Available online at https://sej.journals.ekb.eg/
FACULTY OF ENGINEERING – SOHAG UNIVERSITY
Sohag Engineering Journal (SEJ) Vol. 4, No. 1, March 2024

Layout Automation of Variable Gain Amplifier Circuit Based on TCL Language

Mohamed Gamal*, Ayad Shohdy, Ahmed E. A. Farghal

Electrical Engineering Department, Faculty of Engineering, Sohag University, 82524, Sohag, Egypt

Abstract

This paper presents a method for automating the layout design of a variable gain amplifier (VGA) using the tool command language (TCL) scripting language. TCL automation involves writing scripts that automate tasks such as design synthesis, simulation, verification, and layout generation. The proposed method consists of two steps: first, generating a TCL script that describes the desired layout, and then executing the TCL script to generate the layout. The TCL script is generated by a layout generator, which takes the specifications of the VGA as inputs and generates a TCL script that describes the layout in terms of TCL commands. The TCL script is then executed by a layout placer, which places the cells in the layout according to the instructions of the TCL script. The proposed method has been implemented and evaluated on a given VGA circuit. The results show that the proposed method can automate the layout design of VGA with high accuracy and efficiency.

 DOI: 10.21608/SEJ.2023.235841.1046

Keywords: Layout Automation; Analog Layout; TCL Scripting Language; Floor Planning.

1. INTRODUCTION:

Due to unparalleled technological progress and a lively receptiveness in society represented in economic and industrial environments, microelectronic products inexorably continue to control, connect, and change our world in the 21st century [1]. Apart from the triumphal procession of smartphones, contemporary achievements such as activity trackers, delivery drones and electric vehicles, as well as the ongoing innovations regarding bioimplants, driverless cars and the Internet of Things (IoT) [2] not only address numerous medical, logistical and environmental problems of today, but profoundly revolutionize modern life in many more aspects. Driven by a market demand for low-cost, multi-purpose, and densely integrated circuits (ICs), the semiconductor industry can be seen to follow a trend towards system-on-chip solutions with a growing amount of mixed-signal content, i.e., both digital and analog IC sections [3]. And while digital circuits have long been the main field of interest due to the ever-rising need for more computational power, now analog circuitry also gains notable attention, primarily incited by the desire for functional diversification and system integration [4]. Among others, the soaring importance of sensor functionality and the sophistication of advanced human-machine interfaces make analog circuit parts more and more indispensable.

To keep up with increasing chip complexity and shortening product life cycles, the task of creating an IC design asks for electronic design automation, both in the digital and analog domains [5-6]. But although digital IC design already followed highly automated algorithmic synthesis flows early on, analog circuits are commonly still handcrafted by expert designers with only a low grade of automation. Especially the step of layout design, where a circuit schematic is turned into a physical circuit implementation represents a severely time-consuming bottleneck in the overall design flow as shown in Fig. 1 [7].

Recently, there has been a growing demand for integrating mixed analog digital functions on a single very large-scale integration (VLSI) chip [7-9]. However, due to the lack of effective analog computer aided design (CAD) tools, the custom design cost required for the analog part often imposes a bottleneck for the chip. The development of automatic analog design and layout synthesis tools is thus of vital importance. The layout automation for analog ICs is considered more difficult to deal with than their digital counterparts. To effectively address the diverse analog circuit designs, a custom layout style based on flexible module generation techniques is necessary. Special care needs to be taken to minimize electrical performance degradations due to device mismatches, parasitics [10], and noise couplings in the analog circuit layout. The layout also needs to

* Corresponding author: e-mail: mahamadgamal51@gmail.com

accommodate large variations in analog device sizes and to maintain a compact area. Furthermore, it is desirable that the layout can be quickly changed for different physical aspect ratios. And finally, the layout tool is to be user-friendly for system designers who typically have limited analog circuit knowledge.

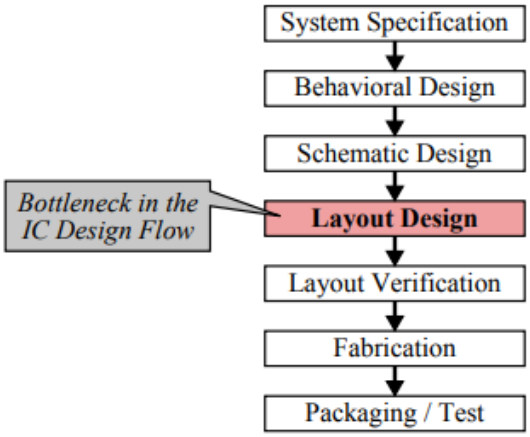


Fig. 1. Simplified illustration of the integrated circuit design flow.

The difficulty in analog layout generation lies in ensuring achievement of the desired performance specifications upon fabrication. Therefore, analog layouts, traditionally, have been drawn manually by expert layout designers at the cost of increased design cycle time. In order to address this issue, presents a methodology for analog layout automation by employing a layout reuse principle, that incorporates the designer's expertise embedded in existing layouts. However, there are also some challenges to automation in layout [11]. One of the biggest challenges is the need for designers to stay up-to-date with the latest software and tools. With technology evolving rapidly, designers must continually learn new skills and adapt to new software updates. This can be time-consuming and requires additional training. Another challenge is the need for designers to balance automation with creativity. While automation can save time and improve consistency, it can also result in designs that lack uniqueness and creativity. Therefore, designers must find a balance between using automation tools and applying their creativity to produce designs that stand out. Also, Layout design is a complex and challenging task, even for experienced engineers. The layout designer must carefully consider a number of factors, including the transistor sizing, the layout parasitics, and the overall performance of the circuit.

