AUTOMATIC LAYOUT OF FPGA TILES USING ONE-LAYER METAL CELLS

By

Simon So

Supervisor: Jonathan Rose

December 2003

AUTOMATIC LAYOUT OF FPGA TILES USING ONE-LAYER METAL CELLS

By

Simon So

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF APPLIED SCIENCE

DIVISION OF ENGINEERING SCIENCE

FACULTY OF APPLIED SCIENCE AND ENGINEERING
UNIVERSITY OF TORONTO

Supervisor: Jonathan Rose

December 2003

Abstract

Automatic Layout of FPGA Tiles Using One-Layer Metal Cells

Bachelor of Applied Science and Engineering, December 2003
Simon So
Division of Engineering Science
Faculty of Applied Science and Engineering
University of Toronto

The design and custom layout of FPGA (Field-Programmable Gate Array) can be a time consuming process.

The physical layout and routing of a modern commercial FPGA is currently done manually customize for each technology to ensure high performance. Automation of this process would greatly assist designer in development of FPGA. University of Toronto has undertaken groundbreaking work in developing a tool that automatically generates physical layouts from FPGA architectural specification.

This research focuses on improving the performance of the placement and routing tool. Using the latest techniques for cell grouping and one-layer metal cells, the area of Xilinx Virtex-E generated by GILES has improved by 135.3% compare to previous results.

Acknowledgements

I would like to thank my supervisor, Jonathan Rose, for the guidance he offered and encouragement that he provided throughout this project. His enthusiasm has provided motivation for me throughout this project.

I would also like to thank Aaron Egier and Ian Kuon for providing me with the resources I needed to conduct this research. Their assistance and support are greatly appreciated.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

There are many options to implement digital circuits. FPGA (Field-programmable gate array), standard-cell or ASIC (Application-Specific Integrated Circuits), and custom layout all offers possible solutions. FPGA offers ease and speed of development at cost of area, speed, power and price compare to other options. As technologies advances in the past two decades, transistors can be made smaller and faster, FPGA is becoming more and more popular for digital design. But the key reason that makes FPGA attractive is that they can be programmed to implement different digital circuits and be-reprogrammed if necessary. It allows designers to achieve lower non-recurring engineering cost and hence faster time-to-market for their design [1]. A well design FPGA can minimize the speed/area/power disadvantage and produced cheaply, thus making them available to a larger market.

Designing a high performance requires careful planning and can be a difficult task. Making a high performance FPGA lies on four major factors: the logic, routing, and the architecture of the FPGA, the circuit design in transistor level to realize the architecture, the software tool that configures it, and the physical layout [1]. Each factor is important and cannot be overlooked. As the architecture determines an FPGA capability, the transistor level design determines the circuit speed performance, the software tool provides the usage for designer and the physical layout impacts the area and cost.

Furthermore, all these factors are highly interrelated. For example, the circuit design will impact the physical layout. And because the four factors are interrelated, one motivation is to automate the whole process flow.

An automated FPGA architecture evaluation tool that considers circuits in transistor level given an architecture description can greatly reduce design efforts. Moreover, if the tool can evaluate the physical layout can reduce time and cost. An ideal tool would be able to determine the circuit from transistor level to physical layout by specifying an architecture description.

In the past decade, University of Toronto has made effort in trying to developing a tool that links three of the four factors indicated above; a CAD tool that can automates the exploration and evaluation of FPGA architectures [2]. GILES, Good Instant Layout of Erasable Semiconductors, is the name of the tool developed.

GILES is based on VPR [4]. It is capable of generating circuit design and the physical layout from an architecture description. Though GILES is not yet close to the quality achieved by humans, the benefits of the automated circuit and layout generation are: reduction in time and initial design effort. Moreover, these designs and layout can serve as starting points, thus reducing the cost and time spend on FPGA design cycle [2]. In order for the realization of these benefits relies on the tool to consider all layout constrains and generate adequate-quality layout based on the architecture description.

The focus of this research is on the physical layout of the part of GILES. The goal is to move closer to area achieved by human. Since the key structure of an FPGA is replicated over and over like tiles, much effort has been devoted to hand-tune a high performance structure that provides good speed in a small area. Aside from University of

Toronto, there have been other similar research projects such as AKORD and commercial tools Synopsys' Cadabra that aid designer in automatic transistor level layout [3]. However, they are only capable of laying out hundreds of transistors, not tens of thousands in an FPGA tile.

In order to evaluate the quality of GILES generated tiles, a commercial architecture Xilinx Virtex-E is chosen as a comparison in this research. However, this comparison is only an approximation because it is not possible to accurately describe the commercial architecture with the relatively simple GILES architecture description [3]. This research is an attempt to move closer to goals of considering physical layout by using a more accurate cell-level netlist; and hence able to give a better description of the commercial architecture and area evaluation.

1.2 Scope

This research involves two phases, In the first phase, we layout actual cells that is going to be used for building Xilinx Virtex-E.

In the second phase of our work, we generate the netlist for the cells and feed it into GILES to perform layout. The scope of this work is limited to area comparison with actual cells. Optimization and transistor layout for GILES is left to future work.

1.3 Thesis Organization

Chapter 2 presents background information and detail about previous work that is relevant to this research. Chapter 3 describes the first phase of our work, difficulties encountered while laying out the cells and solution to the problem. Chapter 4 compares

the results in area after layout. Chapter 5 summarizes the conclusion and gives suggestion for future work.

Chapter 2

Background

This chapter is divided into four sections. The first section of this chapter illustrates the structure of FPGAs. The second section describes the architecture, tiles, and routing of FPGA. The third section will describe GILES – Good Instant Layout of Erasable Semiconductors and its previous work. The fourth and final sections outline GILES performance in creating a commercial product and comparison of its performance to a commercial tool.

2.1 Overview of FPGAs

An FPGA can be configured to perform a wide variety of digital logic. FPGAs are mainly made up of three components: logic blocks, programmable routing, and I/O pads [1]. The logic block and programmable routing is what makes up the core and I/O pads are on the perimeters of the FPGA.

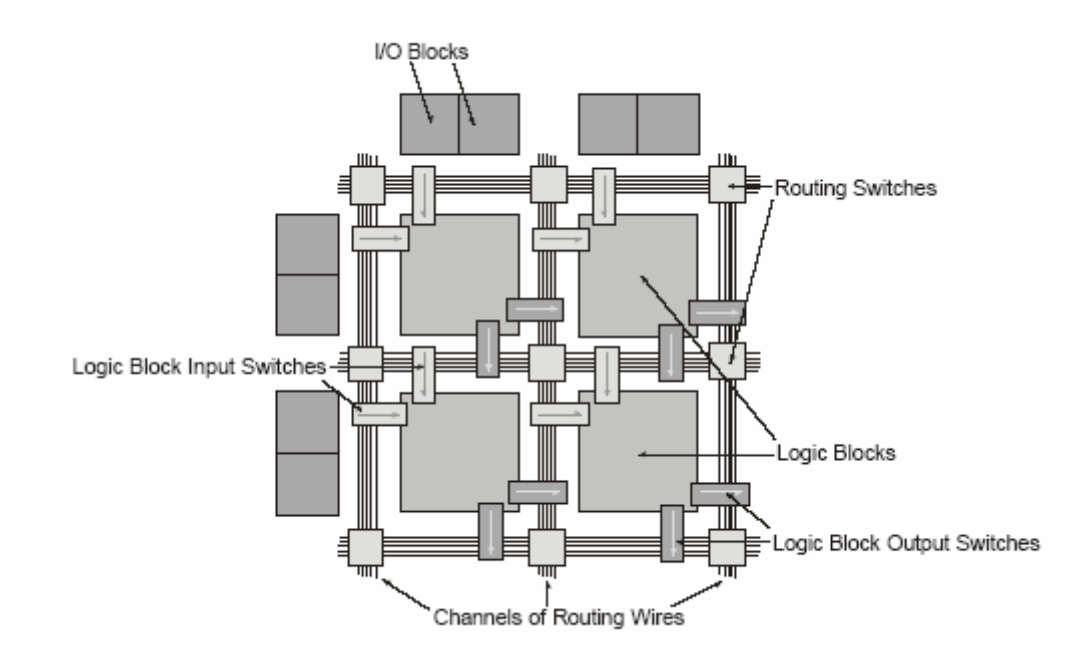


Figure 2.1 Block diagram of a FPGA [2]

The core of the FPGA is created by tiling the same logic block and its surrounding route. Figure 2.1 shows the routing channels that run between logic blocks. Between the channels are switches. These switches controls routing paths and logic of the logic blocks. These switches can simply a pass transistors or a pass transistor with a buffer for greater drive strength. The gate of the pass transistor is control by an SRAM cell. Figure 2.2 shows the schematic of a switch.

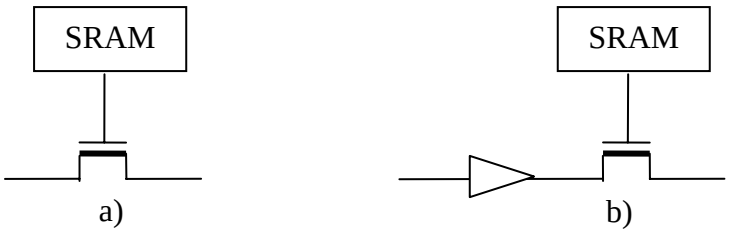
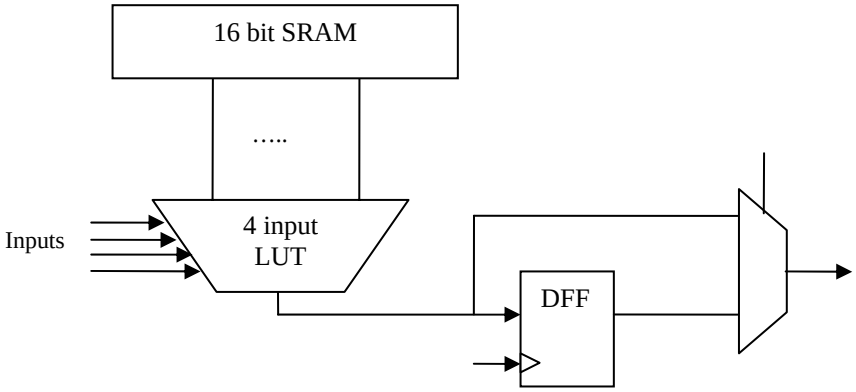
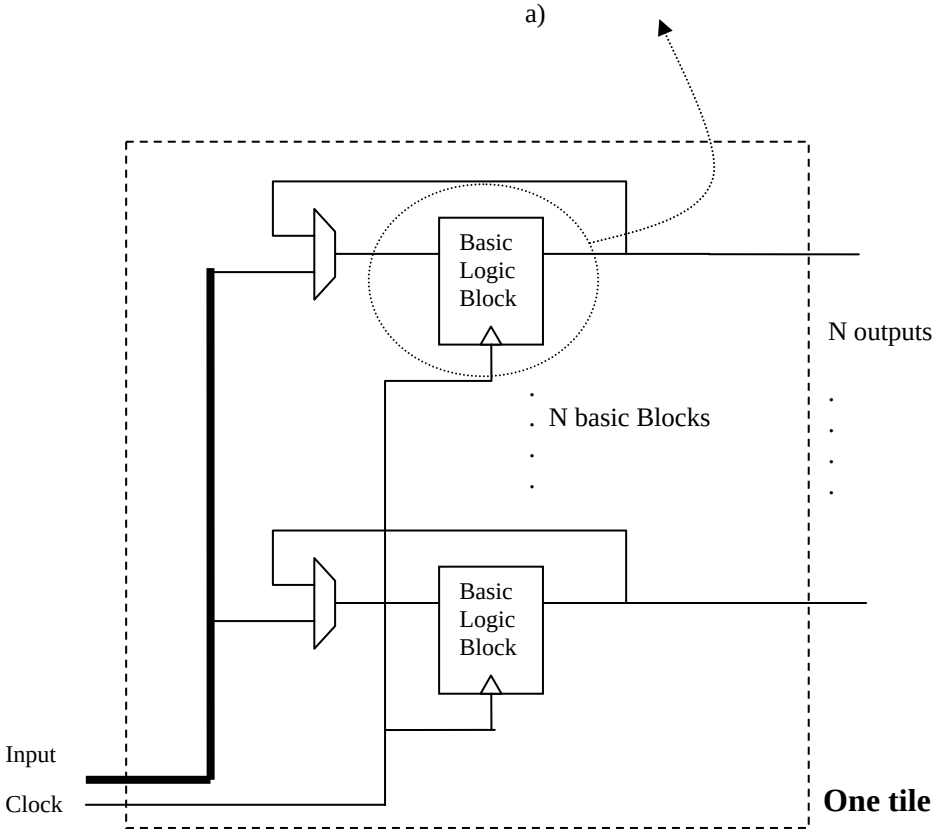


Figure 2.2 Two types of routing switches a) pass transistor b) pass transistor with buffer

The logic block consist of a 4-input Look-Up Table (LUT), D-Flipflop, and Multiplexers. Figure 2.3 shows the block diagram of the logic block.



a)



b)

Figure 2.3 a) One Basic Logic block, b) Block diagram of one tile, it may consist of multiple Basic Logic block so it can have multiple inputs and outputs

The LUT is a 16-input MUX where the input of the LUT is driven by a 16 bit SRAM block. The select of LUT are driven by inputs from other logic blocks or feedback from the output. And thus, by programming the SRAM bits, it is possible to realize any 4-input Boolean logic function, also the routing of the output can be configured so that more complex logic can be implemented efficiently.

2.2 FPGA Tiles

As mentioned earlier, the core of a FPGA is created by replicating tiles. A well designed and layout can yield a high speed and power performance FPGA. With a well planned of a single tile, it can be duplicated regularly. Because the fundamental building block of the FPGA core is a tile, thus the smaller each tile is will translate to a smaller chip area; and hence lower fabrication cost, or more logic per chip. Inside each tile, it may contain one or more logic block. The automatic transistor layout studied in this research the layout of one tile.

2.3 GILES

GILES is a system that produce layout of FPGA tile from a FPGA architecture description file [3]. GILES can perform two operations: area estimation and cell placement. For area estimation, GILES uses information about the transistor sizes from the circuit. If only the layout information is given, the system can place and route the cell; however, this requires manual pre-layout cells such as buffer, multiplexers and etc.

GILES processes the architecture description and eventually generates a layout view of the tile is performed by three basic steps: netlist generation, cell placement and inter-cell routing. Figure 2.5 describes the process flow.

