Fast and Easy Open-Source Stick Diagram Validation with Stixu

1st Nicholas Overacker
Info. & Communication Engineering
Kitami Institute of Technology
Kitami, Japan
m3235380013@std.kitami-it.ac.jp

2nd James Stine
Electrical & Computer Engineering
Oklahoma State University
Stillwater, OK
james.stine@okstate.edu

3rd Michal Ptaszynski
Info. & Communication Engineering
Kitami Institute of Technology
Kitami, Japan
ptaszynski@ieee.org

4th Shingo Yoshizawa
Info. & Communication Engineering
Kitami Institute of Technology
Kitami, Japan
yosizawa@mail.kitami-it.ac.jp

5th Miho Kobayashi
Independent UI Designer
Nagano, Japan
kobayashi@ringodesigners.com

I. Introduction

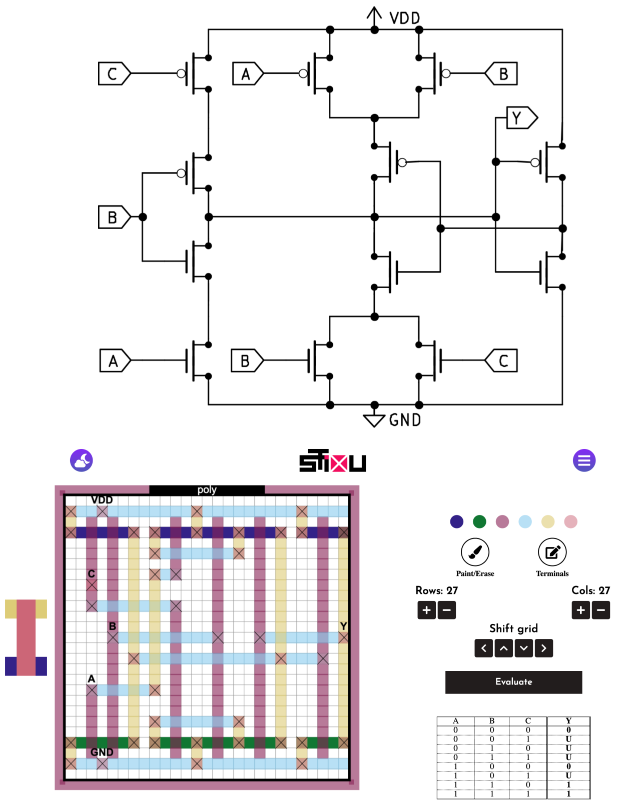
Stixu[1] is an open-source analysis and validation tool for topological stick diagrams of VLSI circuit elements. [1] In the traditional VLSI design flow, stick diagrams are drawn by hand on paper and used as the basis for physical layout. Stick diagrams superficially resemble physical layouts that one might create in software such as Magic VLSI, but they differ significantly and fundamentally from physical layouts in that they only model topology rather than actual dimensions, technology-specific details, and analog behavior. Because of this, stick diagrams provide a simple, easy, and fast model for a level of abstraction appropriate for back-of-the-envelope design work.

In spite of the simplicity and ubiquity of stick diagrams, however, there seem to be no publicly-available validation tools for stick diagrams as such. In general, the only way to validate a topological stick diagram without Stixu is visual inspection followed by manual implementation and simulation of a physical layout based on the diagram. Our work bridges this gap by offering a tool in which users can draw stick diagrams on a PC, tablet, or smartphone as quickly as on paper and then perform static digital output analysis within the same tool with the press of a single button. In this demo session, we will present the basic underlying model employed by Stixu and show attendees how to use the tool, how to interpret its analysis output, and how to export code for further event-based analysis in traditional hardware simulators.

II. How it Works: The Hypergraph Model

Our static analysis algorithm creates a new hypergraph for each input vector tested. For example, a 2-input diagram will generate 4 hypergraphs, with separate hypergraphs for the input vectors 00, 01, 10, and 11. The generation of new graphs

[1]Available online at https://stixu.io. The source code repository is included at the end of this paper.



A	B	C	Y
0	0	0	**0**
0	0	1	**U**
0	1	0	**U**
0	1	1	**U**
1	0	0	**0**
1	0	1	**U**
1	1	0	**1**
1	1	1	**1**

Fig. 1: A CMOS schematic (drawn in KiCad for reference) and stick diagram (drawn in Stixu) of a generalized static Muller C-element [2, p. 100] and its output table as computed by our static analysis algorithm. (Test case 64) On the left of the stick diagram is a layer visualizaiton for the individual highlighted cell in row 4 column 27, showing the user at a glance that the highlighted cell contains the metal-2 layer, the p-diffusion layer, and a contact.

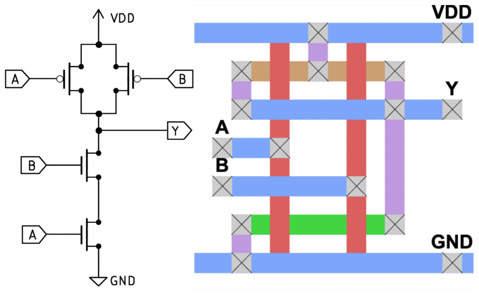


Fig. 2: A 2-input NAND gate represented as a CMOS schematic (drawn in KiCad) and a stick diagram (drawn in our tool).