In this paper an effective automated method based on TCL language [12] will be utilized for VGA layout automation. VGA is an electronic device (amplifier) that varies its gain depending on the applied control voltage. Also, it is in use in various applications, including synthesizers, amplitude modulation, and audio level compression. TCL is a scripting language that is commonly used in electronic design automation (EDA) tools to automate various design tasks. TCL automation involves writing scripts that automate tasks such as design synthesis, simulation, verification, and layout generation. TCL automation can help improve the productivity and efficiency of the design process by automating repetitive and time-consuming tasks. It also helps ensure consistency and accuracy in the design process by reducing the chances of human error. Additionally, TCL automation can help improve the design quality by enabling designers to explore design alternatives quickly and efficiently. TCL has many features such as [12]: (i) reduced development time; (ii) powerful and simple user interface kit with integration of TK; (iii) Write once, run anywhere, it runs on Windows, Mac OS X, and almost every Unix platform. (iv) it is quite easy to get started for experienced programmers; since, the language is so simple that they can learn TCL in a few hours or days. (v) You can easily extend existing applications with TCL. Also, it is possible to include TCL in C, C++, or Java to TCL or vice versa. (vi) Have a powerful set of networking functions. (vii) Finally, it's open source, free, and can be used for commercial applications without any limit.

The rest of the paper is outlined as follows: section two provides layout generation process; a TCL code execution is presented in section three and lastly a conclusion is given in section four.

2. LAYOUT GENERATION PROCESS

The schematic diagram of the top level of the VGA circuit which is aimed to generate its automated layout is given in Fig. 2. The VGA circuit design has three main blocks: (i) Common mode feedback (CMFB) circuit as shown in Fig. 3 is often used in VGA circuits to stabilize the common-mode voltage of the amplifier and improve its performance. In an amplifier circuit, the common-mode voltage is the voltage level that is present on both the input and output terminals of the amplifier. In a differential amplifier configuration, the common-mode voltage is

the average of the two input voltages. The common-mode voltage can vary depending on factors such as temperature, component tolerances, and power supply voltage fluctuations. If the common-mode voltage of an amplifier is not properly controlled, it can lead to problems such as offset voltage, distortion, and instability. CMFB is a technique used to stabilize the common-mode voltage of an amplifier by providing feedback that adjusts the voltage level to a desired value. In a VGA circuit, CMFB is used to stabilize the common-mode voltage of the amplifier over a wide range of gain settings. As the gain of the amplifier is adjusted, the common-mode voltage can vary, which can lead to distortion and instability. CMFB provides feedback that adjusts the common-mode voltage to a desired level, regardless of the gain setting. CMFB can be implemented using various circuit topologies, such as op-amp or transistor-based circuits. The specific implementation depends on the requirements of the application, such as the input and output impedance, bandwidth, noise, and distortion. (ii) Operational transconductance amplifier (OTA) which are often used as the building blocks for VGA circuits. To implement a VGA circuit shown in Fig. 4., an OTA circuit can be used as the variable gain element. By adjusting the control voltage or current of the OTA circuit, the gain of the VGA circuit can be adjusted over a wide range of values. This makes OTA circuits a useful building block for VGA circuits, as they provide a flexible and controllable way to adjust the gain of a signal. (iii) The top level of the VGA shown in Fig. 2: VGA that uses a field effect transistor (FET) as the gain control element ($S_O, S_1,\ S_2,$ and S_3), the current flowing through the FET is varied to adjust the gain of the amplifier. The FET is connected in series with the feedback resistor of the amplifier, and its gate is used to control the current flowing through the FET. By changing the voltage applied to the gate of the FET, the current flowing through the FET changes, which in turn changes the gain of the amplifier.

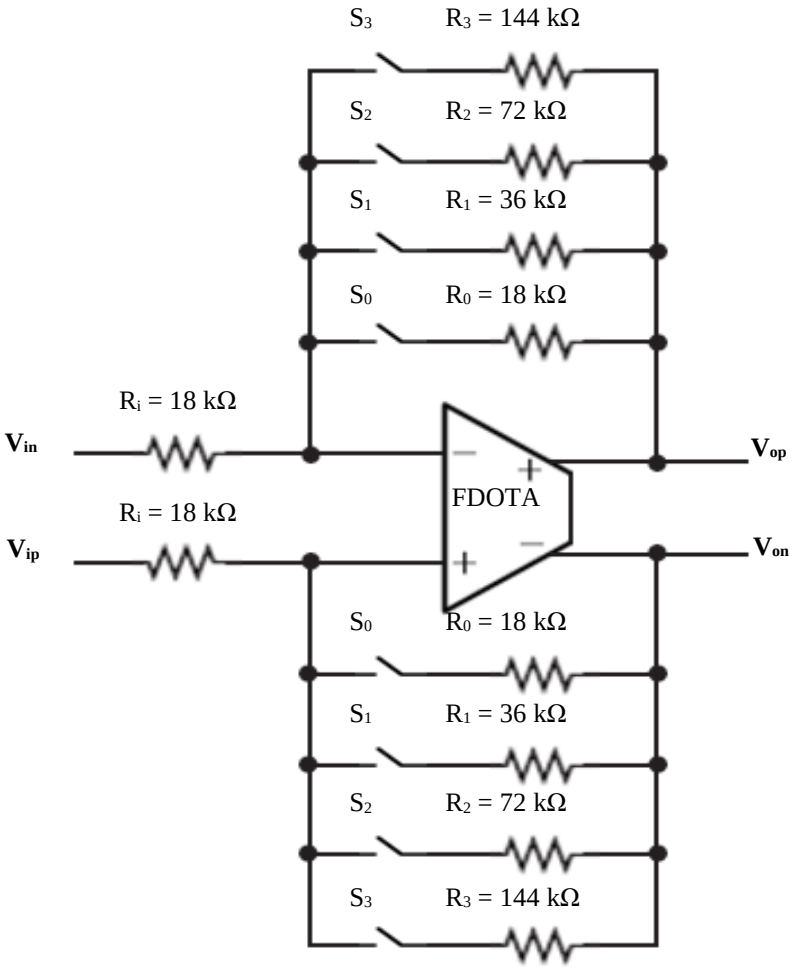


Fig. 2. The schematic diagram of the top level of the VGA.