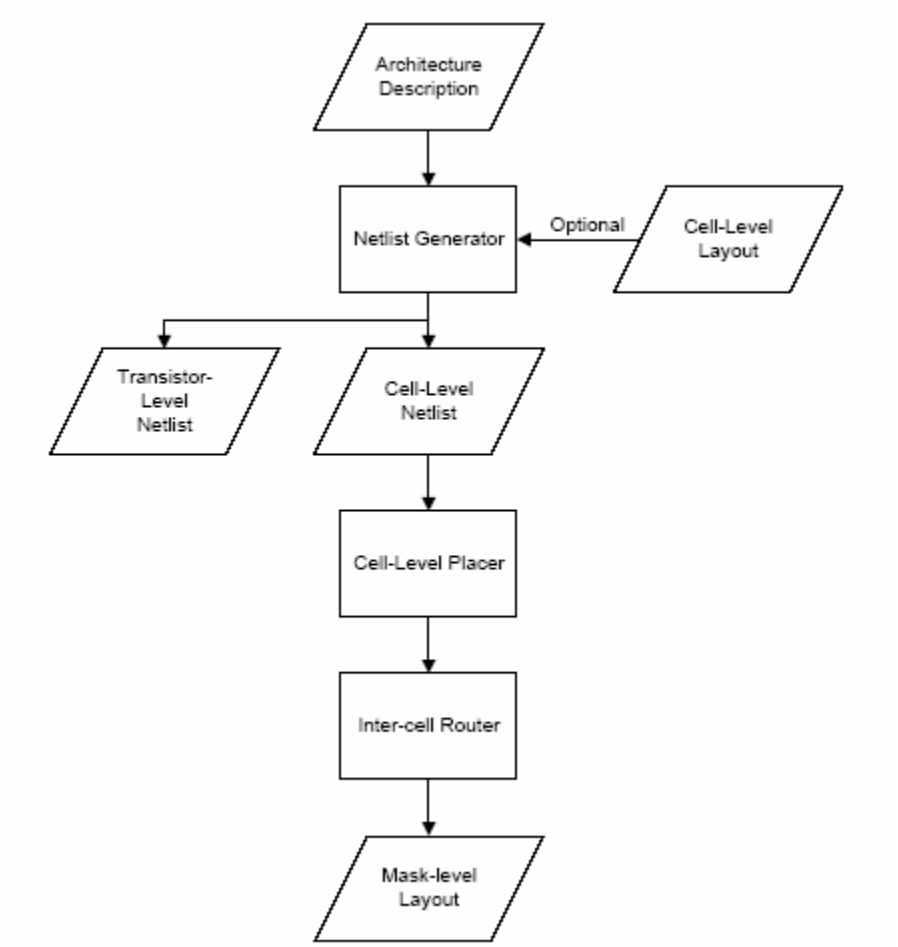


Figure 2.4 GILES CAD flow [3]

The netlist generation is done using an enhanced version of VPR – Versital Place and Route designed at the University of Toronto [4]. VPR performs clustering, placement, and routing of circuit netlist using the architecture description. Figure 2.6 shows a simplify CAD flow of VPR.

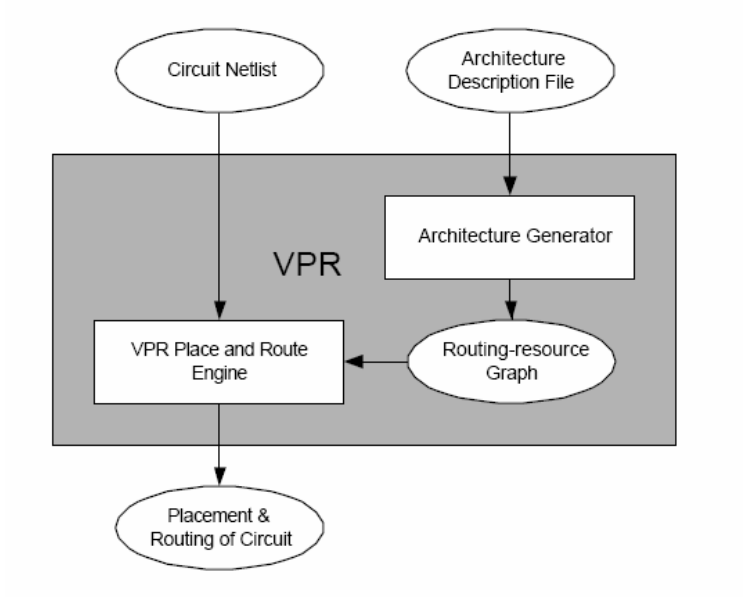


Figure 2.5 Simplied VPR CAD flow [2]

2.3.1 GILES Cell Placement

The GILES placer performs placement and compaction simultaneously [3]. The placement involves placement of cells and ports to from a rectangular FPGA tile. Ports are arranged on the perimeter. All ports must place symmetrically across left to right or top to bottom with its partner as shown in Figure 2.8. And hence, it ensures adjacent cell can connect to the ports.

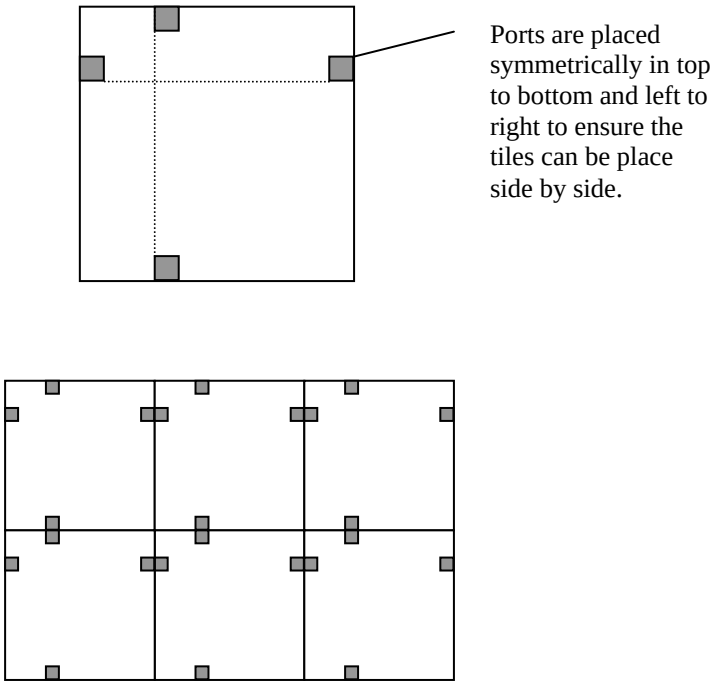


Figure 2.6 Port constrain in a tile

The GILES placer operates by continually making small changes to the layout. Illegal layouts are not considered by the placer, for instance, overlapping of cells is not allowed. Through many iterations, these small incremental changes will gradually decrease the size of the total area, improves cell and port placement.

2.3.2 Area Approximation

The area of a cell is calculated in units of squares, where each square is 0.66 µm x 0.66 µm. 0.66 µm is the minimum distance for two adjacent wires to run side by side. The idea of the grid is so that forces the router to only place wire within the grid. A consequence of that is when cells are being placed; they need to be placed in the 0.66 µm x 0.66 µm grid as well as their ports line up with the grid. That way, the router can route cells without knowing the actual size of the wire, but only any arbitrary unit of squares.

This is a more flexible for future technology changes. Figure 2.10 illustrates the cell placement and wiring in a grid.

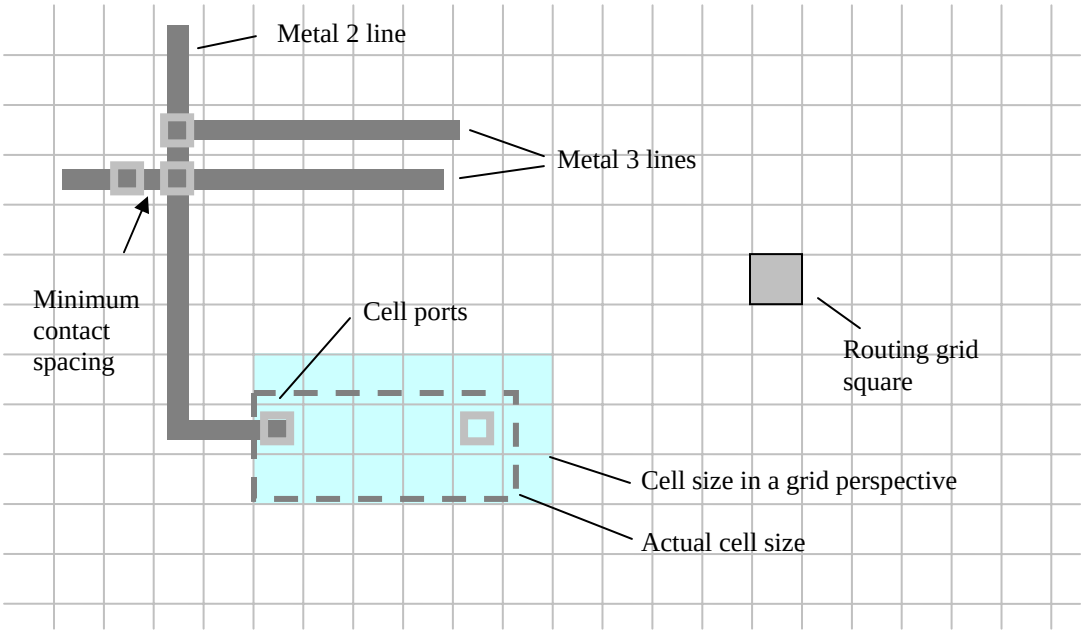


Figure 2.7 Cells placed in a routing grid

To initiate the placement process, the placer first calculates the initial tile dimension based on the total cell area.

$$Area = 3.3 * Complexity * Sum\ of\ all\ gates(0.5 * DriveStrength\ of\ gate + 0.5) \qquad (1)$$

The number 3.3 is used to include area needed the minimum distance between 2 minimum-width transistors. Note that equation (1) is independent of number of pins. The Complexity factor is used to accommodate the difference between types of cells. From empirical results, the equation can estimates the area for simple structure such as inverters, buffers, pass transistors, and flip-flops [3]. However, complex cells such as SRAMs and multiplexers are underestimated since the diffusion region sharing is more

complex to model. Hence, the complexity factor for SRAMs and multiplexer is found to be 1.455 and 1.0 for the other cells.

The width and height of the cells are chosen to make the cells as square as possible. For each cell, one grid square border around the cell. This is to account for the spacing between n-wells in design rule since the height and width of the cell does not account for the n-well. This way, two cells can be placed side by side without any violating design rules.

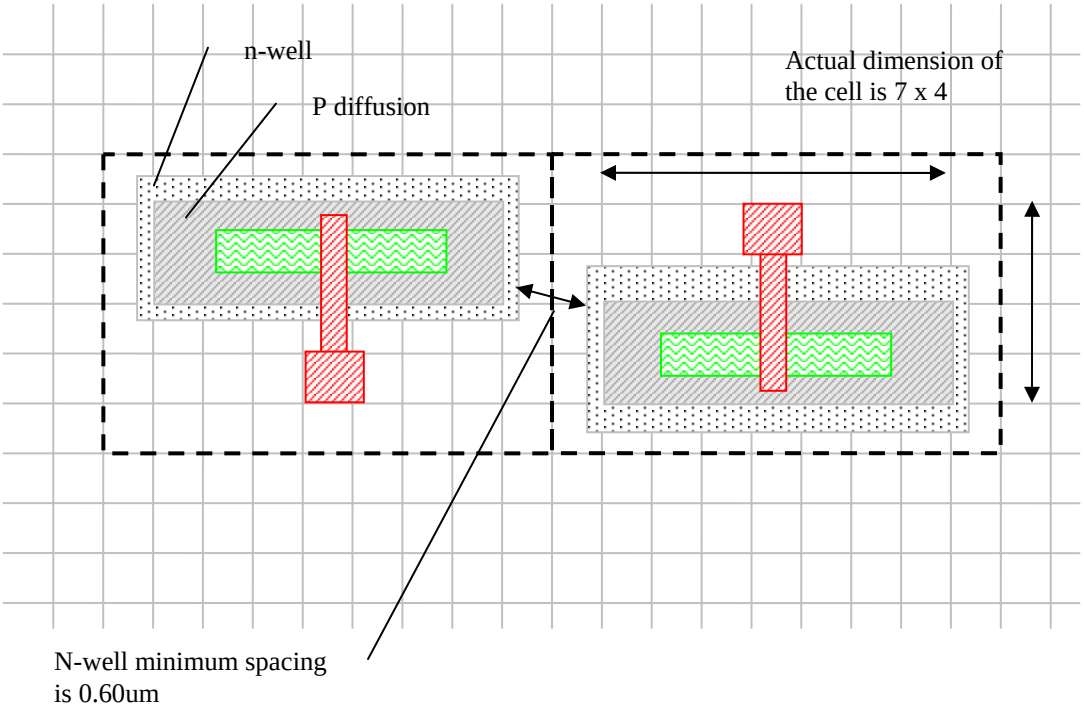


Figure 2.8 N-well does not line with adjacent cell, so 2 grid spacing is require to ensure minimum distance is satisfied

2.3.3 Place and Route

Once the initial area is determined, the cells are placed randomly into the box and the ports are placed on the perimeter. The cells are placed such that no overlapping is

allowed. In the first iteration, the cells are placed from largest to smallest [2]. The heuristic helps the initial placement because smaller cells can often fit in the gaps between the larger cells. The cells are then slowly compacted by a process called Annealing. After the cells are placed, the router routes the signal with a given number of layer of metal. If routing is unsuccessful, GILES will increase the size of the box and tries to route until success. GILES also reserve one layer for global signals such as VDD, GND and clock.

2.3.4 Grouping

As mentioned earlier, 1 grid borders are added around each cell to avoid design rule violation. For large cells such as LUT, the increase in area may not be significant; the impact of this border becomes dramatic for small cells such as pass transistor and inverters. The table below shows the increase in cell area when a border is added. Small cell such as an inverter area has increased by 150%.

Cell	**With border**			**With border**			**% increase**
	Width	Height	Area	Width	Height	Area	
1x Inverter	3	4	12	5	6	30	150.0%
SRAM	5	5	25	7	7	49	96.0%
LUT	13	12	156	15	14	210	34.6%

Table 2.1 Area comparison to show the border effect

The border is merely white spaces that serve no purpose but avoid n-well budding. As a result, it is wise to group certain types of cells together to minimize this inefficient use of space. By grouping certain cells together, n-well for the cells can be share. For instance, in Figure 2.9, the black is the actual cell area and gray is the border. Two separate cells combined occupied 60 grid squares, while the actual area of the two cells is

24 grid squrares [3]; and so 60% of the area is wasted. If the cells were merge together as one, say a 5 by 5 square, the total area now becomes 49 grid squares, a saving of 18%.

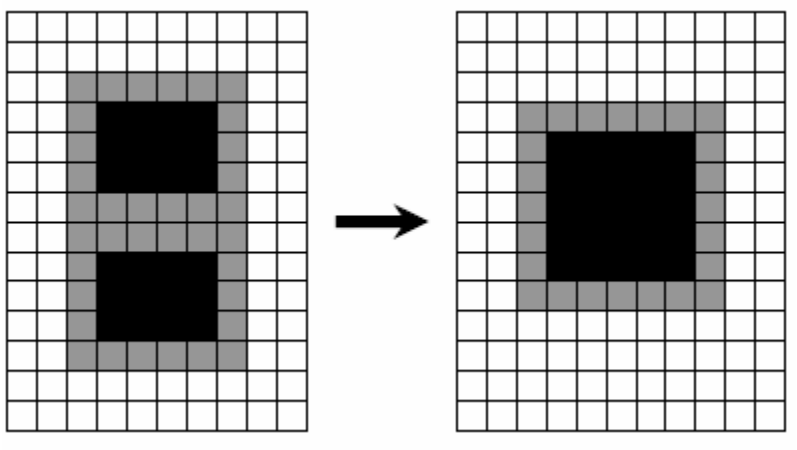


Figure 2.9 Grouping cells saves border area [3]

One of the tradeoffs in combining cells is that the combined cells are larger and limits the amount of freedom to place and compact. Secondly, bigger cells require more manual layout effort. An effective grouping is to group cells that are used more frequently [3]. Previous research shows that flip-flops and LUTs account only 0.3% each where SRAM account for 36% of the cells. Buffers and pass transistors are also common since they serve the foundation of routing. Combining SRAM with pass transistor and buffer would decrease the effective area significantly.

2.4 GILES and Xilinx Virtex-E

Xilinx Virtex-E is manufactured in a six metal layer 0.18um process. The comparison was done by cells with two layer intra-cell connections, three layers for inter-cell connection and one of which is reserved for global structures such as power, ground and clock.

The result GILES achieved, with buffer, pass transistor and SRAM bit grouping, was 101 752 μm^2. The area generated by GILES is 187% larger than the full custom design. Table 2.2 shows a summary of the result.

	Routed area by GILES
Without cell grouping	106 062 μm^2
4x4 SRAM and grouped buffer and pass transistor	161 444 μm^2
Buffer, pass transistor and SRAM bit grouping alone	101 752 μm^2

Table 2.2 Summary of area by GILES at different runs

Though the groupings improves the routed area by reducing the wire length, and hence eliminates some of the congestions, the area is still almost three times larger.

