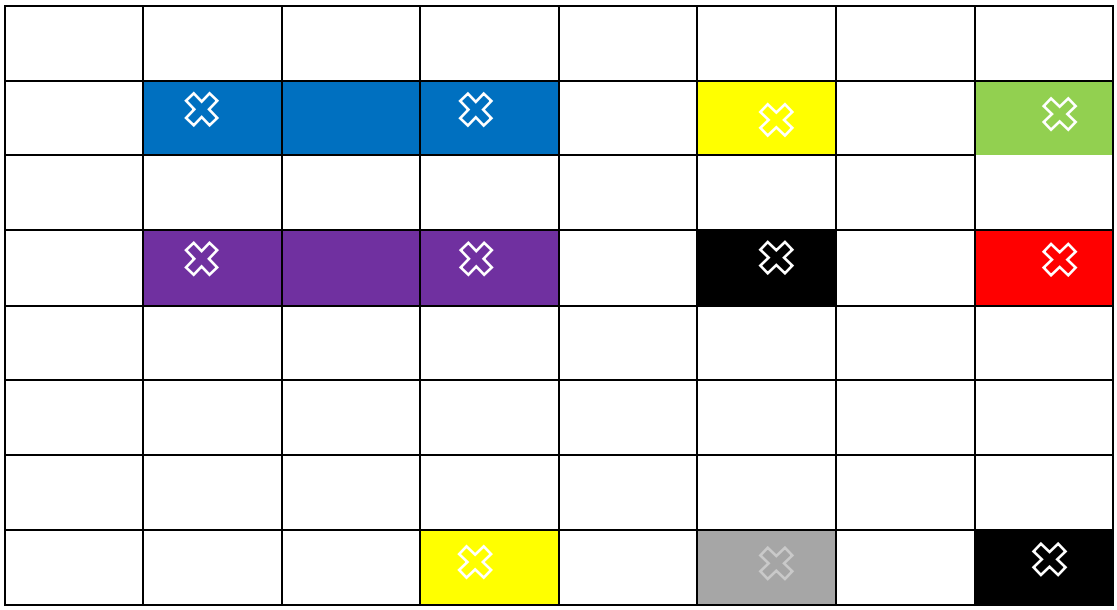
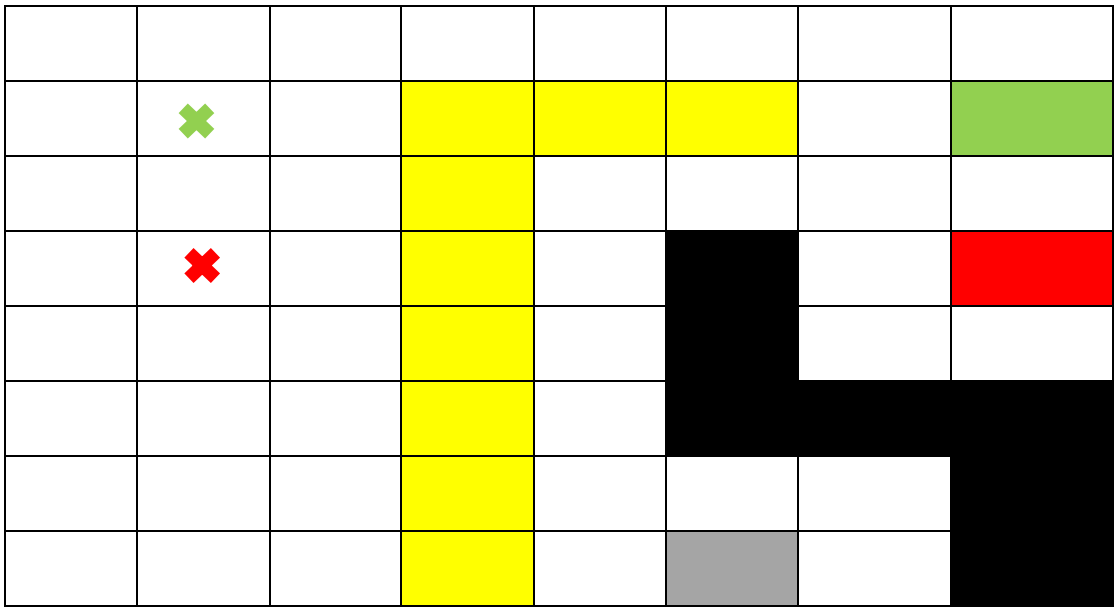


Figure 30: Placement 3 with no extra routing track

Figure 31 shows the layer by layer routing solution for the current placement. In the next sub-section, this smallest routing solution is implemented as physical layout using Magic and the layout is presented in the later part of the report.