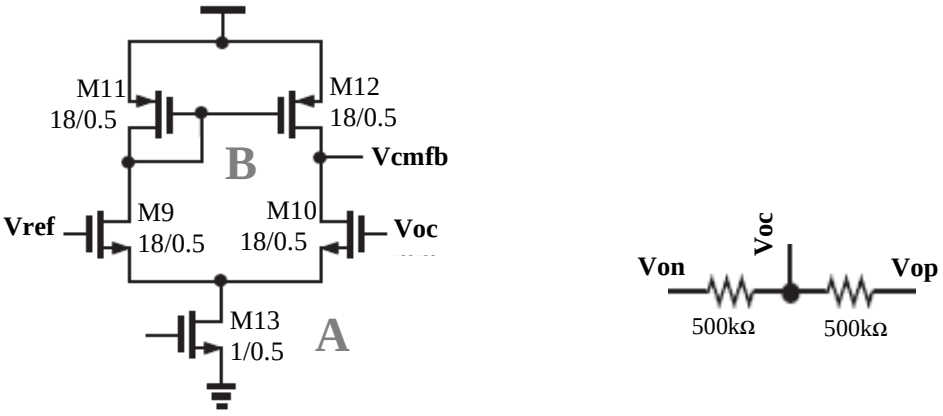


Fig. 3. The schematic diagram of the CMFB.

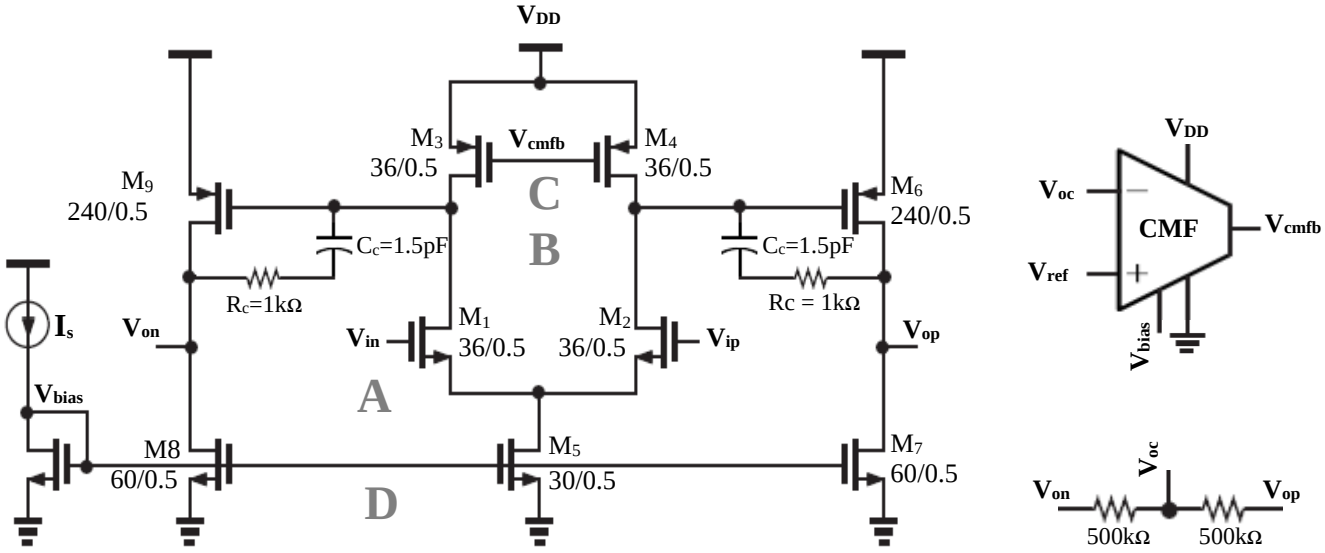


Fig. 4. The schematic diagram of the OTA.

The process of layout automation includes three main steps: Pattern Generation, Floor planning and Routing. These are the main stages that will be discussed, and the results related to our circuit, including TCL commands and circuit layout, will be provided. A custom compiler from Synopsys and Programming by TCL Language is used.

2.1. Pattern Generation

Pattern generation is the process of creating the individual blocks that make up the IC's layout. These blocks can be transistors, wires, capacitors, and interconnects, that will form the building blocks of the integrated circuit. The blocks are typically created using CAD tools and are based on the circuit's functional requirements and performance specifications. The accuracy and quality of the generated patterns are critical as they impact the overall performance and reliability of the circuit. The pattern generator is an important part of the layout automation process. It allows engineers to quickly and accurately generate the layout of complex circuits, which can save time and money. The pattern generator needs to know the circuit's design in order to create the correct shapes [7].

2.2. Floorplanning

Floorplanning is the process of organizing the patterned blocks generated by the previous stage on the chip's physical layout. The objective of this stage is to optimize the placement of the blocks for factors such as area, power consumption, and signal integrity. This is done by determining the location of each block on the chip and the interconnect points between the blocks. The floorplan also includes power and ground distribution networks, which are critical for ensuring proper voltage levels and signal integrity. The floorplanning stage in the layout automation process can impact the subsequent stages, such as the routing stage. If the floorplan does not provide enough routing resources, the routing algorithm may need to use longer routing paths, which can negatively impact the circuit's performance, power consumption, and timing requirements.

There are several common mistakes that designers can make during the floorplanning stage and should get rid of them for good floorplan like: (i) Insufficient Space: One of the most common mistakes in floorplanning is not allocating enough space between the patterned blocks on the chip. This can lead to congestion during the routing stage, which can negatively impact the circuit's performance, power consumption, and timing requirements. (ii) Poor Power Distribution Network: Another common mistake is not properly accounting for the placement of the power distribution network, which distributes power to the various blocks on the chip. A poorly designed power distribution network can lead to unstable power levels and impact the performance and reliability of the circuit. (iii) Ignoring Signal Integrity: Signal integrity refers to the ability of the interconnects to transmit signals without degradation. A common mistake is not accounting for signal integrity during the floorplanning stage, which can lead to crosstalk and signal degradation. (iv) Inefficient Placement: Inefficient placement of the patterned blocks can lead to a suboptimal layout, which can negatively impact the circuit's performance, power consumption, and timing requirements. The designer must use optimization algorithms to efficiently place the blocks on the chip and minimize congestion. (v) Poor Design Closure: Design closure refers to the process of refining and optimizing the layout to meet all design constraints and requirements. A common mistake is not iterating through the floorplanning stage enough times to achieve design closure, which can lead to a suboptimal final layout and negatively impact the circuit's performance, power consumption, and timing requirements. [7]

2.3. Routing

Routing is the final stage in layout automation, where the interconnects between the blocks are created to connect the various components of the integrated circuit. This involves determining the optimal routing paths for the interconnects and ensuring that they meet timing and other performance requirements. The routing stage can be challenging as the number of interconnects can be very large, and the routing resources are limited. The routing algorithm must ensure that the interconnects do not violate design rules, such as minimum line width and spacing, and that they meet timing constraints. The quality of the routing directly impacts the circuit's performance, power consumption, and reliability.