2.5 Standard Cell

Standard cells are common for ASIC implementation. They are called standard cells because they have a specific height and other specifications such as power rails on top and ground rail at the bottom. That way, designer can place standard cells side by side during placement, which is much easier laying out custom cells. To compare the placement and routing capability of GILES, standard cells are used to create Xilinx Virtex-E and placed and routed with Cadence's Design Planner and Silicon Ensemble automatic layout tools. Placement is performed using QPlace and WRoute is used for global and detail routing [3]. The result from GILES and Cadence Design tools is summarized in Table 2.3.

	GILES generated tile area (μm^2)	Standard Cell tile area (μm^2)	% Difference
Final Routed Area	101 752	71 569	-30%
Cell Area	35 836	61 000	70%
Wirelength	274 069	166 971	-39%

Table 2.3 Standard Cell comparison

The result shows that even though the Standard Cell area was bigger initially, the commercial tool was able to route larger standard cell more efficiently [3]. One major factor is due to Standard Cell used only one layer of metal, and thus leaving one more layer of metal for inter-cell routing. Also, the Cadence router is intelligent enough to use not have to reserve one layer for global signals such as VDD, GND and clocks.

Chapter 3

Cell Layout

3.1 Goal

The goal of the research is to improve GILES in placement and routing in minimizing the area of laid out tiles. As mentioned earlier, routing in FPGA is actually the bottleneck in minimizing area for GILES. It is for this reason, we are motivated to create actual cells with one layer metal for intra-cell and reserve an extra layer for routing purpose. This chapter is divided into two sections. First, it describes the problem encounter in laying out cells with 1 metal layer and how it affects cell area. The second describes the final routed area and comparison with the previous place and routed area.

3.2 One layer metal layout Result

Laying out cells with only one layer metal resource can be a difficult and time consuming task. Reason being that routing becomes challenging. The placements of various components are critical to avoid any crisscrossing of metals. Figure 3.1 shows a schematic of a SRAM layout with one metal layer. The poly of the feedback transistor is extended and wrapped around to avoid traffic in the middle. In some cases, this wrapped around has causes huge area inflation. The following subsection will describe three methods in overcoming these inflations.

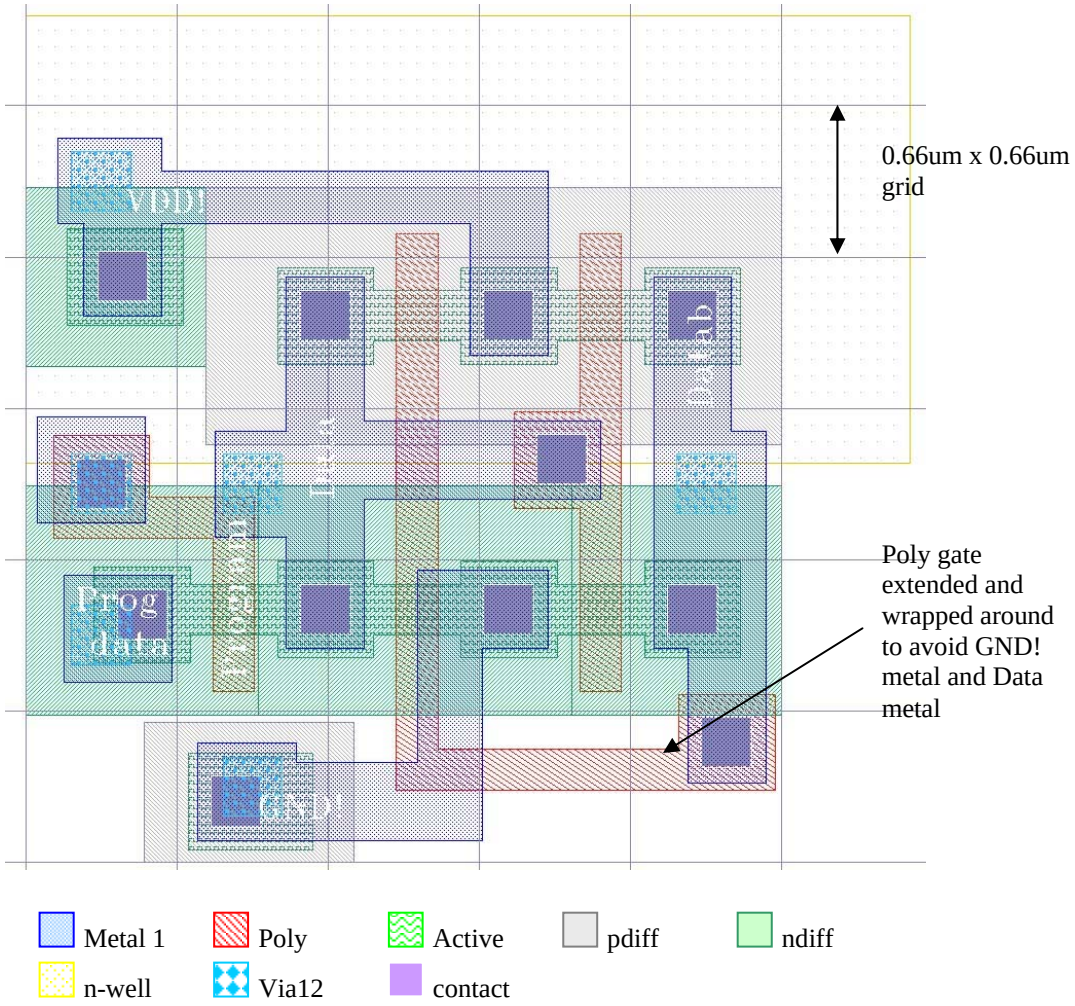


Figure 3.1 SRAM cell with 1 layer of metal

3.2.1 Polysilicon routing

In some cases, it is unavoidable to have wires crisscross each other. One method used in resolving this problem is by using polysilicon to route. As shown in the Figure 3.1, the polysilicon is used route to the Datab metal to avoid the GND metal and the Data metal. The downside of that is speed since polysilicon tends to have a high resistance.

3.2.2 Multiple Vdd and Gnd contact

Wrapping wires around can be used to avoid traffic, however, a disadvantage of that is it causes area inflation. In some of the bigger cell, area increase may be severe. For instance, 6x Buffer shown in Figure 3.2 has a dimension of 5 x 10 squares. And to save space, the N-well is shared by four pmos. As a consequence, the four nmos are splited resulting two separate GND metals. If a metal was to stretch and connect the two wire GND pin together, the cell would require an extra column of grid. This results an increase in area by 20%.

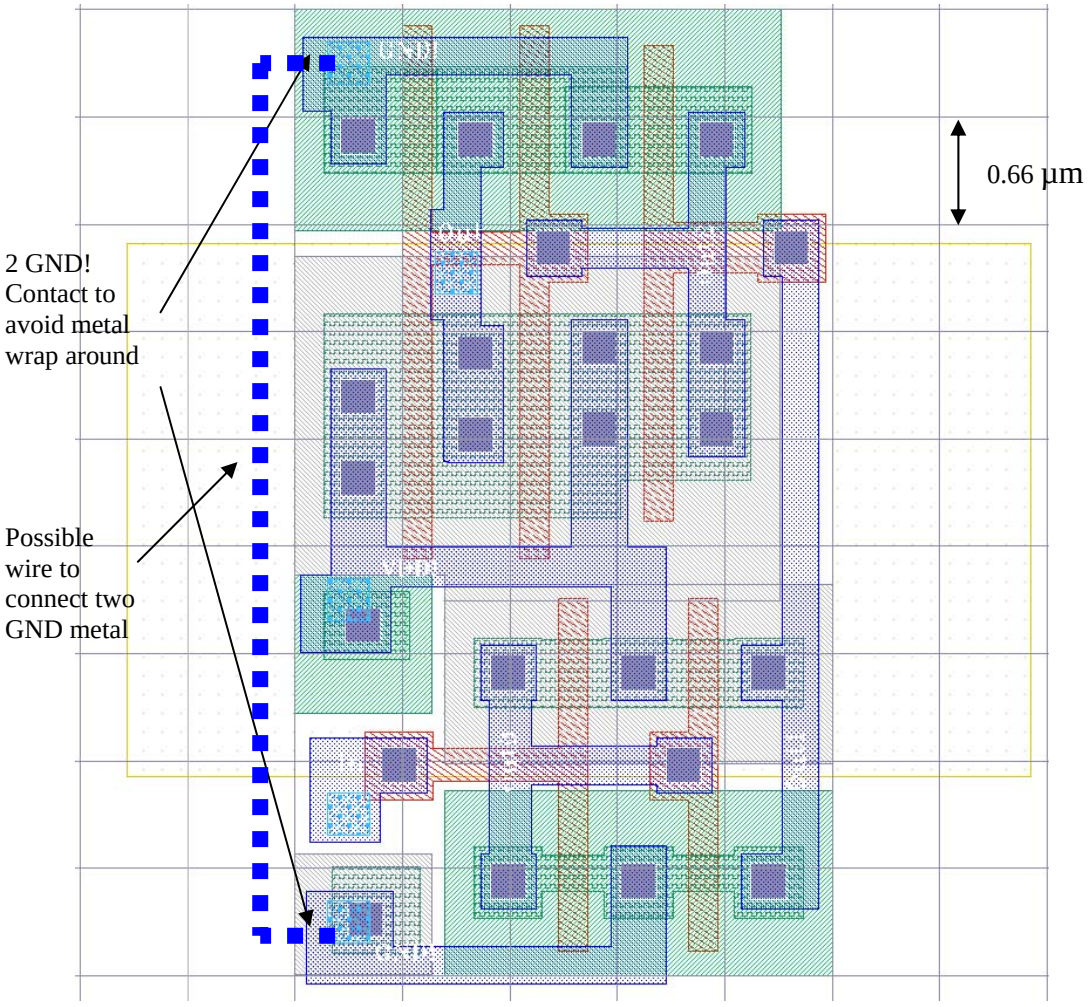


Figure 3.2 6x Buffer

To resolve such this problem without sacrificing unnecessary area is by having multiple GND pins. Because GND resource is routed in Metal 2 and it is available in many locations, and thus having multiple GND pins would not cause routing problem. Besides GND pins, multiple VDD pins are placed as well to avoid unnecessary area inflation. Refer to Appendix # for example of multiple VDD pins.

3.2.3 Cell Grouping

To show that grouping indeed saves cell area as mentioned previously, a pass transistor, a buffer, and an SRAM is grouped together for area comparison. The sizes each component is list below.

Cell	**Without Border**			**With Border**		
	Width	Height	Area	Width	Height	Area
6x Buffer	5	10	50	7	12	84
SRAM	5	5	25	7	7	49
Pass transistor 6.25x	5	3	15	7	5	35
Total area			90			168

Table 3.1 Cell area

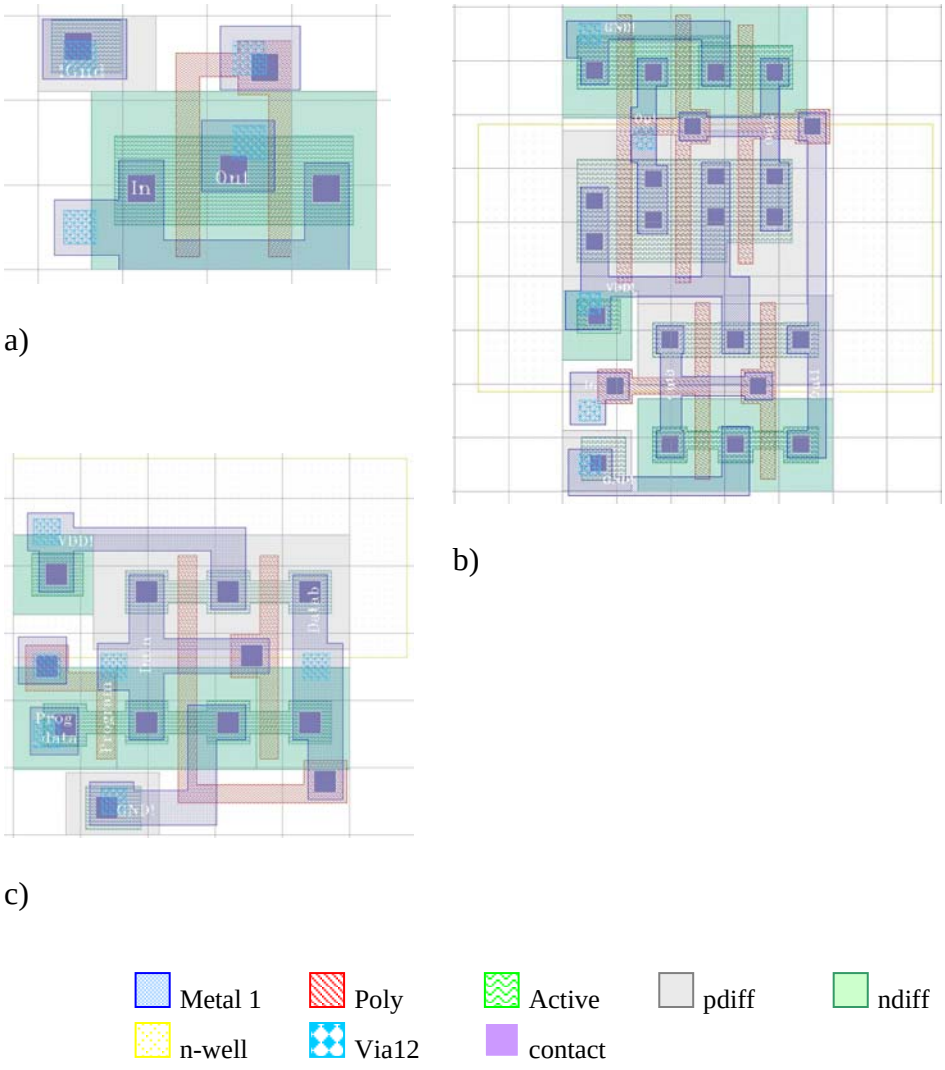


Figure 3.1 Layout of individual cells a) Pass transistor 6.25x, b) Buffer 6x, c) SRAM

The area after adding border has almost doubled. If the cells were combined together, making it as square as possible, the final area obtained is 99 without border, but only 143 with border. Although there is a slight increase in the raw area, this is due to the fact that the cells require some minimum spacing in between. However, the final area is 25 squares smaller, 14.9% decreases, since the border spacing overestimates the amount of spacing required between the cells.

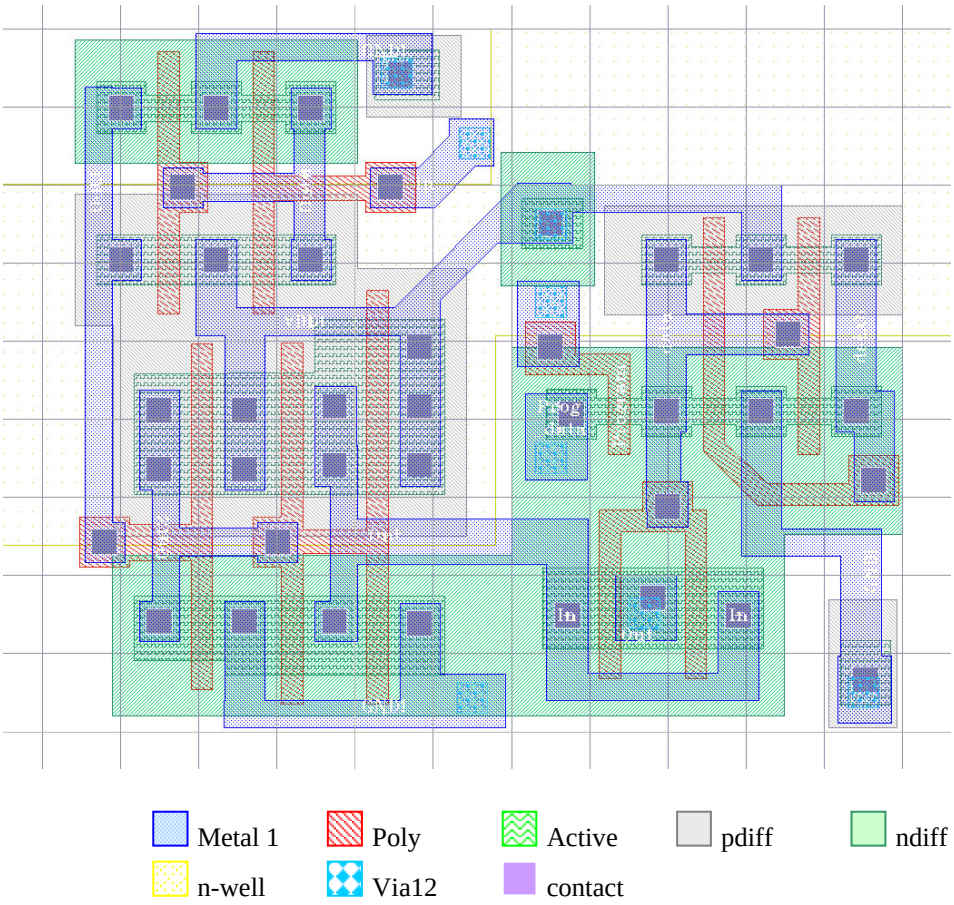


Figure 3.2 SRAM-pass transistor 6.25x-Buffer 6x switch

Chapter 4

Results

4.1 Summary

One layer metal cells was predicted larger than using two layer metal cells. Table 4.1 shows the sizes for the list of cells use for comparison. The list consist of various sizes of inverters, MUX and Pass transistors, and LUT to obtain an average inflation.