(a)



(b)

(c)

Figure 31: Routing result of placement 3 (a) metal layer 1, (b) metal layer 2, (c) metal layer 3

Area for implemented layout	Current Implementation /VPR area model [3]	Current Implementation /COFEE area model [5]
Placement 1: $1280\lambda^2$	1.33	1.38
Placement 2: $1344\lambda^2$	1.4	1.45
Placement 3: $1024\lambda^2$	1.01	1.11

Table 2: Comparison between the previous works and current theoretical layout area

5.4 DRC Spacing Adjustments & Physical Layouts:

The smallest successful routing (placement 3) found in the previous section has been interpreted as an actual layout to account for additional area requirement to create a functional 4:1 multiplexer. Magic VLSI design layout system was used to implement the layout. There are multiple tech files in the magic library. Even though the standard tech file follows the lambda based rules very strictly, many other tech files change these rules slightly and have its own version of DRC rules. Unfortunately, the standard tech file in Magic does not allow the use of stack via in the layout, which is a crucial part of this investigation. Therefore, a different tech file was used to implement the layouts which had slightly different DRC rules, which includes having a contact size of 5x5 instead of 4x4 for metal 2 and higher. The spacing between two poly-silicon's also changes from 2λ to 3λ. The overhead poly silicon length changes from 2λ to 3λ as well. All these facts were taken into consideration and the spacing requirement from area model [3] was redefined to get a competitive comparison between the model and the layout results.

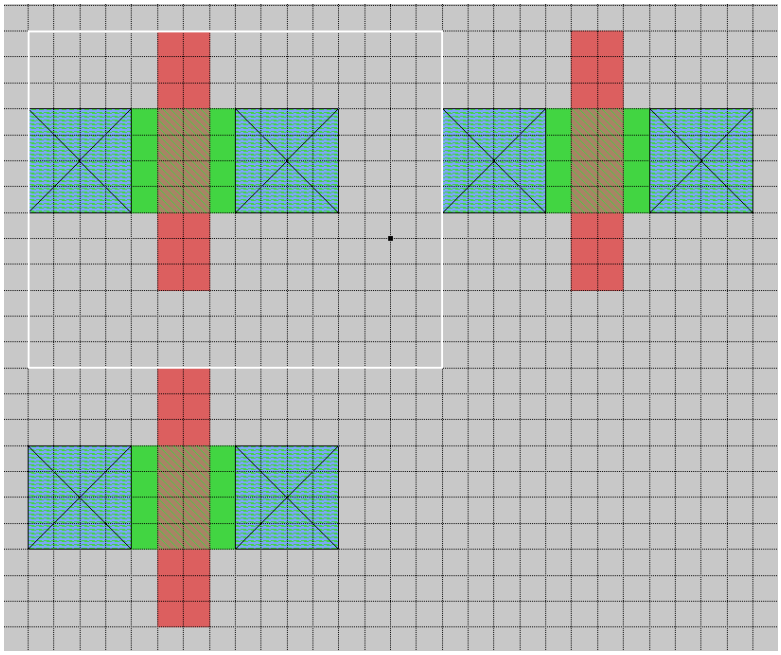


Figure 32: Updated area model based on CMOS 8 metal layer tech file

Figure 32, shows the updated spacing required to layout one transistor. In the Figure, each small square represents an area of 1λ. Therefore, it can be easily seen that, the minimum height of the cell changed from 10λ to 13λ. Consequently, the minimum total area required to layout a transistor changes from $160\lambda^2$ to $208\lambda^2$.

Figure 33 shows the layouts for placement 3, which achieved the smallest layout area:

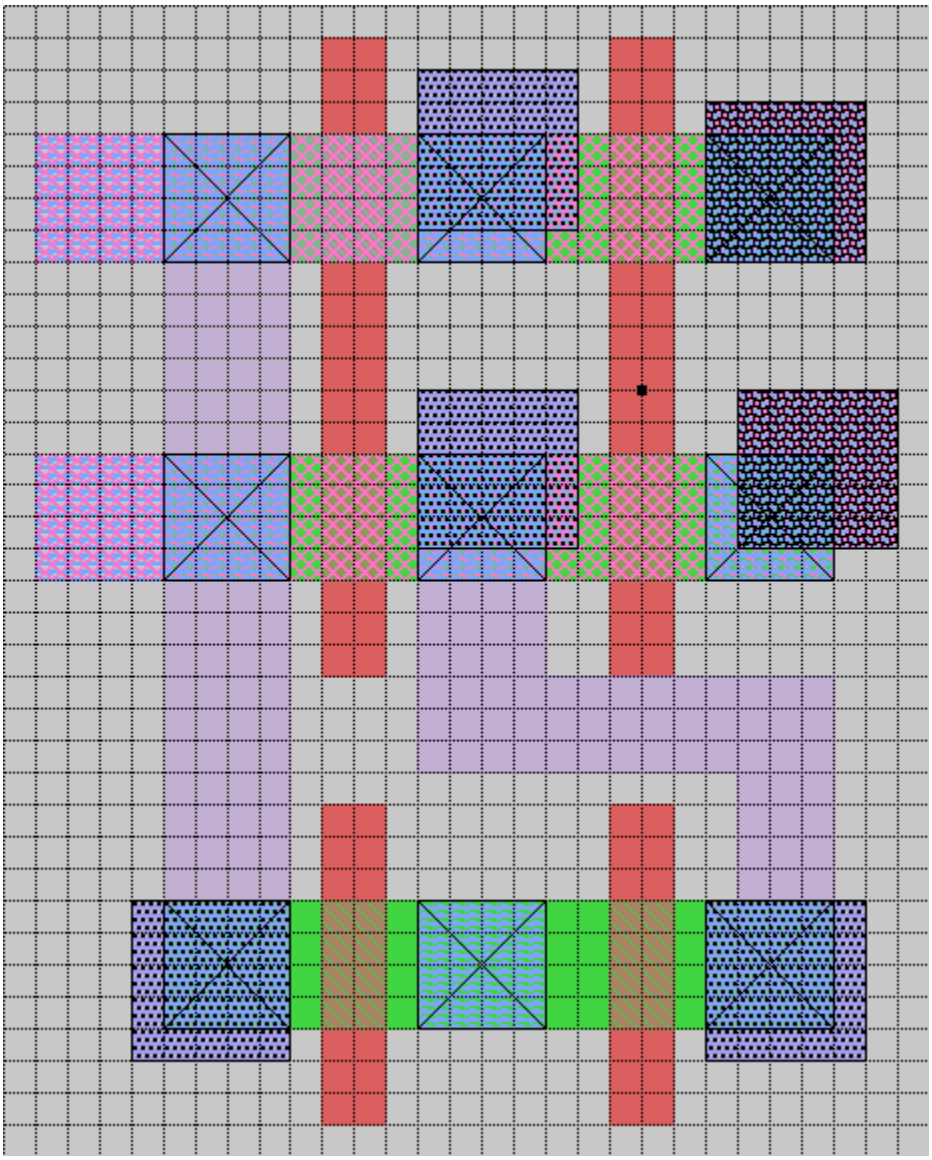


Figure 33: Layout for placement 3

Layout 1 which represents placement 3 has the smallest layout area from all the other placements. It has a length of 37λ and a width of 31λ, which makes the area $1147\lambda^2$. After adjusting the DRC rules based on the tech file used to implement the layout, the area of a 4:1 multiplexer based on the equation of model [3] was calculated as $1248\lambda^2$. Therefore, the current layout is only 92% of what the model [3] predicts. Current layout is also smaller than the prediction of model [5] and only requires 96% area compare to the prediction.