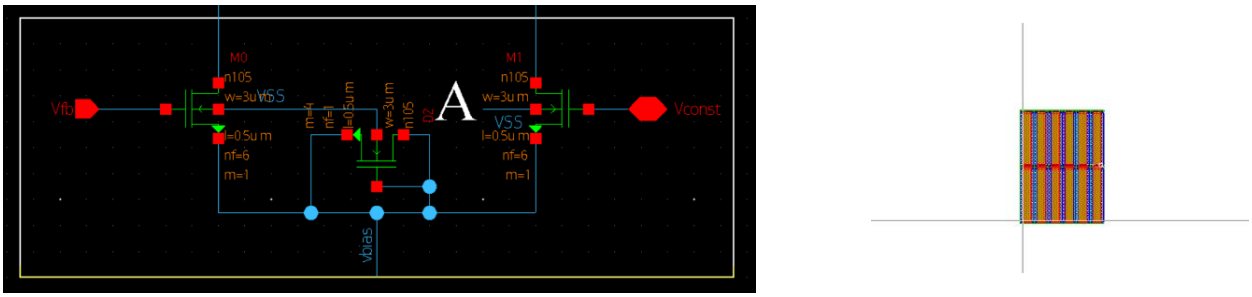
There are several common mistakes that designers can make during the routing stage such as: (i) Inefficient Routing: One of the most common mistakes in routing is inefficient routing, where the routing algorithm does not optimize the interconnects' length and placement. This can result in longer interconnects, which can negatively impact the circuit's performance, power consumption, and timing requirements. (ii) Congestion: Congestion can occur during routing when there are too many interconnects competing for limited routing resources. Congestion can lead to longer routing paths, which can negatively impact the circuit's performance, power consumption, and timing requirements. A common mistake is not accounting for congestion during the floorplanning stage, which can lead to congestion during the routing stage. (iii) Crosstalk: Crosstalk refers to the unwanted coupling of signals between adjacent interconnects. A common mistake is not accounting for crosstalk during the routing stage, which can lead to signal degradation and impact the circuit's performance and reliability. (iv) Design Rule Violations: Design rules specify the minimum feature size, pitch, and spacing between the patterned features on the chip. A common mistake is violating design rules during the routing stage, which can result in a non-manufacturable layout. (v) Incomplete Routing: Incomplete routing can occur when the routing algorithm is unable to create all the required interconnects due to congestion or other design constraints. Incomplete routing can result in a non-functional circuit [7].

3. TCL CODE EXECUTION

The circuit is divided into blocks and subblocks to generate its pattern then a floorplaning is applied to all blocks. The TCL codes of each block and subblocks are given in the appendix at the end of the paper. The code is running, and its result is presented.

3.1. CMFB Block:

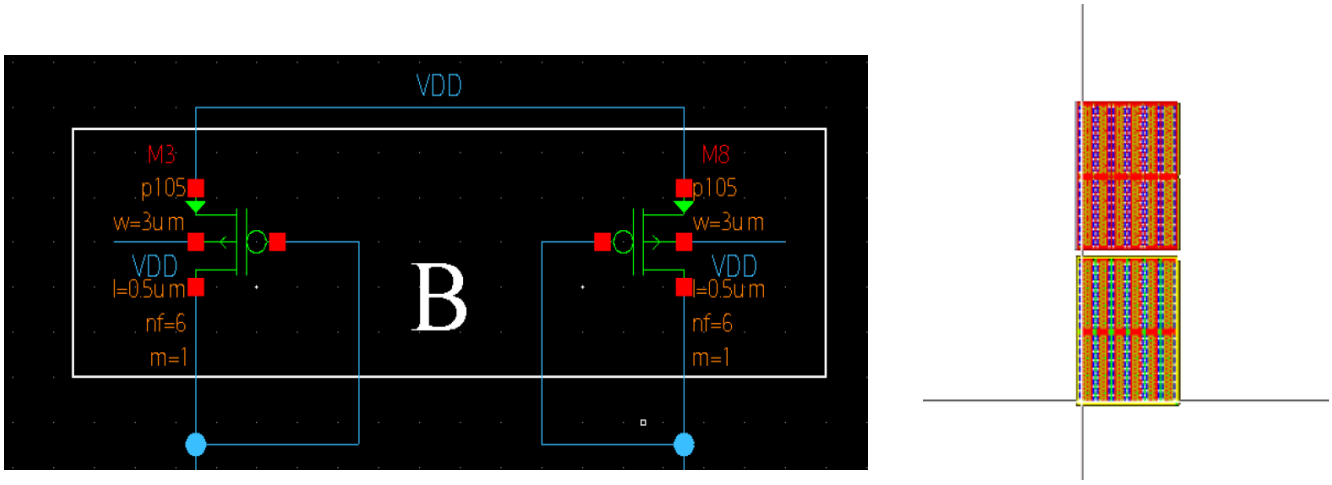
This block includes two subblocks: (i) CMFB subblock A as shown in Fig. 5(a). This subblock consists of two matching transistors differential pair. The generated layout is given in Fig. 5(b).



(a) (b)

Fig. 5. CMFB subblock A, (a) The schematic, (b) The automated layout.

(ii) CMFB subblock B shown in Fig. 6(a). This subblock B in CMFB is the current mirror with the block C in OTA. The generated layout added to CMFB subblock A and presented in Fig. 6(b).



(a) (b)

Fig. 6. CMFB subblock B, (a) The schematic, (b) The automated layout.