Cell	**2 layer**			**1 layer**			**% increase**
	Width	Height	Area	Width	Height	Area	
1x Inverter	3	4	12	4	4	16	33.3%
2x Inverter	3	4	12	4	5	20	66.7%
4x Inverter	4	4	16	5	5	25	56.3%
4x Buffer	5	6	30	6	5	30	0.0%
2 Input MUX	3	3	9	3	4	12	33.3%
12 Input MUX	11	10	110	13	9	117	6.4%
24 Input MUX	17	16	272	23	14	322	18.4%
LUT	13	12	156	17	11	187	19.9%
SRAM	5	5	25	5	5	25	0.0%
Pass transistor 3x	3	2	6	4	3	12	100.0%
Pass transistor 8x	4	3	12	4	3	12	0.0%
Average							30.4%

Table 4.1 Area Comparison between 2 layer and 1 layer without border

Cell	**2 layer**			**1 layer**			**% increase**
	Width	Height	Area	Width	Height	Area	
1x Inverter	5	6	30	6	6	36	20.0%
2x Inverter	5	6	30	6	7	42	40.0%
4x Inverter	6	6	36	7	7	42	36.1%
4x Buffer	7	8	56	8	7	56	0.0%
2 Input MUX	5	5	25	5	6	30	20.0%
12 Input MUX	13	12	156	15	11	165	5.8%
24 Input MUX	19	18	342	25	16	400	17.0
LUT	15	14	210	19	13	247	17.6%
SRAM	7	7	49	7	7	49	0.0%
Pass transistor 3x	5	4	20	6	5	30	50.0%
Pass transistor 8x	5	6	30	6	5	30	0.0%
Average							18.8%

Table 4.2 Area Comparison between 2 layer and 1 layer with border

The Xilinx Virtex-E is placed and routed with the one layer metal cells. The final area, with the best grouping (SRAM-pass transistor-buffer), is 53 807 μm^2. Compare to the Virtex-E full custom tile with an area of 35462 μm^2 [3], the area obtain by GILES is 51.7% larger. And compare to the standard cell, GILES performed 33.0% better.

Step	**Grid size**	**Area (μm^2)**
Placement	292 X 312	39 785
Post-Routing	346 X 357	53 807
% difference		35.5%

Table 4.3 GILES results for placement and final routing for 6 metal layer (4 layer for inter-cell routing)

Step	**Grid size**	**Area (μm^2)**
Placement	292 X 312	39 785
Post-Routing	504 X 540	126 857
% difference		218.8%

Table 4.4 GILES results for placement and final routing for 5 metal layer (3 layer for inter-cell routing)

As the result has shown, the extra layer of metal was critical in minimizing the final area.

Chapter 5
Conclusion

5.1 Final result

The goal of the research is to improve GILES in placement and routing in minimizing the area of laid out tiles. This research has shown a significant improvement in the GILES' performance.

Using the Xilinx Virtex-E architecture as a comparison, by making the cells larger on average of 18.8%, but reserving an extra layer of metal for GILES to routing, the area has improved by 135.3%. The technique used to achieve this result is through most frequent functional cell grouping and one layer metal intra-cell. It is shown that with one layer cell, GILES is capable to place and route the tile for 33.0% smaller compare using Standard Cells. Though, compare the full custom layout Virtex-E, the area is still 51.7% larger.

There are several reason contribute to why manual layout is still superior than automatic layout. As discussed previously, the overestimation of border spaces to avoid design rule violation. But by grouping cells together minimizes the cell area. But there downside of bigger cells that is they are more difficult to layout. And second, they are harder to place due to lack of freedom. Minimizing the border spaces to compact the cell is left for future work and analysis.

5.2 Future Work

Though in this research, it was successful in minimizing the tile area, to be able to compete with full custom manual layout, much study and analysis is needed to improve the GILES placer and router. The outcome of this research yield several paths for further investigation, some of which will be discuss below.

GILES perform place and routing in two separate steps as mentioned. During placement, the placer is only concern in placing the cells as compact as possible. Afterward, the route take the initial area and tries to make possible routing. If routing is unsuccessful, it expands the area until routing is possible. Future work can examine re-placement of certain cells to make routing possible. An investigation of the cost function can include the possibility of a successful route. Besides predicting a failure, the function can predict how close it is with the current configuration to achieve a solution. The placer can also be investigated in finding out where the signals of cells connected to, and so during placement optimization, cells that are connected together will be place closer together. Hence routing can be more efficient and avoid congestion.

The current routing reserved a designated layer for VDD, GND and clocks. With the commercial router, it knows where the global signals are and so it manages to use that layer as a routing resource. Future work can investigate on the possibility of using the global reserved layer for routing.

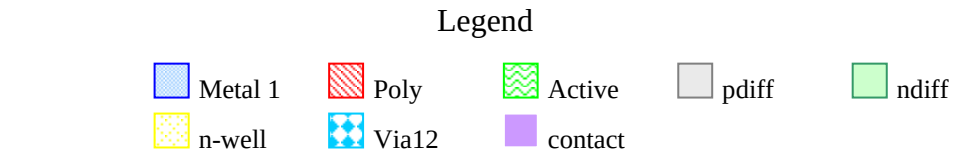
Another possible future work can minimize the amount of white space in each layer. The current GILES requires all layers to line up along a 0.66um x 0.66um grid. This is a disadvantage for cells that are partial occupying an incomplete grid. Moreover, metal layers that do not require this much room between them still needs to obey the

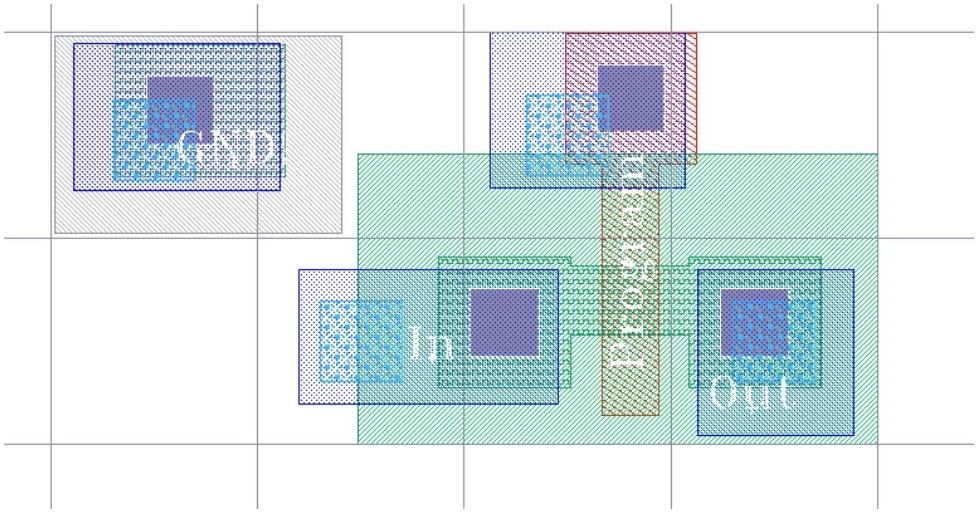
0.66um x 0.66um grid rule. For instance, metal 1 only need 0.23um space and minimum width is 0.23um, and so the minimum grid size for this can be 0.46um. This would allow more space for routing. Also, the border of cell is used by GILES is overly conservative. As shown previously example, by compacting cells together, an addition of 14.9% space can be saved. Knowing where the n-wells are and appropriately adding border only at where necessary can further reduce the size of the tile. This also lead to the investigation of sharing n-well, n-well contact and substrate contact among cells.

Appendix A

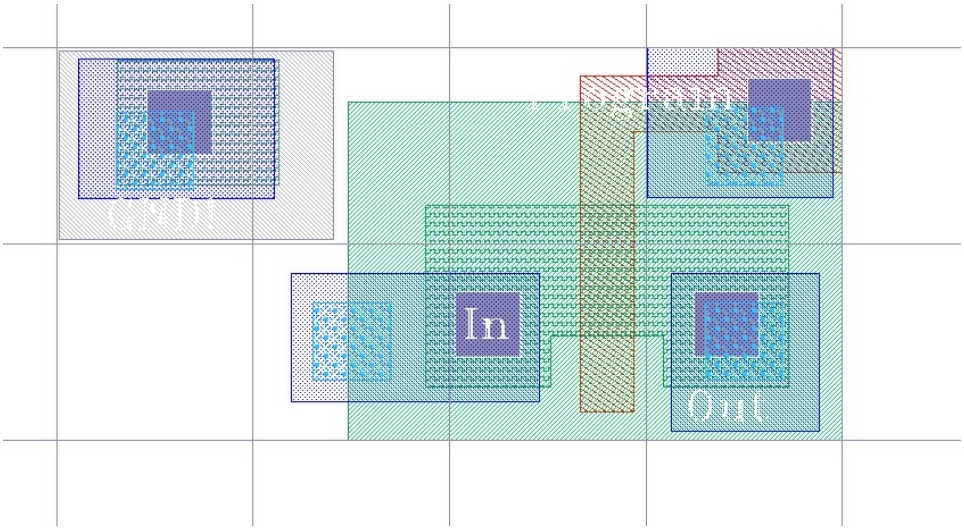
Layout of the cells

Appendix A shows the layout of all the cells used for GILES for Virtex-E.