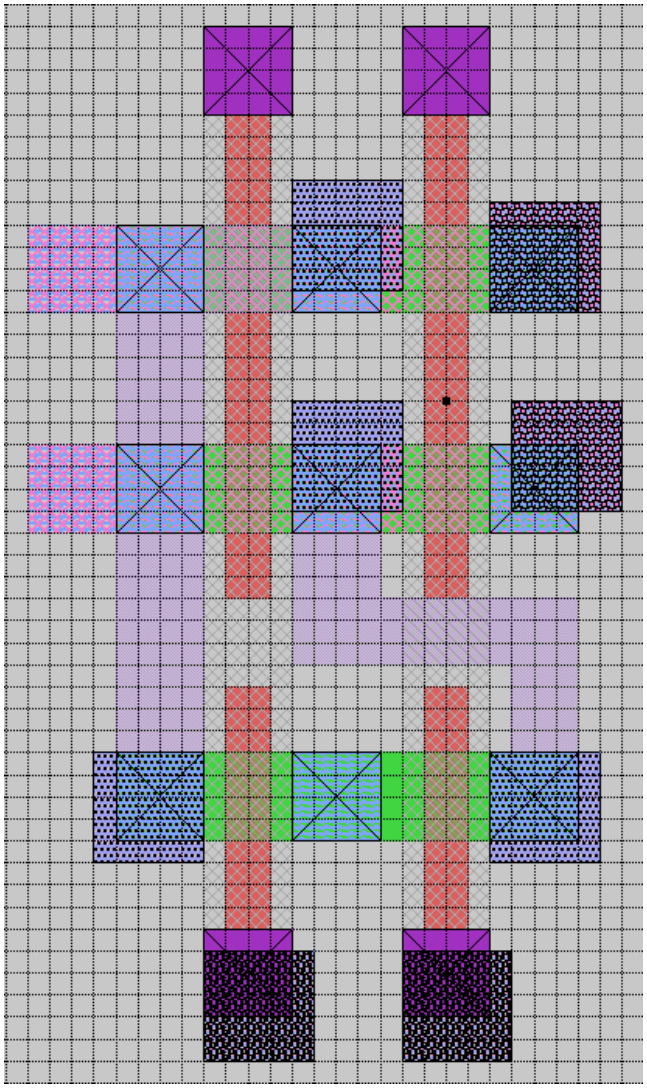


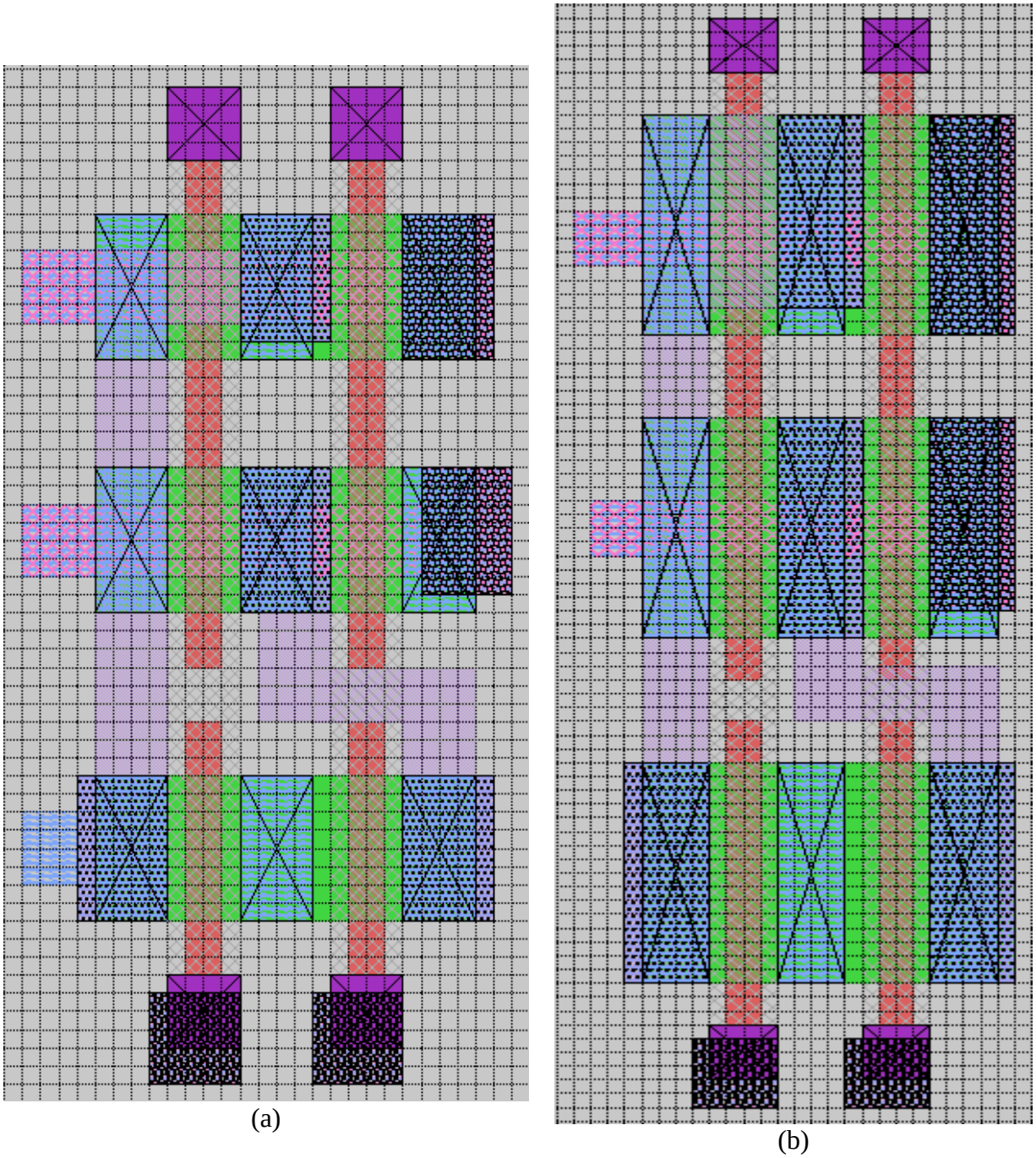
Figure 34: 2-Input LUT with Poly-Silicon contacts