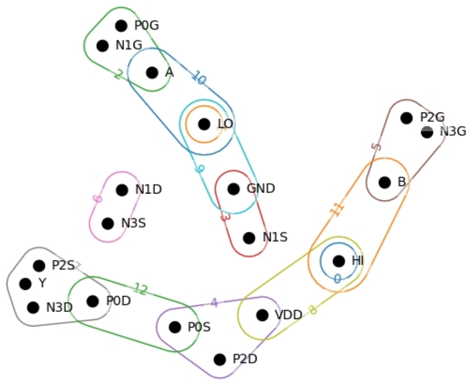


Fig. 3: The hypergraph generated for the stick diagram in Fig. 2 given inputs (A,B)=(0,1). Vertices whose labels begin with "P" or "N" represent the source, drain, or gate terminals of a PMOS or NMOS transistor, respectively. There is a path from **Y** to **HI**, so the output for this input is 1.

for each input is necessary because the connections between vertices in our model depend on the state of each transistor, which in turn depends on input values.

Figures 2, 3, 4 illustrate our hypergraph model applied to a 2-input NAND gate[2] In Figure 3, the inputs **A** and **B** are set to **0** and **1**, respectively. This is represented in the hypergraph by a loop (hyperedge) connecting the **A** and **LO** vertices to each other. Because this is a NAND gate and the input **A** is set to **0**, there is a path from the **HI** vertex to the **Y** vertex. This indicates that the output of the circuit for this set of inputs is **1**.

In Figure 4, both **A** and **B** are set to **1**. As a result, there are hyperedges connecting both input vertices to the **HI** vertex. Furthermore, there is a path from **LO** to **Y**, indicating that the output for this set of inputs is **0**.

III. Static Evaluation

For each input vector, the output can be any of the following values:

- **X**: Path exists from **LO** to **HI** anywhere in the circuit, or such a path can be made through a transistor with an undriven gate.

[2]Hypergraph visualizations were produced with HyperNetX. [3]

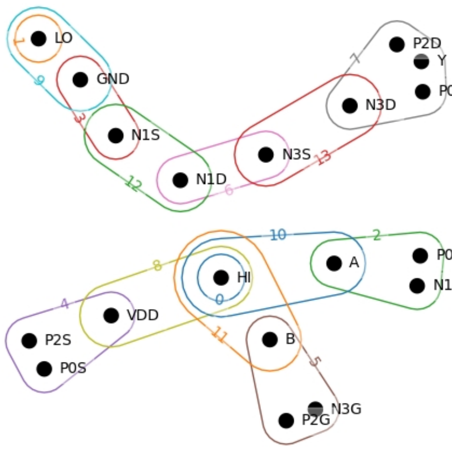


Fig. 4: The hypergraph generated for the stick diagram in Fig. 2 given inputs (A,B)=(1,1). There is a path from **Y** to **LO**, so the output for this input is 0.

- **Z**: Floating; no possible path from **HI** or **LO** to the output.
- **0**: Path from **LO** to the output.
- **1**: Path from **HI** to the output.
- **L**: Floating, but possible path to **LO** through transistors with undriven gates.
- **H**: Floating, but possible path to **HI** through transistors with undriven gates.
- **U**: Floating, but possible path to either **LO** or **HI** through transistors with undriven gates, and no possible path from **LO** to **HI**. These are often but not always latched values.

IV. Code Generation and Event-based Evaluation

Because static analysis is insufficient for evaluating sequential circuits, our tool uses the hypergraph representation of stick diagrams to generate Verilog code that can be used for event-based analysis of stick diagrams. In principle, this can and will be easily adapted to support the generation of other formats, such as SPICE and IRSIM files.

In the demo, we will demonstrate how to use this feature and how to extend it.

V. Source Code

The source code for Stixu is available at the following address: https://github.com/NickOveracker/StickDiagrammer

References

[1] N. Overacker, J. Stine, M. Ptaszynski, S. Yoshizawa, and M. Kobayashi, "Stixu: An open-source idea for planning integrated circuit layouts efficiently," in *TENCON 2024 - 2024 IEEE Region 10 Conference (TENCON)*, 2024, pp. 782–785.

[2] J. Sparsø, "Asynchronous circuit design - a tutorial," 2001. [Online]. Available: https://api.semanticscholar.org/CorpusID:67482001

[3] B. Praggastis, S. Aksoy, D. Arendt, M. Bonicillo, C. Joslyn, E. Purvine, M. Shapiro, and J. Y. Yun, "Hypernetx: A python package for modeling complex network data as hypergraphs," *Journal of Open Source Software*, vol. 9, no. 95, p. 6016, 2024. [Online]. Available: https://doi.org/10.21105/joss.06016