3.2. OTA Block:

OTA block includes six subblocks as shown in Fig. 7. The subblocks are generated and added in accumulative manner. The generated layouts of these subblocks are presented in Fig. 8.

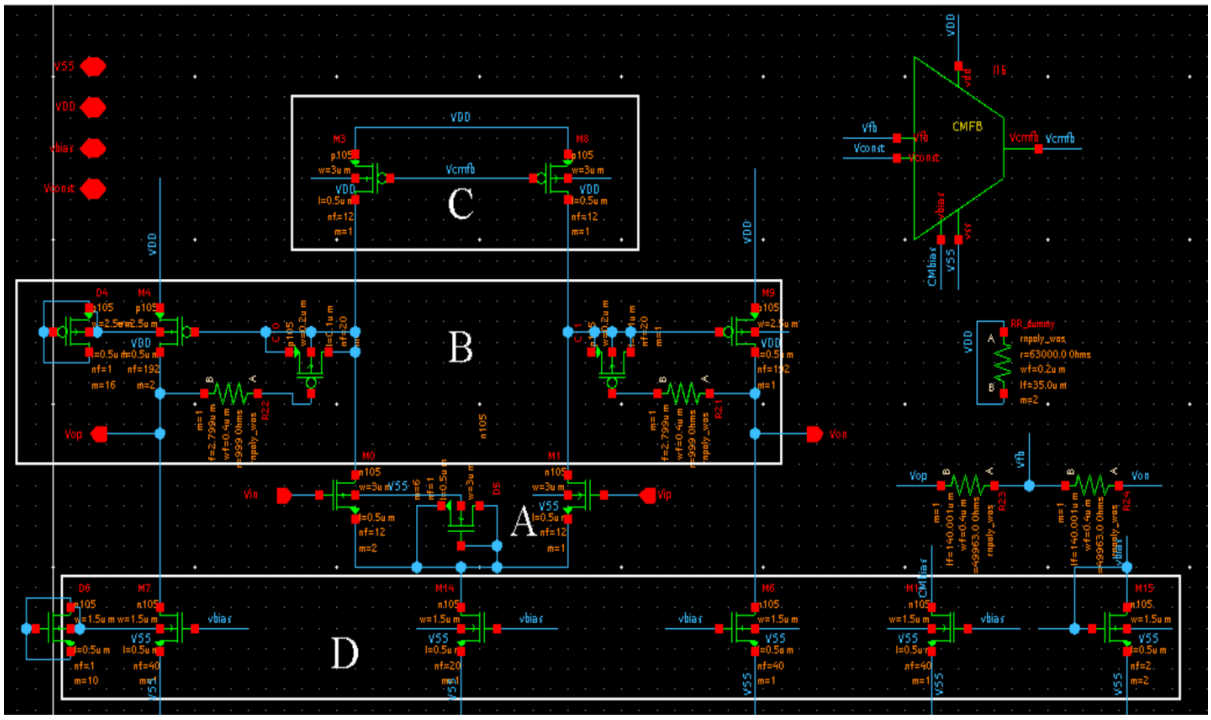


Fig. 7. Schematic of OTA subblocks.

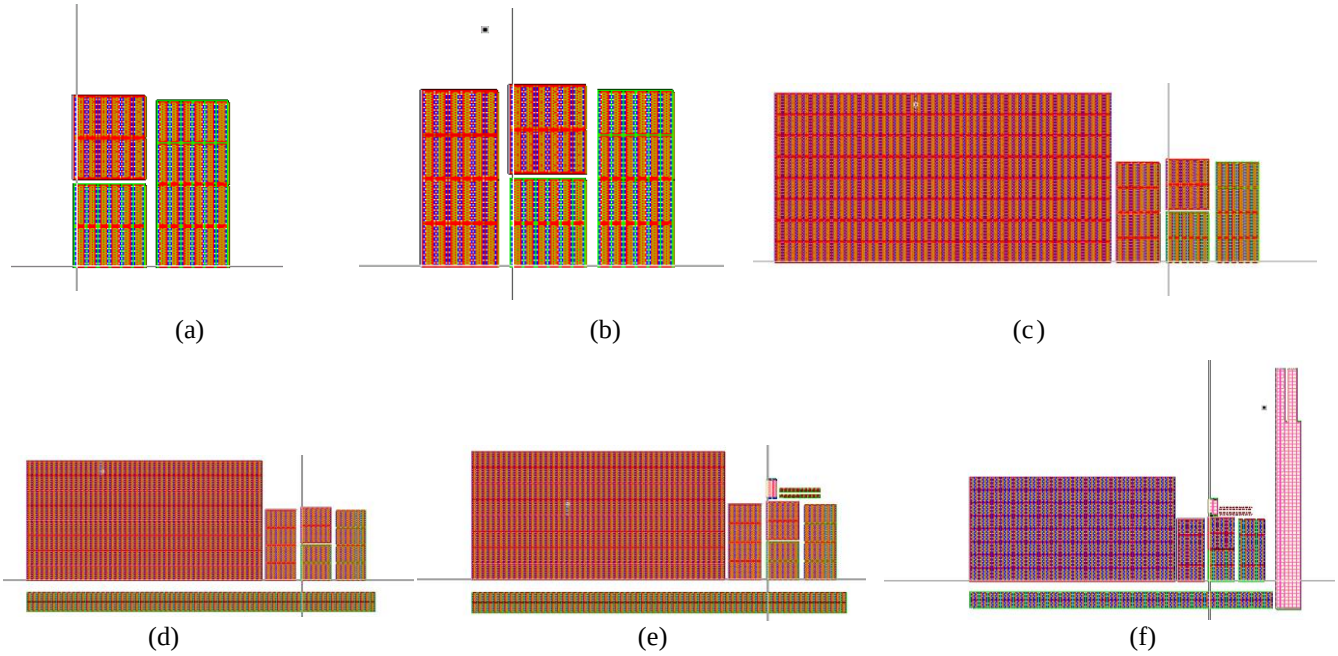
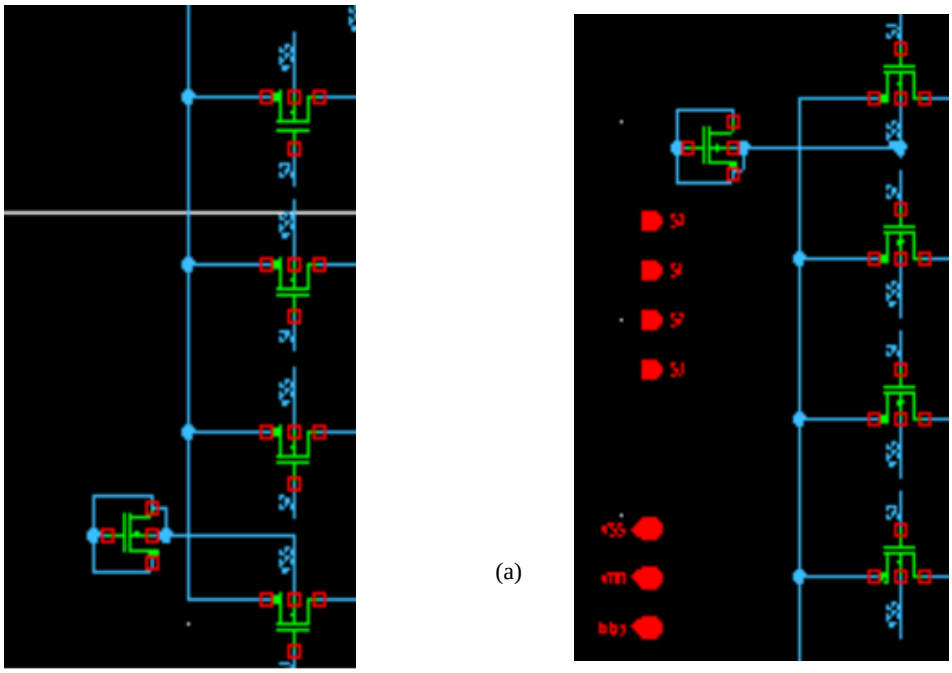