However, it will be inappropriate to compare the current layout area with the layout area of [4], since that work implements a complete functioning 4:1 multiplexer layouts which also considers the area connecting multiplexer select signals to the poly-silicon. Therefore, before comparing the multiplexer layout with [4], a complete functioning 4:1 multiplexer layout was implemented by adding in the poly-silicon signals in the current layout. The updated layout is shown in Figure 34.

This layout has a length of 50λ and a width of 31λ, which sums up to total area of $1550\lambda^2$. The previously implemented layout [4] had a layout area of $2208\lambda^2$ for 3 metals [29], which is 30% bigger than the current layout and a layout area of $2805\lambda^2$ for 2 metals, which is 45% bigger than the current layout. However, current layout uses 4 layers of metals instead of 2 or 3 layers of metals. Current investigation was illustrated even further by implementing completely functioning 4:1 multiplexers with higher transistor sizes (e.g: 2x, 4x, 6x) and comparing the results with model [3], [4] & [5]. The layout of the multiplexer using the transistors of size 2x takes an area of $1530\lambda^2$ and the layout using the transistors of size 4x takes an area of $2220\lambda^2$. Table 2 compares these layout areas with the adjusted values of previous equation based models [3] [5], as well as the previously implemented layout area [4,26].

Area for implemented layout	Current Implementation/ VPR area model [3]	Current Implementation/ COFEE area model [5]	Current Implementation/ area model [29]
Minimum width transistor: $1550\lambda^2$	1.24	1.29	0.70
2 x Minimum width transistor: $1798\lambda^2$	0.96	1.15	N/A
4 x Minimum width transistor: $2460\lambda^2$	0.79	1.14	N/A
6 x Minimum width transistor: $3090\lambda^2$	0.71	1.14	0.75

Table 3: Comparison among the current investigation and the previous works



(a) (b)

Figure 35: (a) 2-Input LUT constructed using 2x transistors, (b) 2-Input LUT constructed using 4x transistors