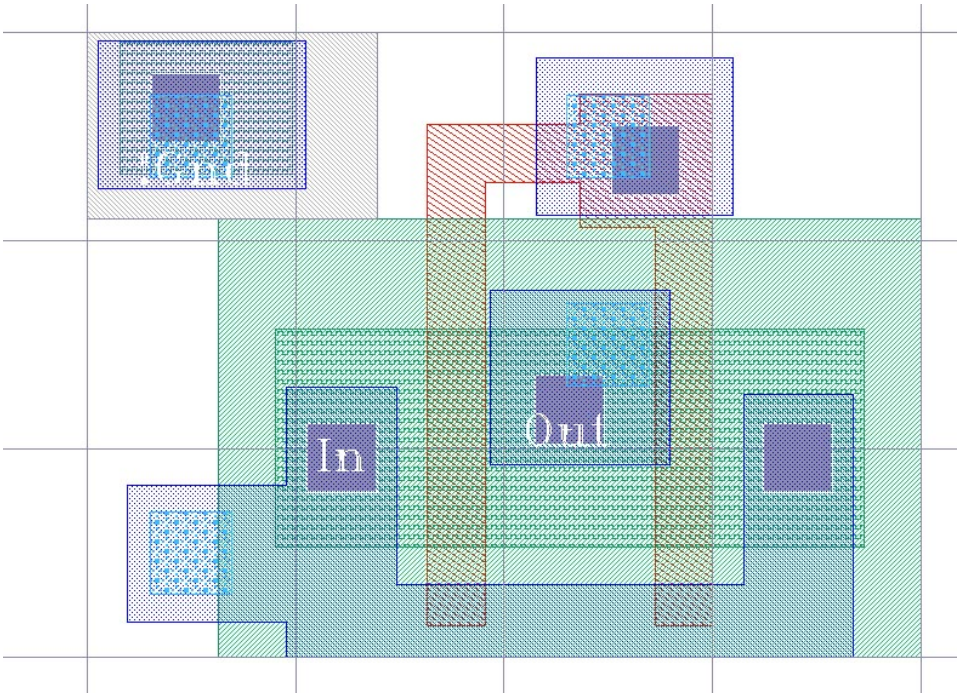




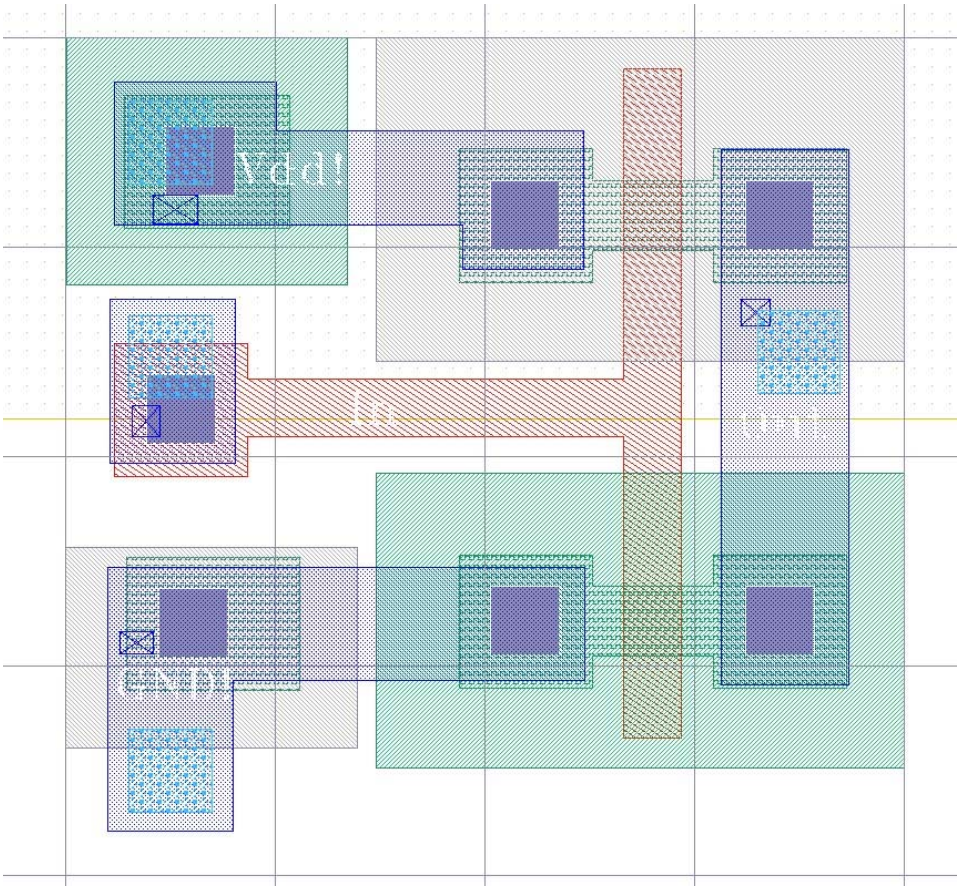
Pass transistor 1x



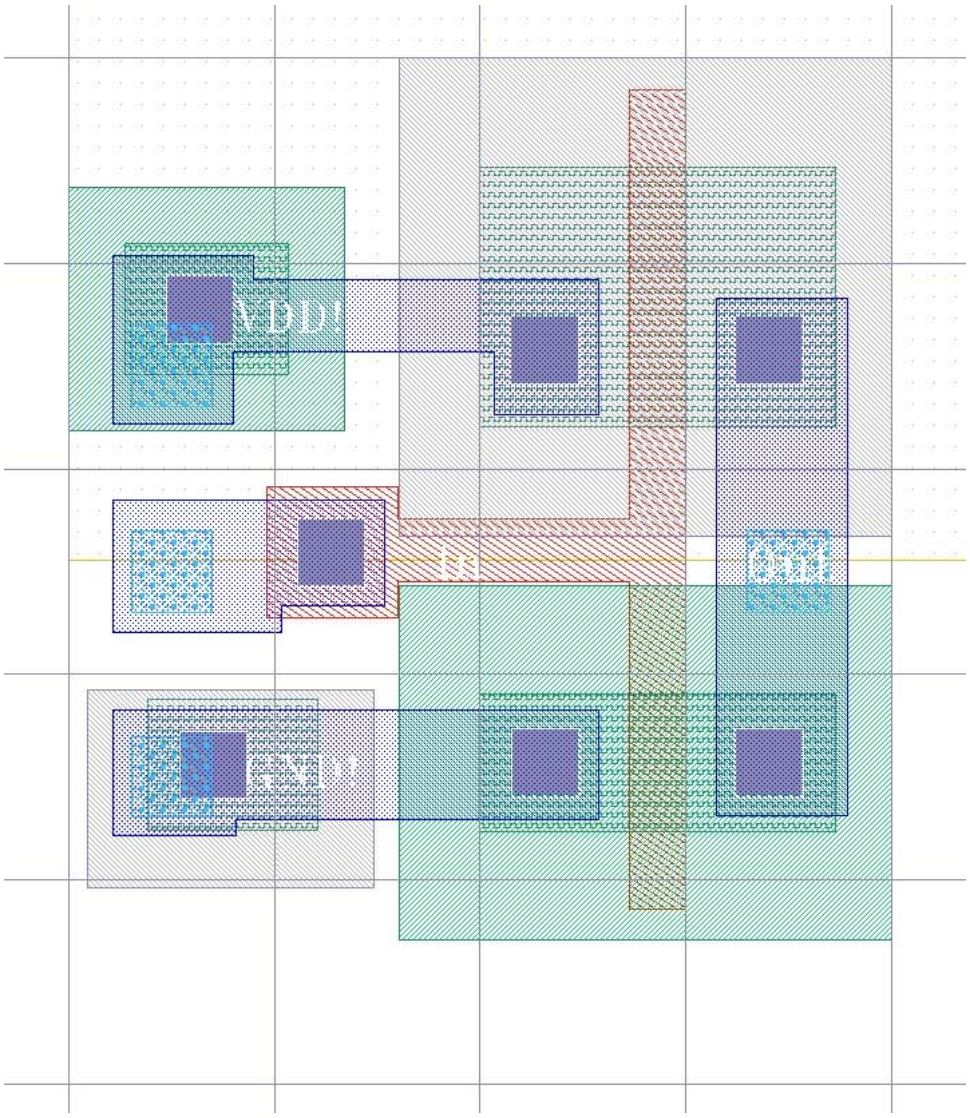
Pass transistor 2x



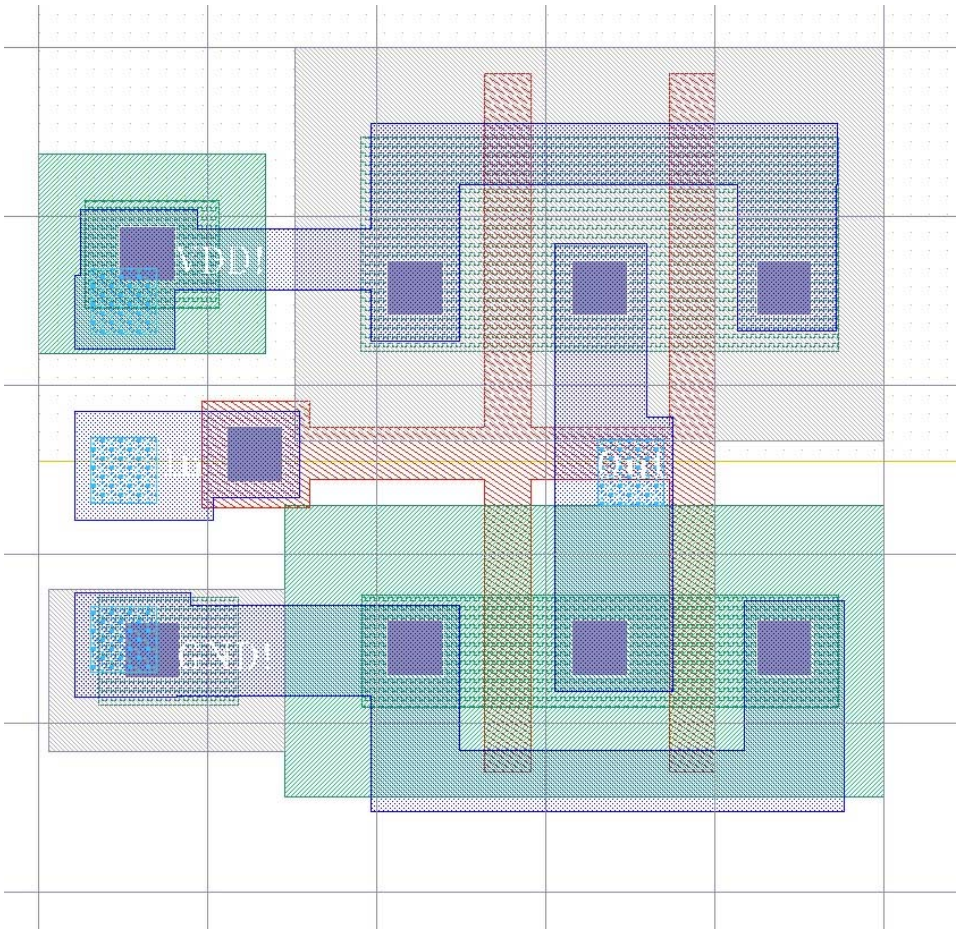
Pass transistor 6.25x



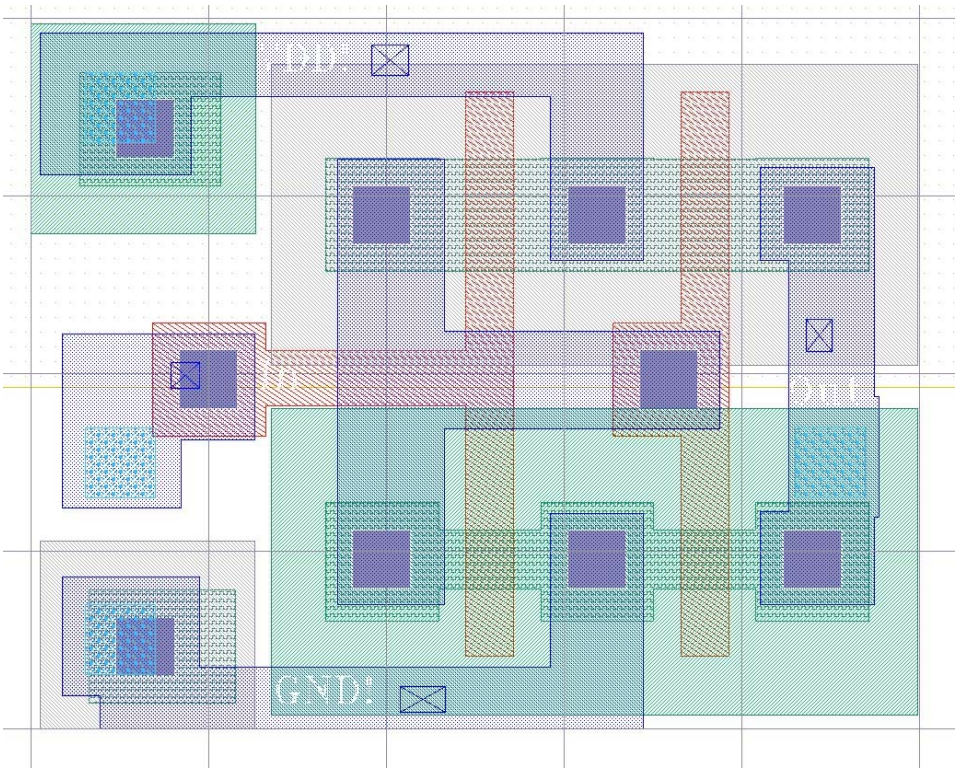
Inverter 1x



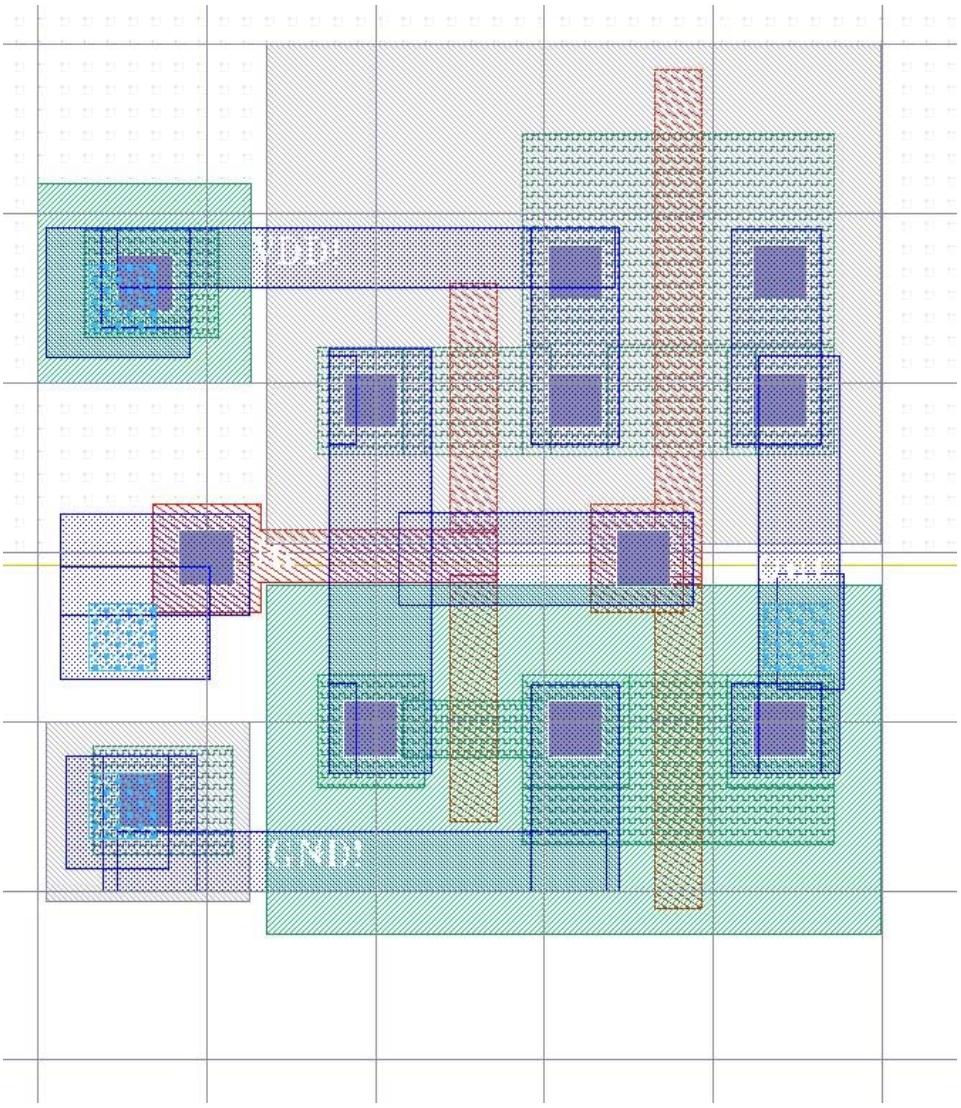
Inverter 2x



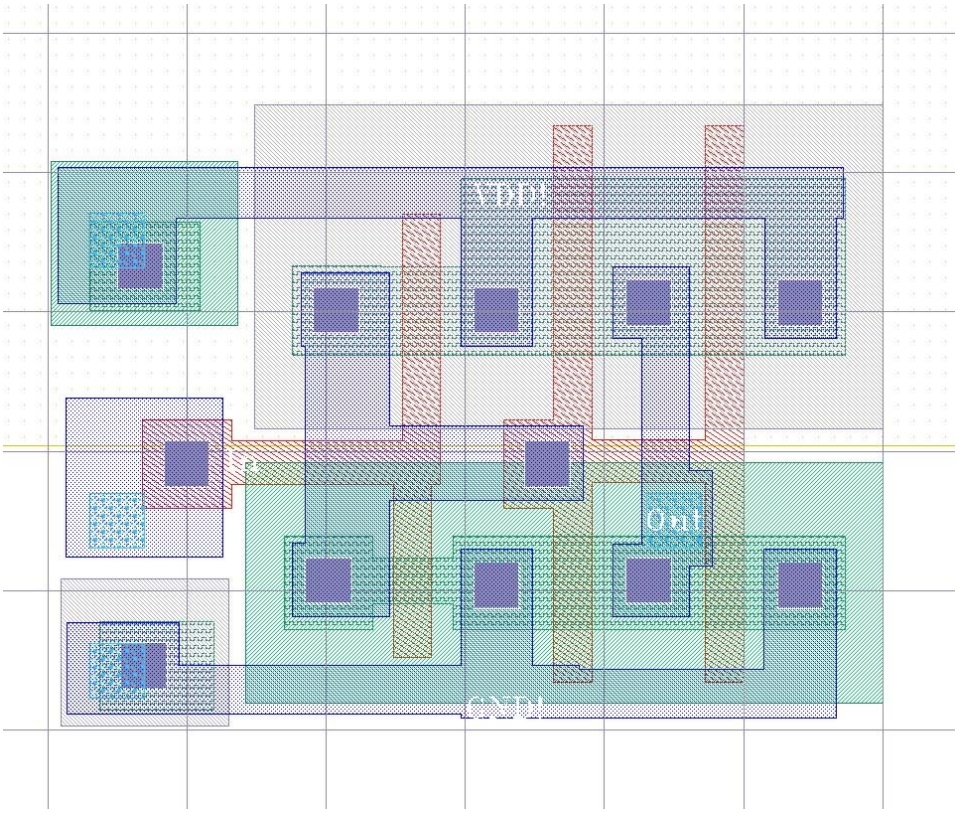
Inverter 4x



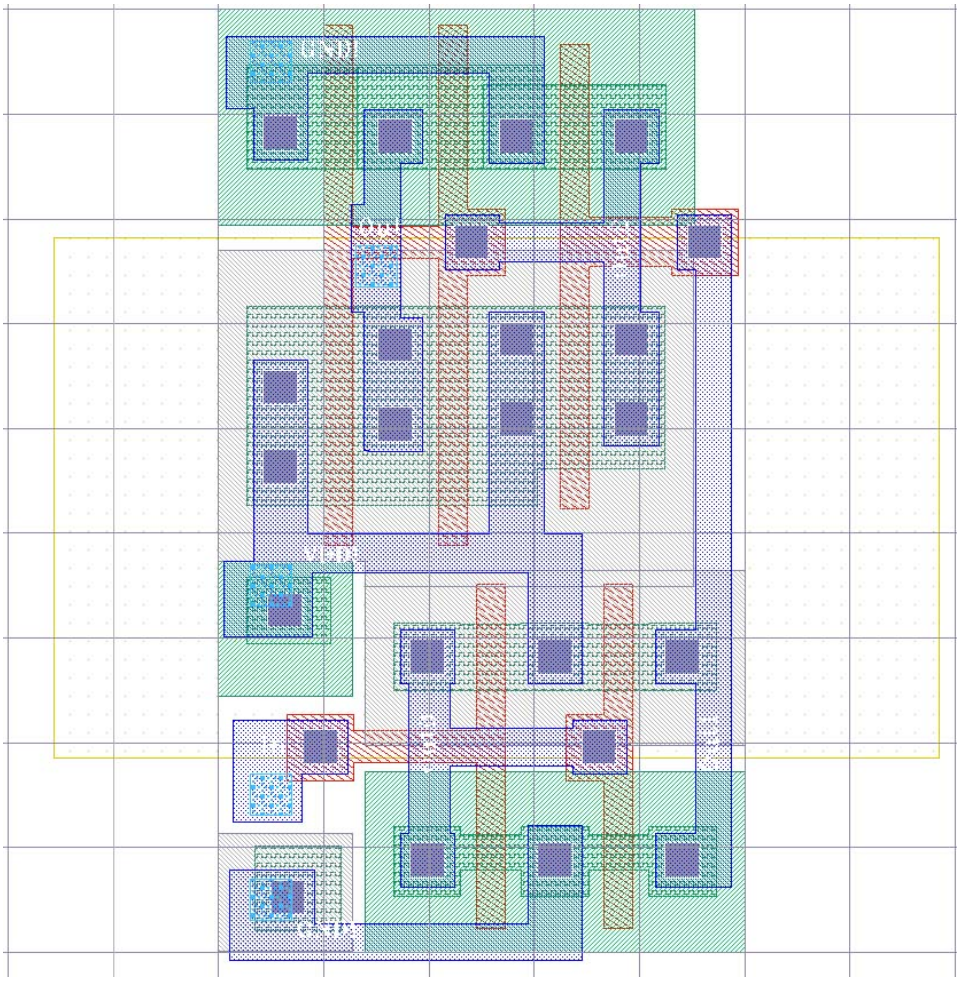
Buffer 1x



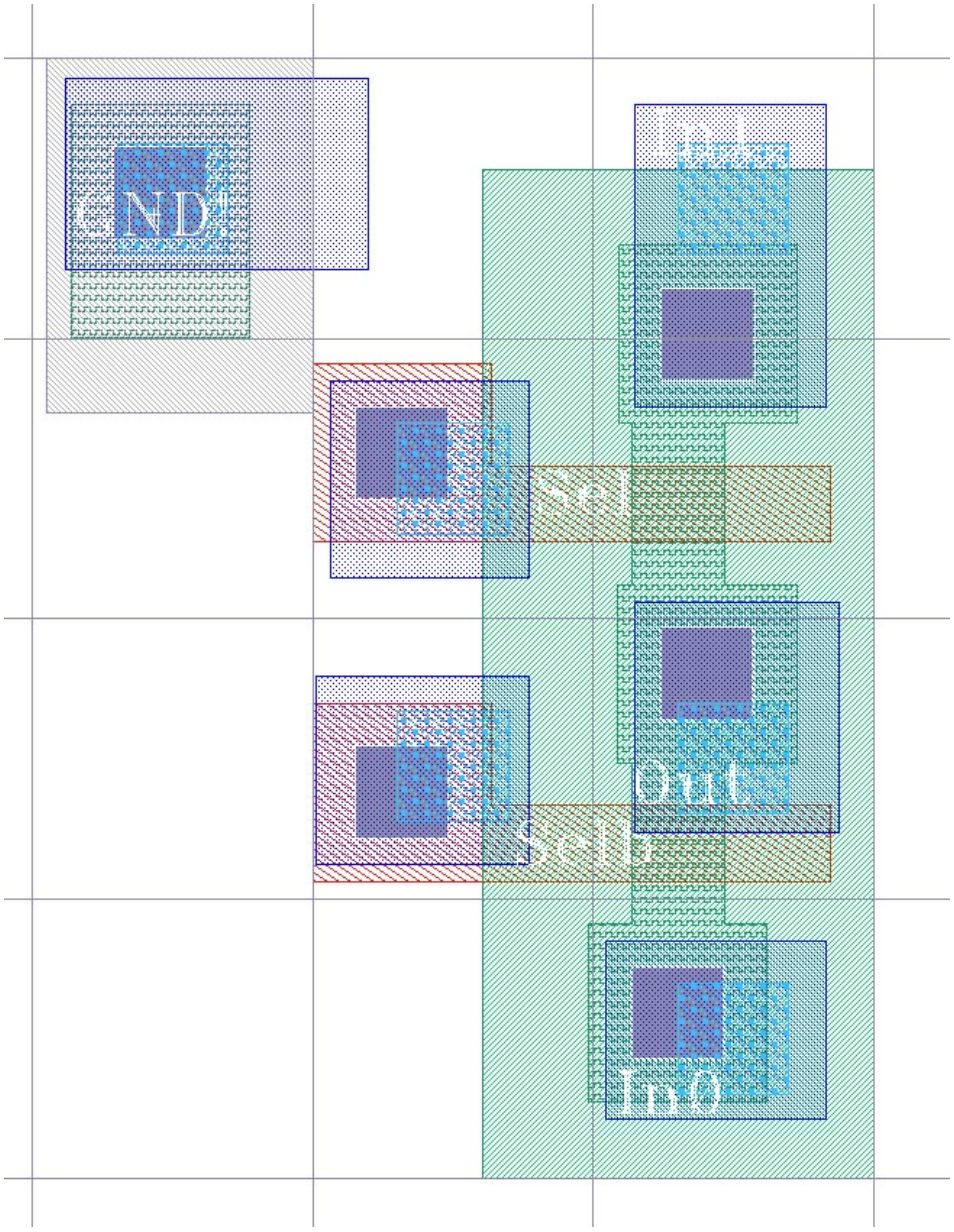
Buffer 3x