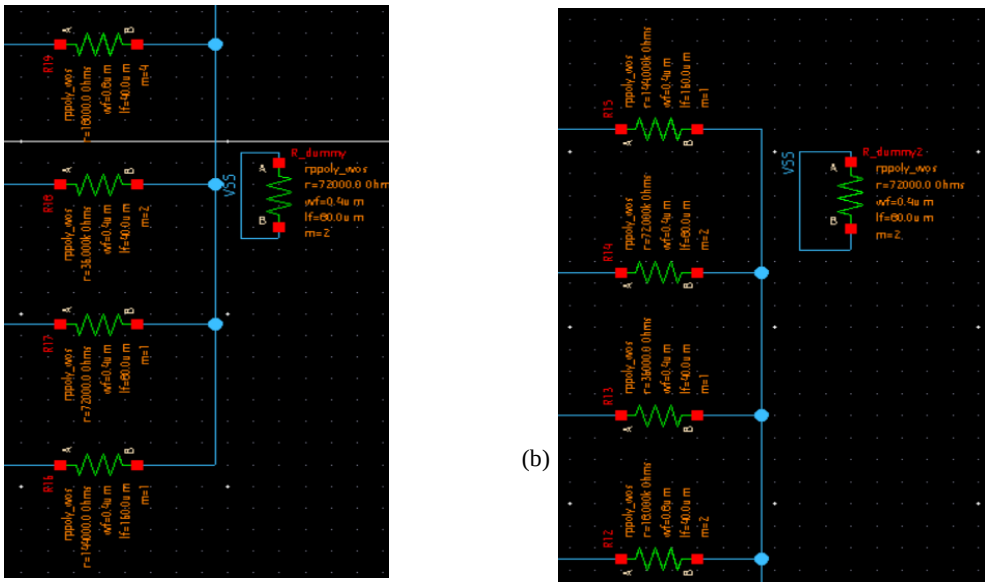
Fig. 8. OTA automated subblock layouts, (a) OTA subblock A, (b) OTA subblock C, (c) OTA subblock B, (d) OTA subblock D, (e) OTA subblock C10, C11, R21, R22 Layout, (f) OTA subblock R23, R24.

3.3. The top level of VGA:

This is the last block, and it includes two subblocks, VGA transistors and VGA resistors as shown in Fig. 9. The subblocks are generated and added in accumulative manner. The generated layouts of these subblocks are presented in Fig. 10. The final VGA automated layout is given in detail in Fig 10(b).

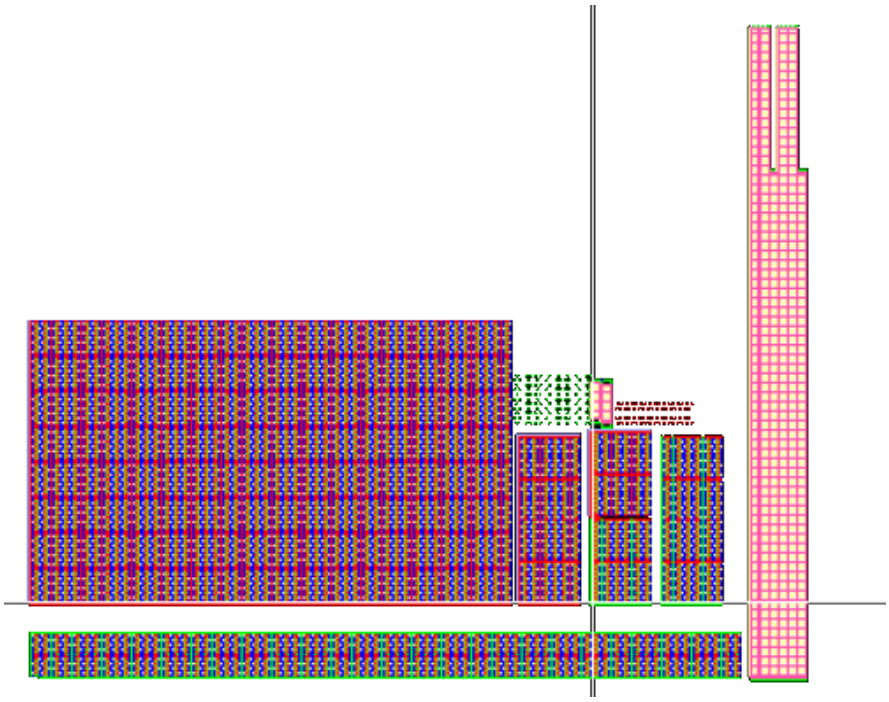


(a)