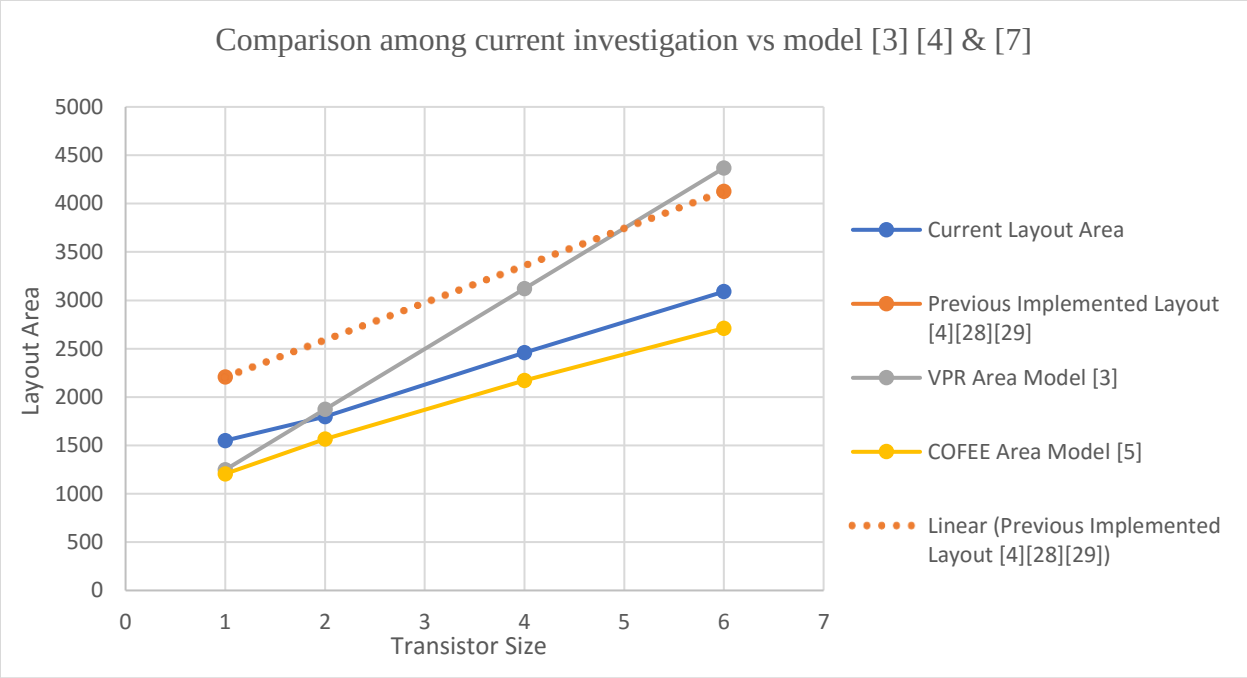


Figure 36: Comparison between current and previous works

Figure 36 compares the layout area of current investigation and previous equation based models for varying transistor sizes. It clearly shows that; current work gives a smaller layout area for circuits with bigger transistor width. Even though, the current model gives a higher layout area for minimum width transistor, but it considers the area for poly silicon contacts to make the circuit fully functional, which was completely ignored by the equation based models. Our layout also gives a smaller layout area for the minimum width transistor compared to the previously implemented layout [4].

To illustrate our claim even further, physical layouts of fully functional 3 and 4 input LUT's were implemented by using the 2-input LUT layout repeatedly. The physical layouts along with the area comparison are tabulated below:

Area for implemented layout	Current Implementation/ VPR area model [3]	Current Implementation/ COFEE area model [5]	Current Implementation/ area model [29]	Required number of metal layers
2 input LUT: $1550\lambda^2$	1.24	1.29	0.70	4
3 input LUT: $4125\lambda^2$	1.42	1.47	0.75	8
4 input LUT: $9438\lambda^2$	1.51	1.57	0.75	8

Table 4: Comparison among the current investigation and the previous works for 2,3 & 4 Input LUT