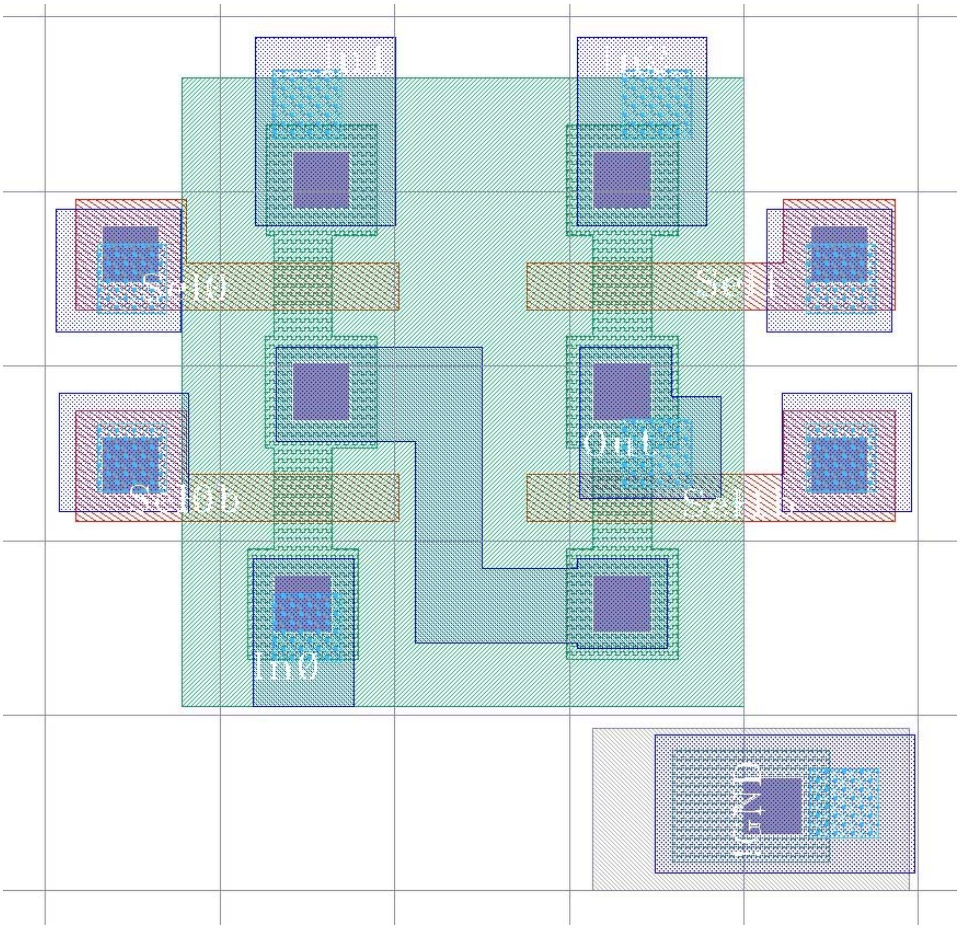
Buffer 4x



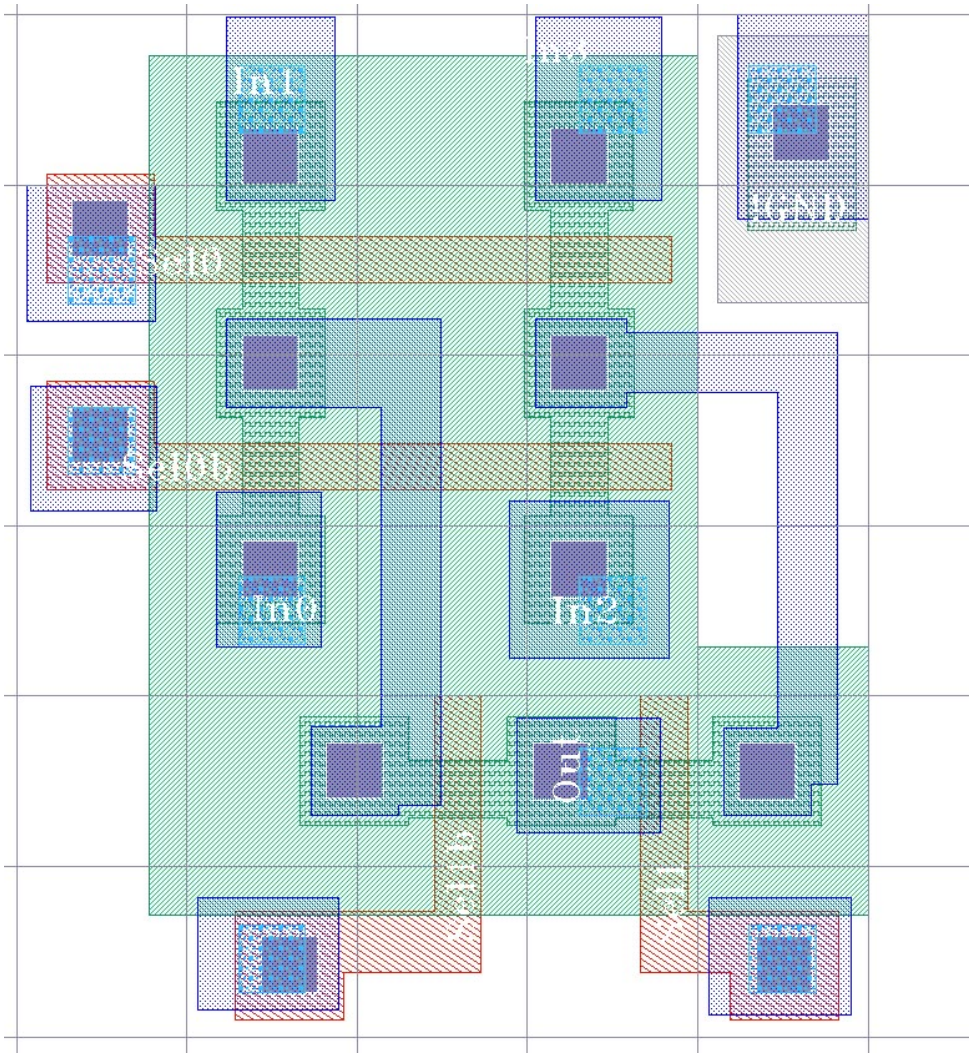
Buffer 6x



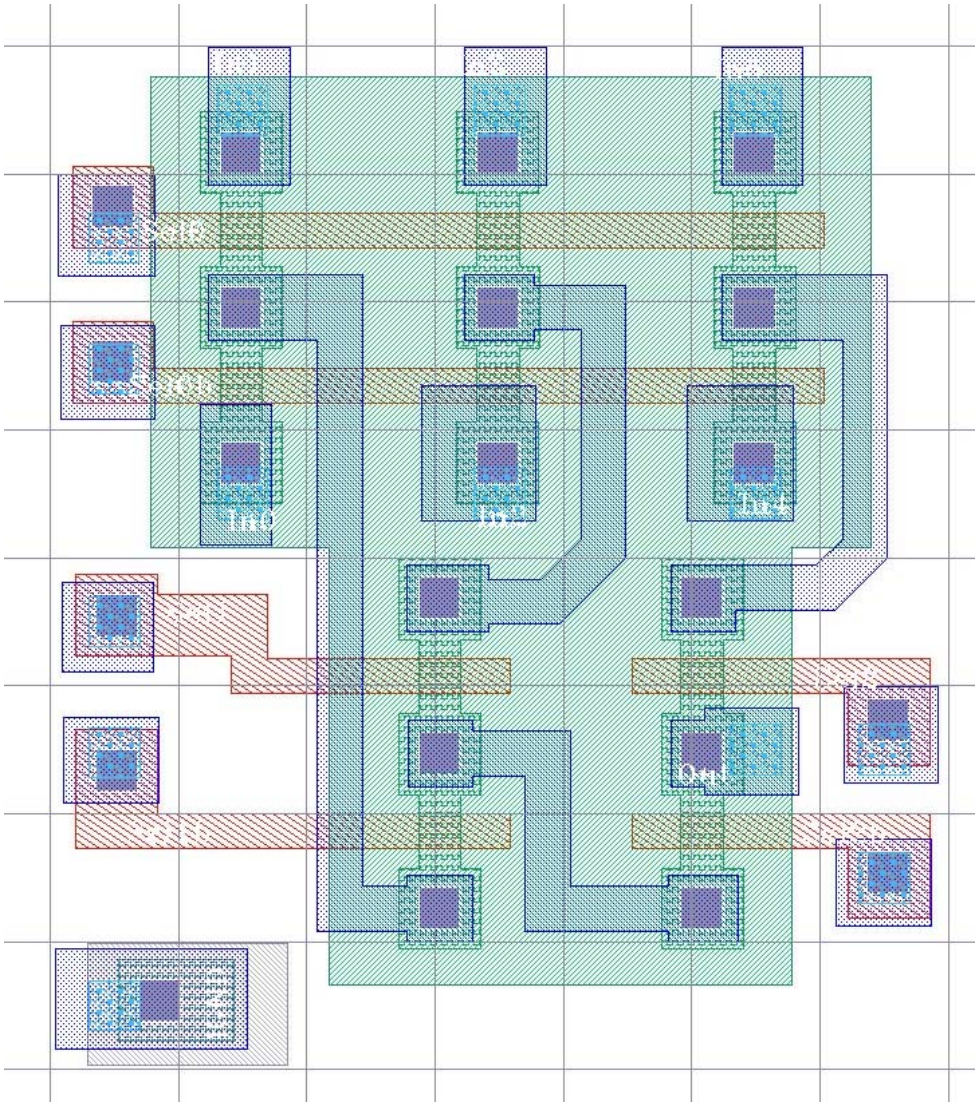
2-input Multiplexer



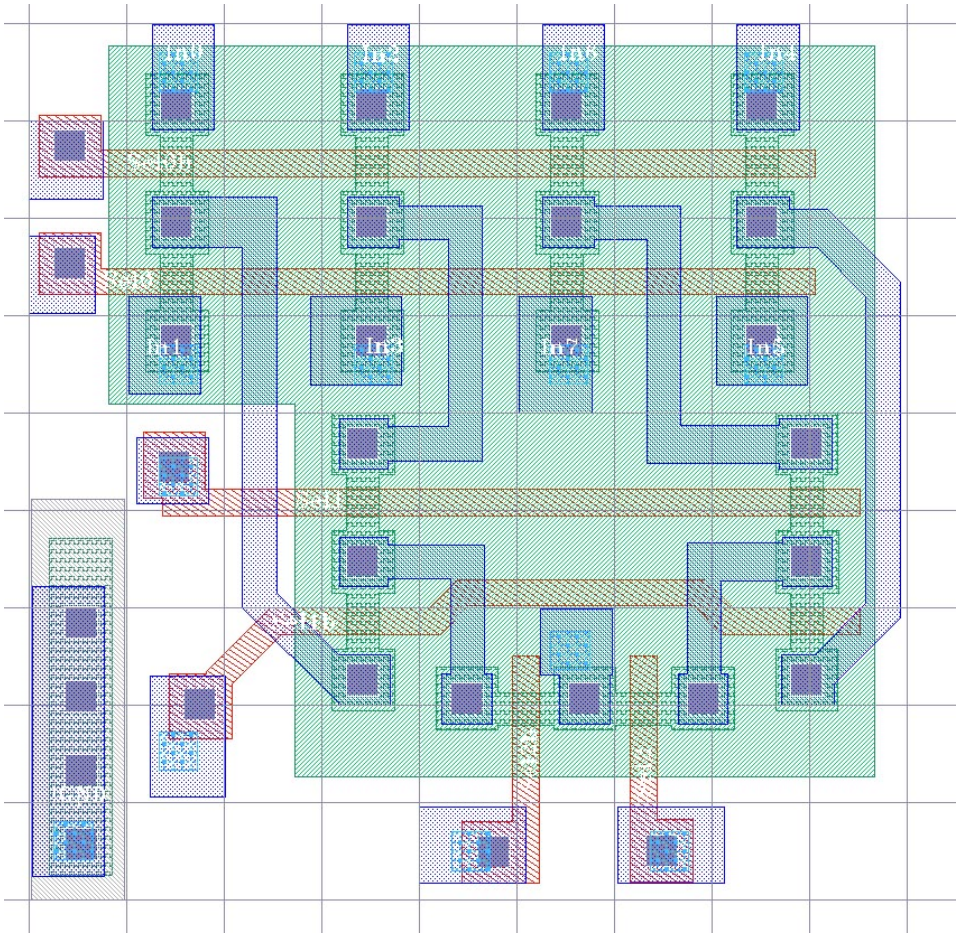
3-input Multiplexer



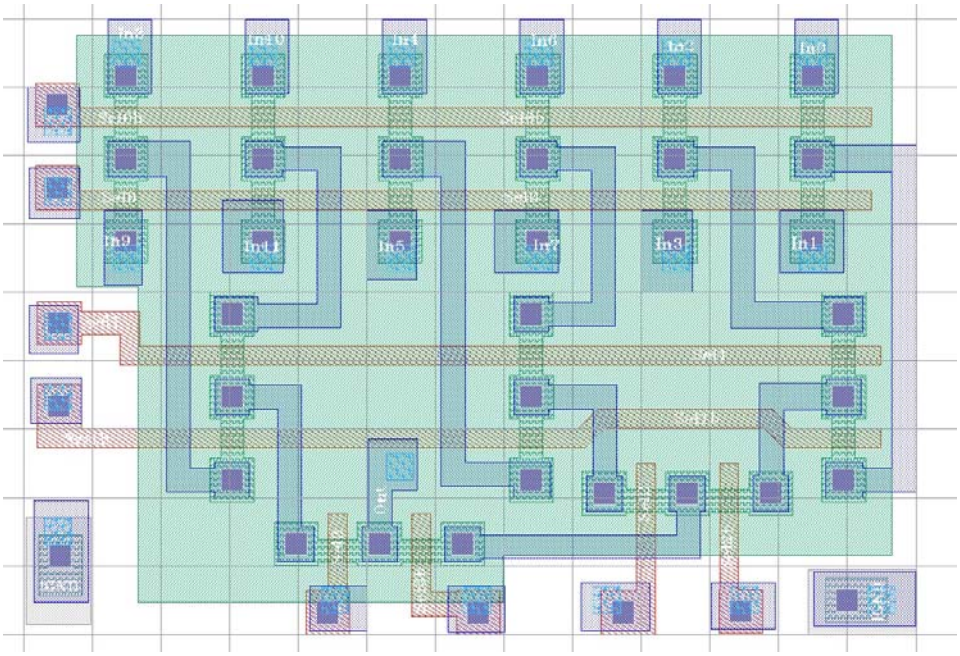
4-input Multiplexer



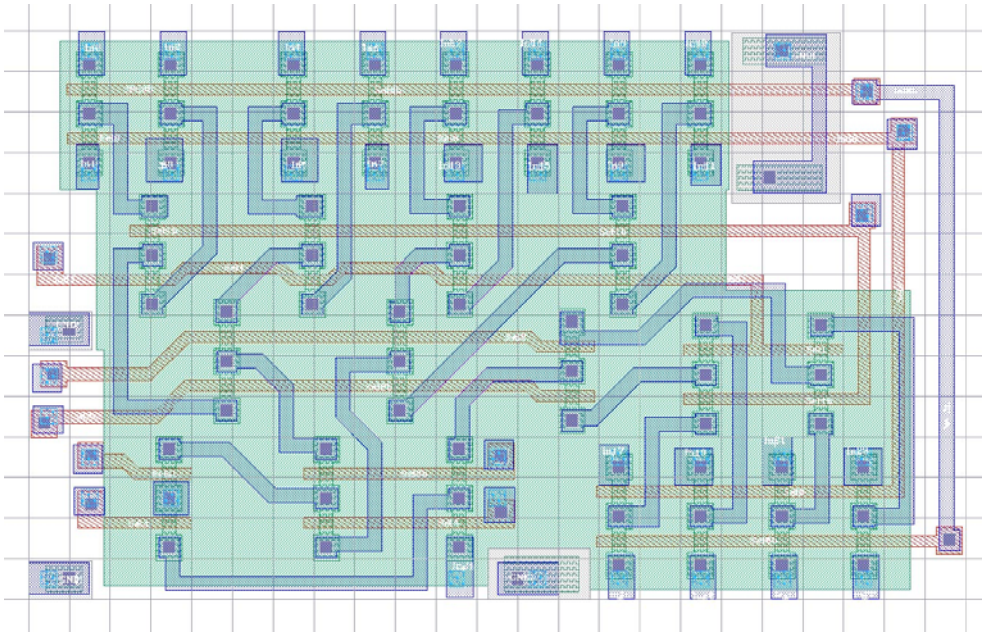
6-input Multiplexer



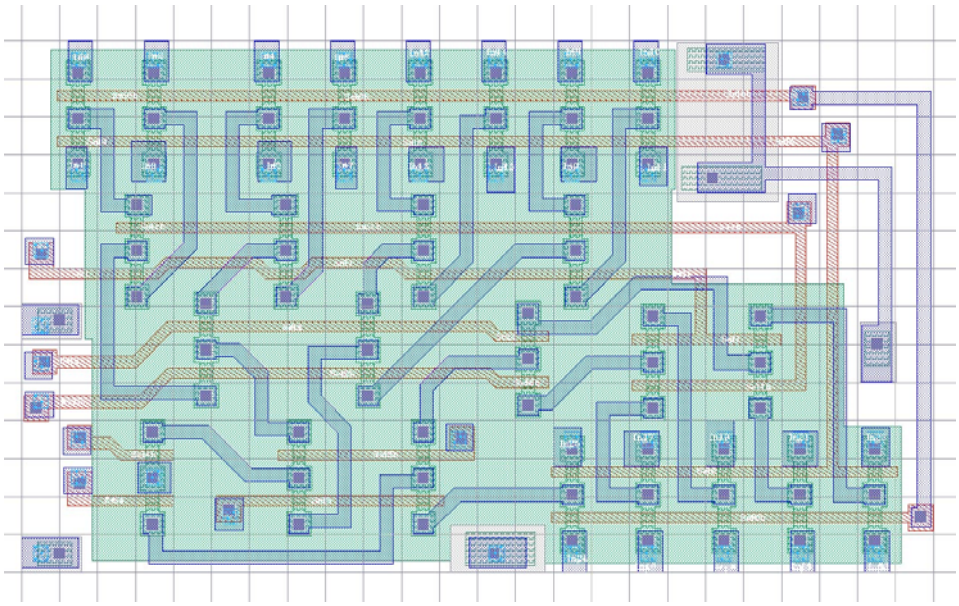
8-input Multiplexer



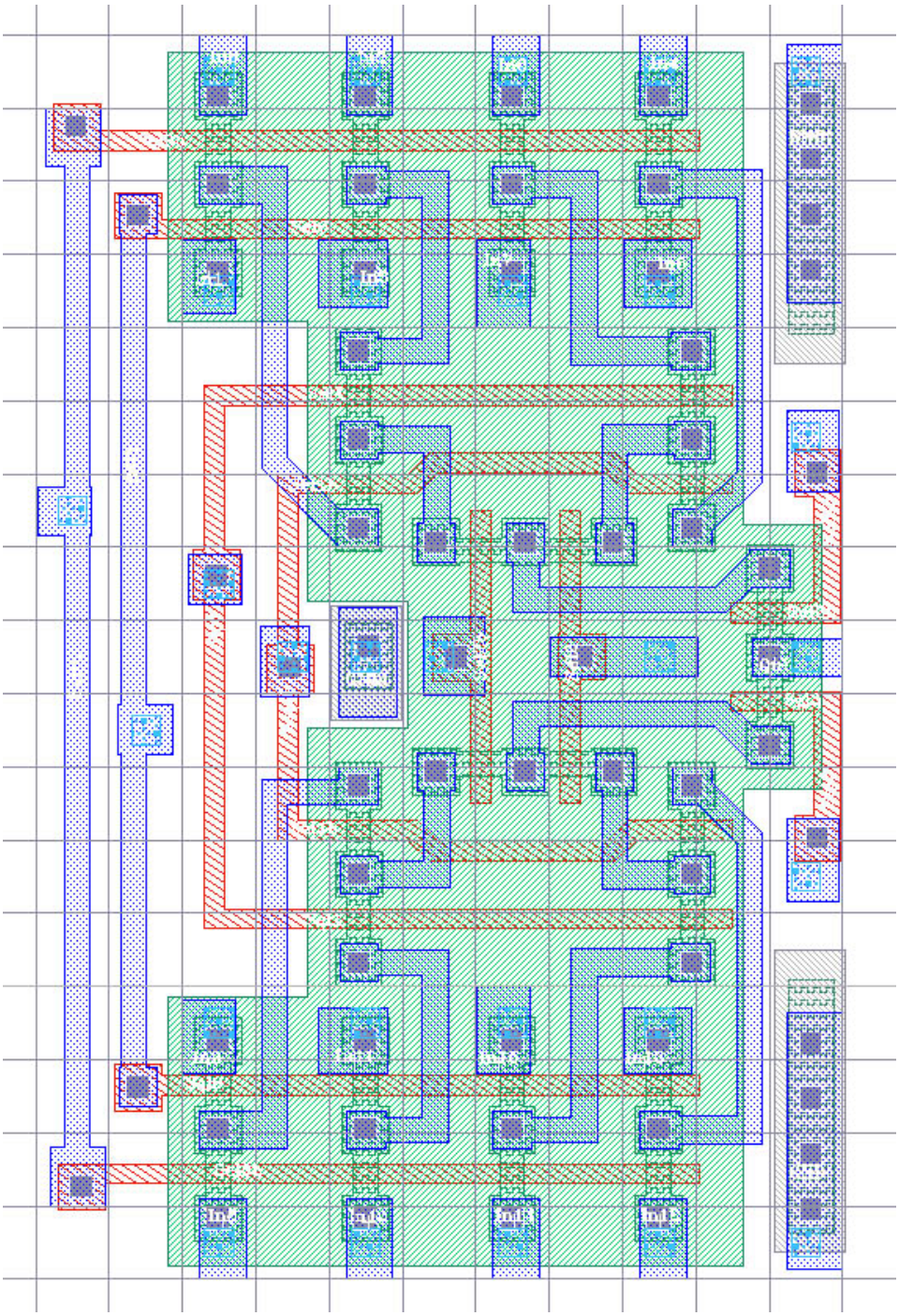
12-input Multiplexer



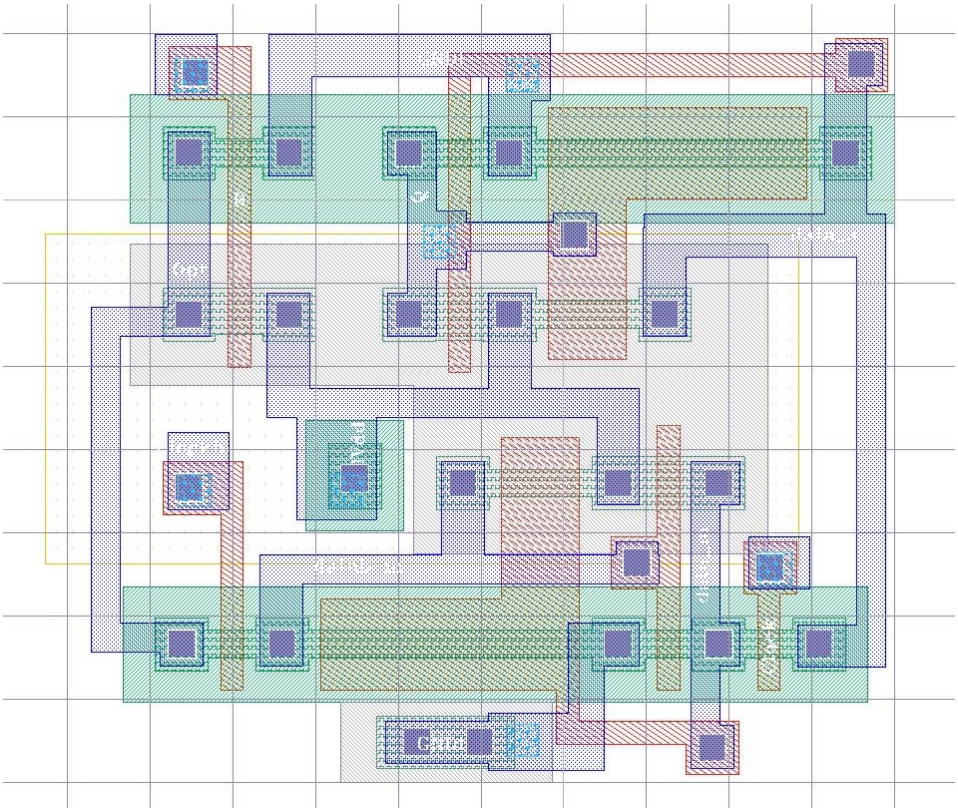