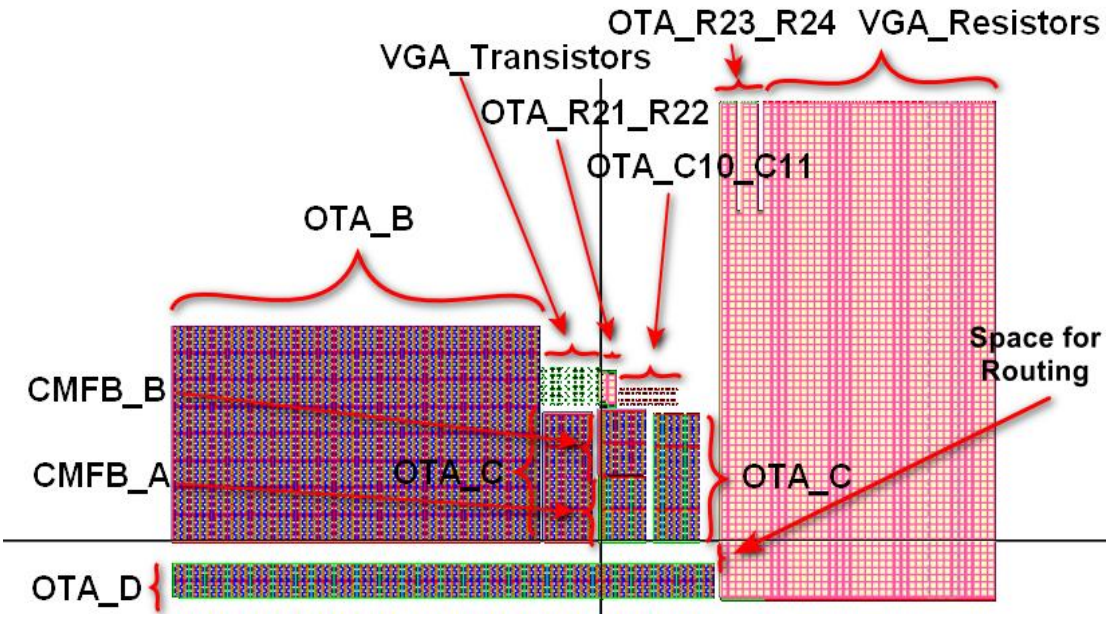


(b)

Fig. 9. Schematic of the top-level VGA two subblocks, (a) VGA transistors, (b) VGA resistors.



(a)



(b)

Fig. 10. VGA automated subblocks layouts, (a) VGA transistors, (b) VGA resistors and the final automated VGA layout.

4. CONCLUSION

In this paper, an effective method of generating an automated layout of a VGA circuit design is presented. The method is based on building code scripts of TCL language. The features of this language are discussed. To obtain the automated VGA layout, the VGA circuit is divided into three main blocks, each block is divided into groups of subblocks. A TCL code is written to generate each subblocks. The subblocks are added in accumulative manner to provide the final automated VGA layout. The final automated VGA layout is provided in detail. From the resulting layout, the TCL language can be considered one of the most powerful languages that can be used to automate the circuit layout with high accuracy. As future work, routing between the devices and other blocks, solving design rule checking (DRC), layout versus schematic (LVS) errors and extracting parasitics using TCL commands can be studied.

References

[1] O. Bonnaud and L. Fesquet, "Microelectronics at the heart of the digital society: technological and training challenges," 2019 34th Symposium on Microelectronics Technology and Devices (SBMicro), Sao Paulo, Brazil, 2019, pp. 1-4, doi: 10.1109/SBMicro.2019.8919405.

[2] Burgess, M., "What Is the Internet of Things?" WIRED Magazine, Accessed on: Nov. 10, 2023. [Online]. https://www.wired.co.uk/article/internet-of-things-what-is-explained-iot.

[3] K. Cheval et al., "Progress in the manufacturing of molded interconnected devices by 3D Microcontact Printing", *Advanced Materials Research*, vol. 1038, pp. 57-60, 2014.

[4] A. Hastings, "The Art of Analog Layout", Second Edition, *Pearson Prentice Hall*, New Jersey, 2nd ed. 2006, ISBN: 978-0-13-146410-0.

[5] Y. P. Xu, "A two-semester project-based mixed-signal IC design course using commercial EDA tools - from design to chip evaluation," in IEEE International Conference on Microelectronic Systems Education (MSE), IEEE Computer Society. IEEE, pp. 25–26, June 2003.

[6] Jef Rijmenants / James B. Litsios / Thomas R. Schwarz / Marc G. R. Degrauwe, "ILAC: An Automated Layout Tool for Analog CMOS Circuits", *IEEE Journal of Solid-State Circuits*, vol. 24, no. 2, pp. 417–425, Apr. 1989, DOI: 10.1109/4.18603.

[7] N. A. Sherwani, "Algorithms for VLSI Physical Design," in *Springer Science & Business Media*, Dec 6 ,2012.

[8] K. L. Kishore, "Trends in VLSI technology: rural application perspective", *IETE Technical Review*, Vol. 24, No.4, pp. 243-248, , July-August 2007.

[9] L. Paris / G. Berbel / T. Osés, "Floorplanning Strategy for Mixed Analog-Digital VLSI Integrated Circuits", Proc. of European Conference on Design Automation, pp. 346–350, Feb. 1991, DOI: 10.1109/EDAC.1991.206422.

[10] N. Jangkrajarng, L. Zhang , S. Bhattacharya, et al. „Template-based parasitic-aware optimization and retargeting of analog and rf integrated circuit layouts," Proc ICCAD, pp. 342-348, 2006.