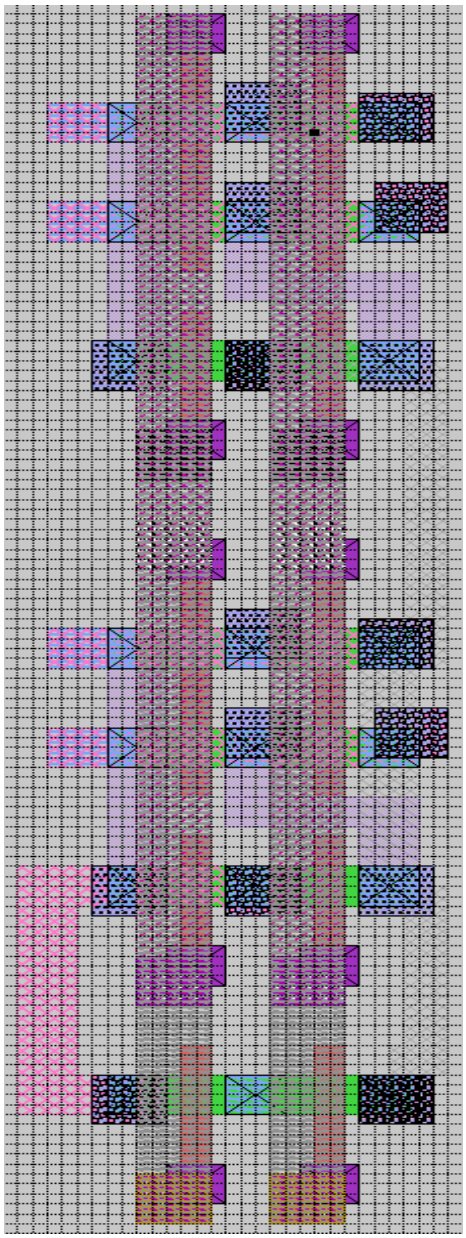


Figure 37: Physical Layout of a 3 input LUT

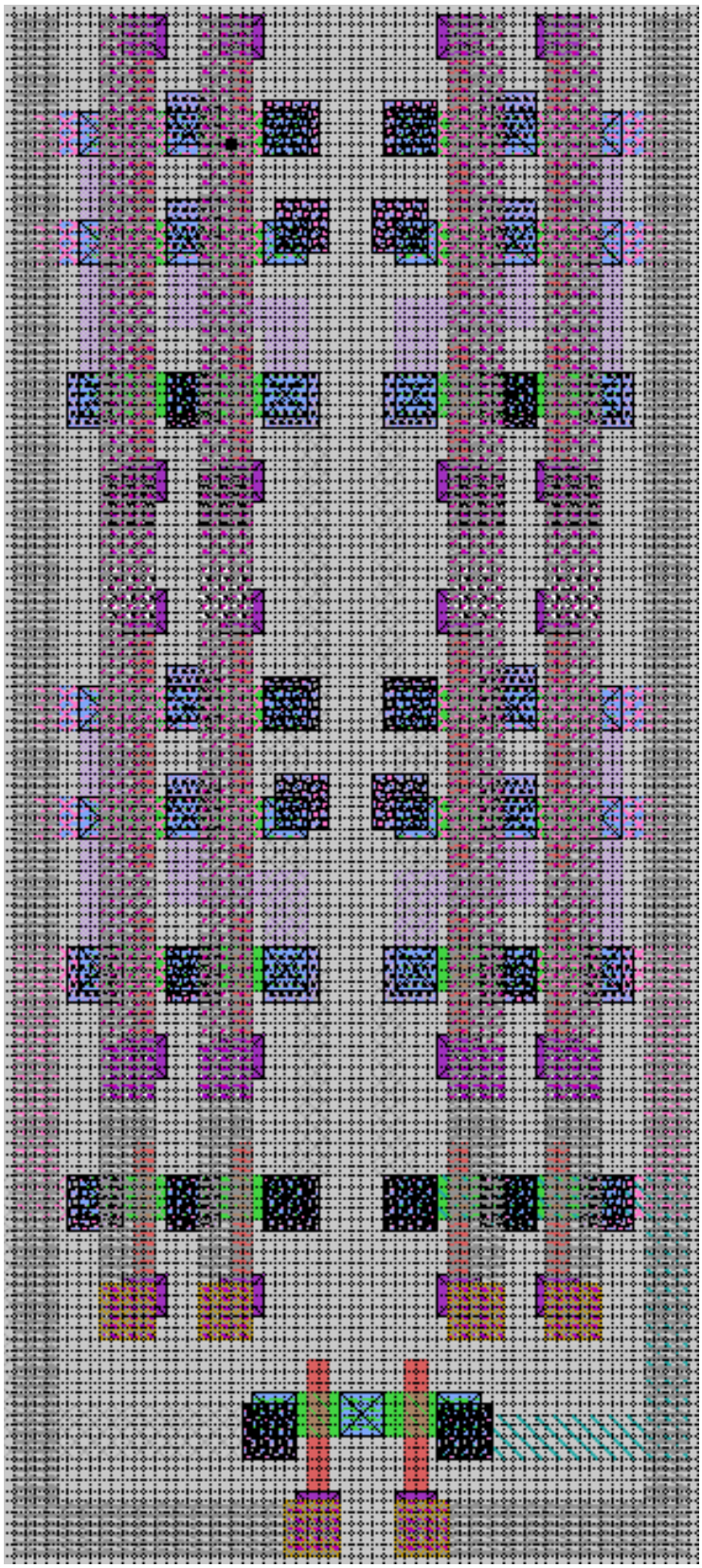


Figure 38: Physical Layout of a 4 input LUT

Chapter 6:

Future work & Conclusion:

There are yet a lot of scopes to improve this work which may produce even better results. First of all, implementing negotiate congestion (NC) in the software will be a huge improvement towards the efficiency of the automated tool. Without NC, the user is responsible for selecting the order of each pair and fed them for routing based on their order. There are scenarios, when the routing for the same placements fail several times because of the routing order of the pairs. However, it is impossible for the user to go through all the possible combinations and figure out if there is a successful solution. If NC is implemented successfully, the program itself will oversee selecting the orders of the pairs. That way the user doesn't have to choose the order of the pairs randomly.

Moreover, different versions of the tool could be developed based on the lambda rule of different tech files. That way the user does not have to make any manual adjustment to get an accurate comparison. It will also make the manual routing process very easier, where the layout designer just has to follow the path given by the program.

Finally, the described process of routing can play a vital role in reducing the layout area of a FPGA chip as a whole, which could be considered as very significant in terms of FPGA area efficiency. It can also be used to find out the smallest area possible for any given circuit. Moreover, it can assume the smallest are possible for a circuit with variable metal layers. All in all, the investigation fulfills the initial goals and produces a very significant result. Therefore, it can be said that the complete process of research and implementation was extremely successful.