25-input Multiplexer



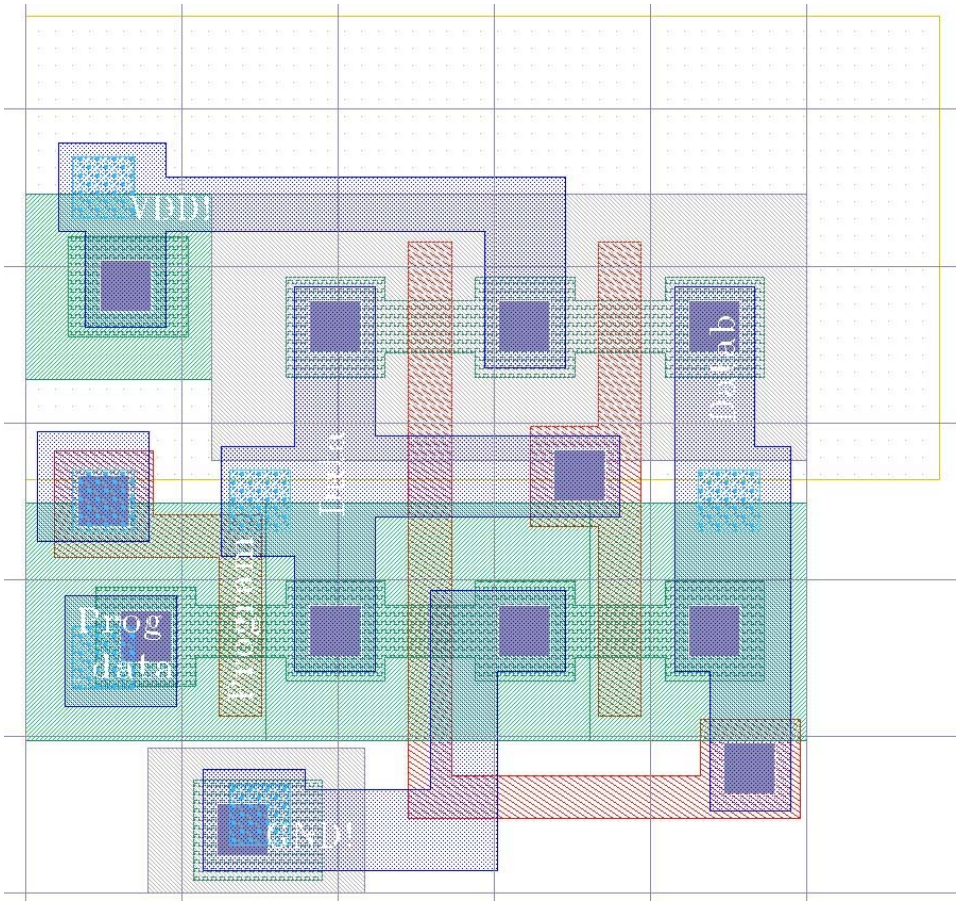
26-input Multiplexer



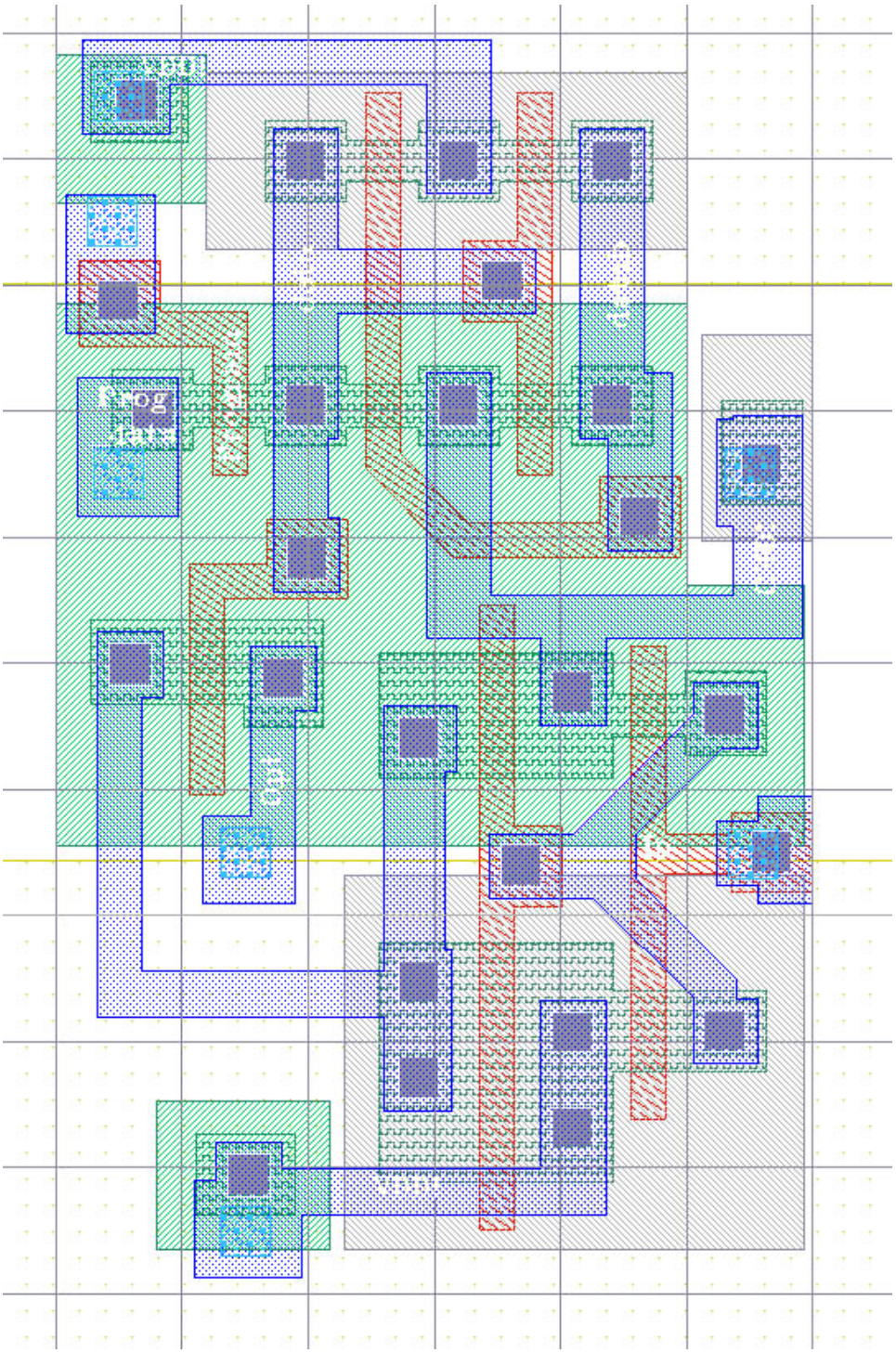
LUT



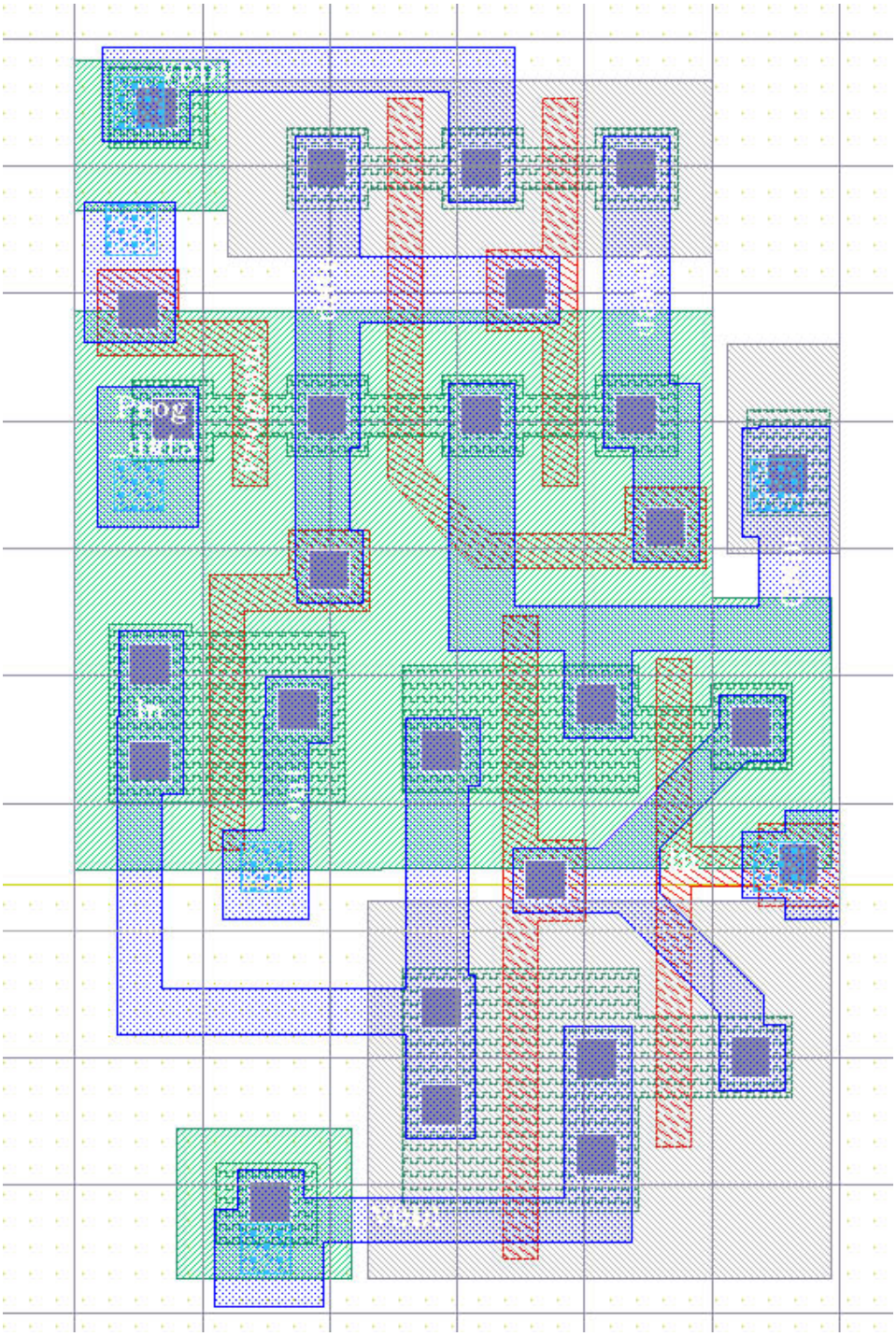
Flip-flop



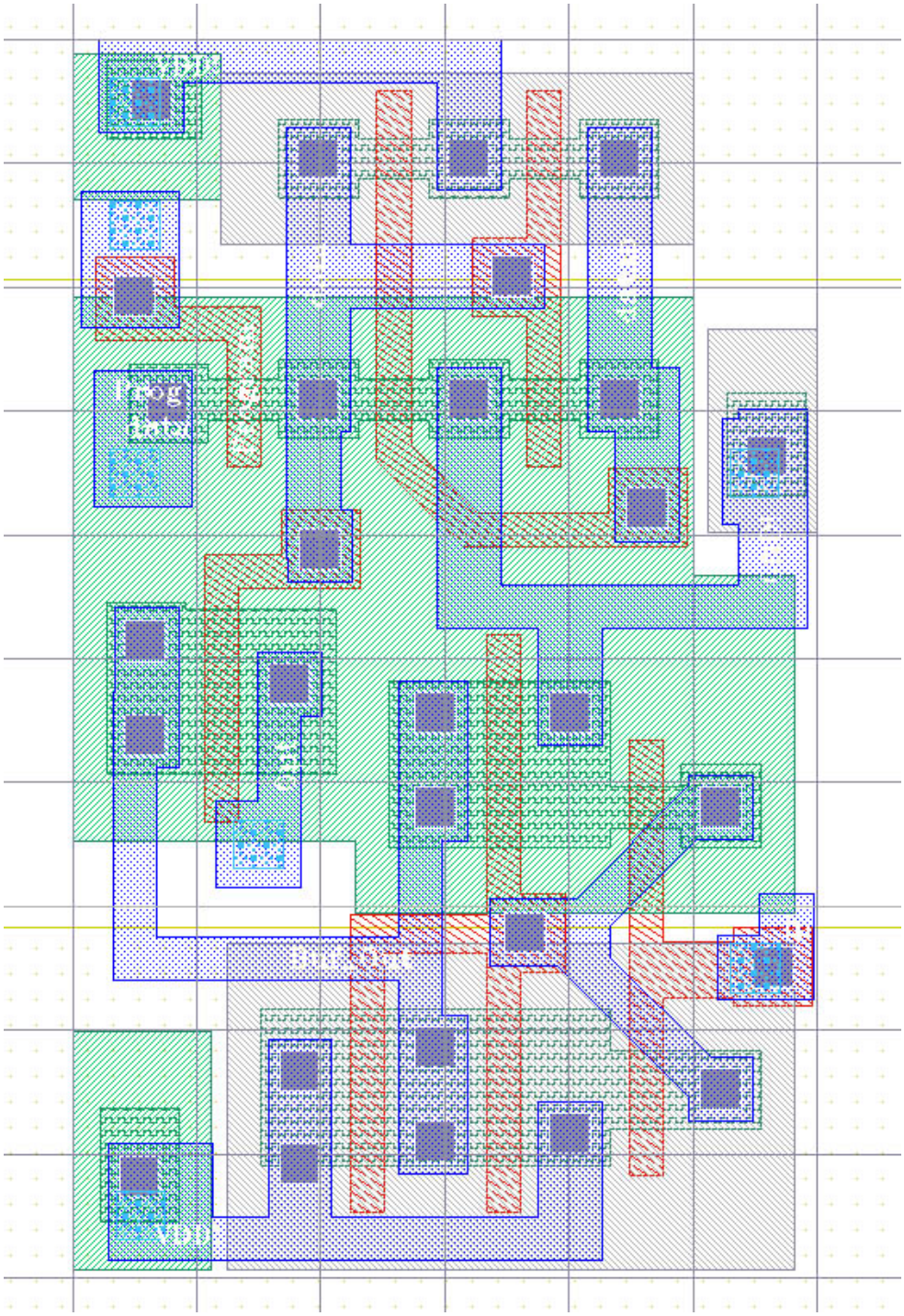
SRAM



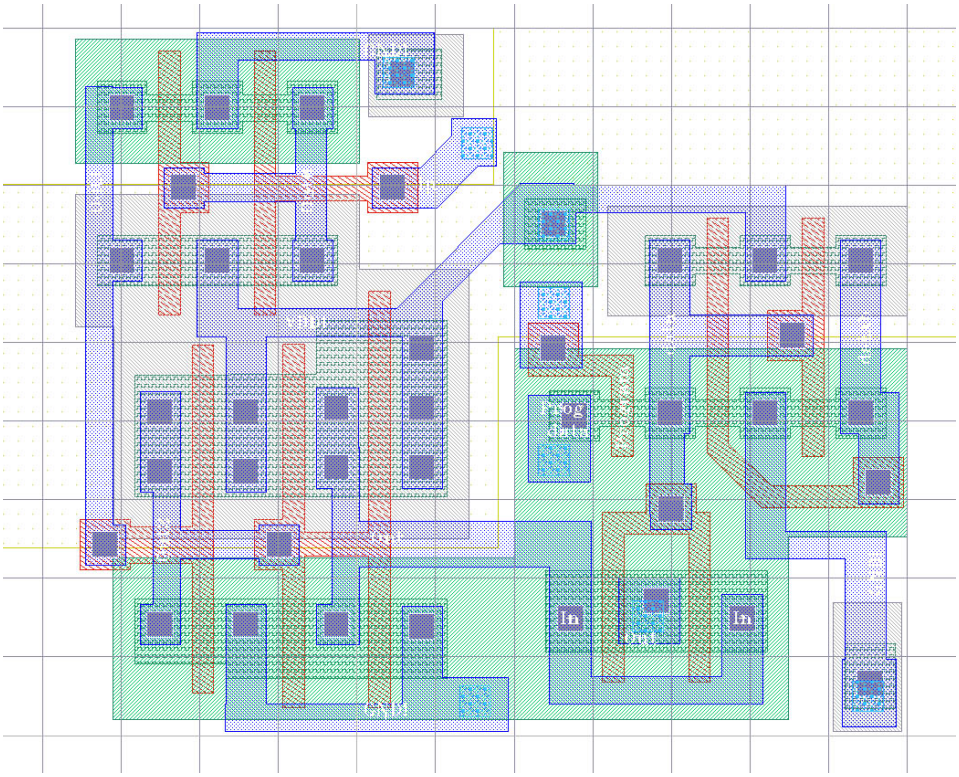
SRAM-Pass transistor 2x-Buffer 3x



SRAM-Pass transistor 4x-Buffer 3x



SRAM-Pass transistor 4x-Buffer 4x



SRAM-Pass transistor 6.25x-Buffer 6x

References

[1] Padalia, Ketan.. Automatic Transistor-Level Design and Layout Placement of FPGA Logic and Routing from an Architectural Specification. Bachelor's Thesis. University of Toronto, 2001

[2] Fung, R.. Optimization of Transistor-Level Floorplans for Field-Programmable Gate Array. Bachelor's Thesis. University of Toronto, 2002

[3] Kuon, I, Egier A, Rose J. Transistor Grouping and Metal Layer Trade-offs in Automatic Tile Layout of FPGAs. submitted to Great Lake Symposium on FPGAs, December 2003

[4] V. Betz, J. Rose and A. Marquardt, Architecture and CAD for Deep-Submicron FPGA, Kluwer Academic Publishers: Boston, 1999