[11] H Chen, M J Liu, X Y Tang, K R Zhu, N Sun, and David Z. Pan, "Challenges and opportunities toward fully automated analog layout design." in *J. Semicond.*, 2020, 41(11), 111407. http://doi.org/10.1088/1674-4926/41/11/111407.

[12] Accessed on: Nov. 10, 2023. [Online]. Available: https://www.tutorialspoint.com/TCL-tk/TCL_overview.htm.

APPENDIX:

1- TCL code of generating the CMFB subblocks

CMFB subblock A.

```
set libName SAED_PDK_32_28
set cellName nmos4t
set design ed
set rows 2
set cols 6
set dx 0.804
set dy 3.25
set params [list {w 3u} {l 0.5u} {nf 1} {name "CMFB_A"} ]
set CMFB_A [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {0 0} -
rows $rows -cols $cols -dx $dx dy $dy -params $params]
```

CMFB subblock B.

```
set libName SAED_PDK_32_28
set cellName pmos4t
set design ed
set rows 2
set cols 6
set dx 0.804
set dy 3.25
set params [list {w 3u} {l 0.5u} {nf 1} {name "CMFB_B"} ]
set CMFB_B [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {0
6.91} -rows $rows -cols $cols -dx $dx dy $dy -params $params]
```

2- TCL code of generating the OTA subblocks

OTA subblock A

```
set libName SAED_PDK_32_28
set cellName nmos4t
set design ed
set cols 6
set rows 4
set dx 0.804
set dy 3.25
set params [list {w 3u} {l 0.5u} {nf 1} {dummy 8} {name "OTA_A"} ]
set OTA_A [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {5.727
0} -cols $cols -rows $rows -dx $dx dy $dy -params $params]
```

OTA subblock C

```
set libName SAED_PDK_32_28
set cellName pmos4t
set design ed
set cols 6
set rows 4
set dx 0.804
set dy 3.25
set params [list (w 3u} {l 0.5u} {nf 1} {dummy 8} {name "OTA_C"} ]
set OTA_C [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {-5.727
0} -cols $cols -rows $rows -dx $dx dy $dy -params $params]
```

OTA subblock B

```
set libName SAED_PDK_32_28
set cellName pmos4t
set design ed
set cols 48
set rows 8
set dx 0.804
set dy 2.75
set params [list  {w 2.5u} {l 0.5u} {nf 1} {dummy 16} {name "OTA_B"} ]
set OTA_B [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {-45.039
0} -cols $cols -rows $rows -dx $dx -dy $dy -params $params]
```

OTA subblock D

```
set libName SAED_PDK_32_28
set cellName nmos4t
set design ed
set cols 71
set rows 2
set dx 0.804
```

```
set dy 1.75
set params [list {w 1.5u} {l 0.5u} {nf 1} {dummy 4} {name "OTA_D"}]
set OTA_D [le::createInst libName $libName -cellName $cellName -design [$design] -origin {-45.039
-5.727} -cols $cols -rows $rows -dx $dx dy $dy -params $params ]
```

OTA subblock C10_C11

```
set libName SAED_PDK_32_28
set cellName nmos4t
set design ed
set cols 10
set rows 2
set dx 0.604
set dy 1
set params [list {w 0.3u} {l 0.3u} {nf 1} {dummy 2} {name "OTA_C10_C11"} ]
set OTA_C10_C11 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin
{2 14.094} -cols $cols -rows $rows -dx $dx dy $dy -params $params ]
```

OTA subblock R21_R22

```
set libName SAED_PDK_32_28
set cellName rnpoly_wos
set design ed
set rows 1
set cols 2
set dx 0
set dy 0
set params [list {lf 2.799} {wf 0.4} {nf 1} {r 999} {name "OTA_R21_R22"} ]
set OTA_R21_R22 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin
{0 14.094} cols $cols - rows $rows -params $params ]
```

OTA subblock R23_R24

```
set libName SAED_PDK_32_28
set cellName rnpoly_wos
set design ed
set dx 0
set dy 0
set params [list {name "OTA_R23_R24"} ]
set R23_1 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {12.733 -
5.727} -cols 2 - rows 1 - params {lf 50.438} {wf 0.4} {nf 1} {r 18k}}]
set R23_2 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {14.253 -
5.727} -cols 1 - rows 1 - params {lf 39.126} {wf 0.4} {nf 1} {r 13.963k}}]
set R24_1 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {15.013 -
5.727} -cols 2 - rows 1 - params {{lf 50.438} {wf 0.4} {nf 1} {r 18k}}]
set R24_2 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {16.533 -
5.727} -cols 1 -rows 1 - params {lf 39.126} {wf 0.4} {nf 1} {r 13.963k}}]
```

3- TCL code of generating the top-level VGA subblocks

VGA Transistors subblocks

```
set libName SAED_PDK_32_28
set design ed
set rows 1
set cols 8
set dx 0.804
set dy 3.25
set params [list {w 3.5u} {1 0.03u} {nf 1} {name "VGA_Transistors"} ]
set VGA_Transistors [le::createInst -libName $libName -cellName $cellName -design [$design] -
origin {-6.035 14.094} -rows $rows -cols $cols -dx $dx -dy $dy -params $params ]
```

VGA Resistors subblocks

```
set libName SAED_PDK_32_28
set cellNane rppoly_wos
set design ed
set params [list {lf 50.438} {wf 0.4} {nf 1} {r 18k} {name "VGA_Resistors"}]
set R15 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {17.293 -
5.727} -cols 8 -params $params]
set R14 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {23.373 -
5.727} cols 4 -params $params]
set R13 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {26.413 -
5.727} cols 2 -params $params]
set R12 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {27.933 -
5.727} cols 1 -params $params]
set R16 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {28.693 -
5.727} cols 8 -params $params]
set R17 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {34.773 -
5.727} cols 4 -params $params]
set R18 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {37.813 -
5.727} cols 2 -params $params]
set R19 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {39.333 -
5.727} cols 1 -params $params]
set R2 [le::createInst libName $libName -cellName $cellName -design [$design] -origin {40.093 -
5.727} cols 1 -params $params]
set R3 [le::createInst -libName $libName -cellName $cellName -design [$design] -origin {40.853 -
5.727} -cols 1 -params $params]
```