Bibliography

1. I. Kuon and J. Rose. Quantifying and Exploring the Gap between FPGAs and ASICs 2009.
2. A.Amara, F.Amiel and T.Ea. FPGA vs. ASIC for low power applications. Microelectronics Journal 37(8), pp. 669-677.2006. DOI: 10.1016/j.mejo.2005.11.003.
3. V. Betz, J. Rose, and A. Marquardt, Architecture and CAD for Deep Submicron FPGAs. Boston: Kluwer Academic Publishers, Feb. 1999.
4. F. F. Khan and A. Ye. Measuring the accuracy of minimum width transistor area in estimating FPGA layout area. 2015, DOI: 10.1109/FCCM.2015.33.
5. C. Chiasson and V. Betz. "COFFE: Fully-Automated Transistor Sizing for FPGAs", IEEE International Conference on Field Programmable Technology (FPT), 2013
6. E. Lee, G. Lemieux and S. Mirabbasi. Interconnect driver design for long wires in field-programmable gate arrays. *Journal of Signal Processing Systems 51(1),* pp. 57-76. 2008.. DOI: 10.1007/s11265-007-0141-y.
7. J. Luu *et al*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS),* vol. 7, pp. 1-30, 2014.
8. S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE,* vol. 103, pp. 318-331, 2015.
9. J. Cong and B. Xiao, "FPGA-RPI: A Novel FPGA Architecture With RRAM-Based Programmable Interconnects," *IEEE Transactions on very Large Scale Integration (VLSI) Systems,* vol. 22, pp. 864-877, 2014.
10. Xilinx, “Virtex-5 user guide,” UG190 (v2.1), October 2006.
11. V. Betz and J. Rose, “How Much Logic Should Go in a FPGA Logic Block?” IEEE Design & Test Magazine, vol. 15, no. 1, pp. 10-15, Jan-Mar. 1998.

12. A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. A. El-Ayat, and A. Mohsen, “An architecture for electrically configurable gate arrays,” IEEE Journal of Solid-State Circuits, vol. 24, no. 2, pp. 394– 398, April 1989.
13. W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, “A user programmable reconfiguration gate array,” in Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 233– 235, May 1986.
14. Xilinx, “Xilinx 3000 series data sheet,” http://direct.xilinx.com/bvdocs/publications/3000.pdf
15. H.-C. Hsieh, W. S. Carter, J. Ja, E. Cheung, S. Schreifels, C. Erickson, P. Freidin, L. Tinkey, and R. Kanazawa, “Third-generation architecture boosts speed and density of field-programmable gate arrays,” in Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 2/1– 31.2/7, May 1990.
16. Altera Corporation, “Stratix II device handbook SII5V1-3.1,” http://www. altera.com/literature/hb/stx2/stratix2 handbook.pdf, July 2005.
17. A. Aggarwal and D. Lewis, “Routing architectures for hierarchical field programmable gate arrays,” in IEEE International Conference on Computer Design, pp. 475– 478, October 1994.
18. S. D. Brown, R. Francis, J. Rose, and Z. Vranesic, Field-Programmable Gate Arrays. Kluwer Academic Publishers, 1992.
19. W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, G. Varghese, J. Wawrzynek, and A. DeHon, “HSRA: High-speed, hierarchical synchronous reconfigurable array,” in Proceedings: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 125– 134, February 1999.

20. Altera Corporation, “FLEX 10K embedded programmable logic device family, DS-F10K-4.2,” http://www.altera.com/literature/ds/dsf10k.pdf, January 2003.
21. Altera Corporation, “APEX 20K programmable logic device family data sheet, DS-APEX20K-5.1,” http://www.altera.com/literature/ds/apex.pdf, March 2004.
22. Altera Corporation, “APEX II programmable logic device family, DSAPEXII-3.0,” http://www.altera.com/literature/ds/ds ap2.pdf, August 2002.
23. Altera Corporation, “Cyclone device handbook. C5V1-2.1, ver. C5V1-2.1,” http://www.altera.com/literature/hb/cyc/cyc c5v1.pdf, January 2007.
24. Lattice Semiconductor Corporation, “LatticeXP family data sheet, version 03.1,” http://www.latticesemi.com/lit/docs/datasheets/fpga/xp data sheet.pdf, September 2005.
25. Xilinx, “Virtex-4 family overview,” DS112(v1.4), http://direct.xilinx.com/ bvdocs/publications/ds112.pdf, June 2005.
26. N. H. E. Weste and D. Harris. CMOS VLSI Design Circuits and Systems Perspective. Pearson Addison-Wesley, 2005.
27. F. Rubin. The lee path connection algorithm. *IEEE Transactions on Computers C-23(9)*, pp. 907-914. 1974.. DOI: 10.1109/T-C.1974.224054.
28. F. F. Khan and A. Ye. An evaluation on the accuracy of the minimum width transistor area models in ranking the layout area of FPGA architectures. 2016, DOI: 10.1109/FPL.2016.7577325.
29. F. F. Khan and A. Ye. An evaluation on the accuracy of the minimum width transistor area models in ranking the layout area of FPGA architectures. 2016. DOI: 10.1109/FPL.2016.7577325.