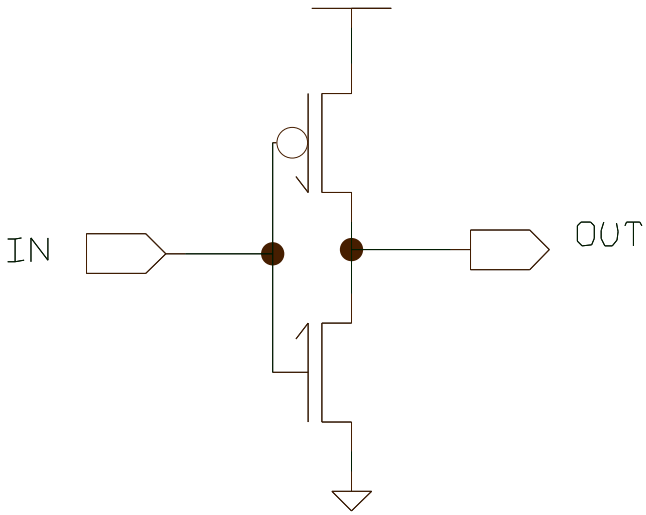
zDraw - schematic editor

that came back from the cold.

Many years ago, before synthesis became useful, the VLSI design was entered as schematics of gates and transistors. I still have and cherish Daisy logo somewhere.

Then synthesis arrived and need for schematics was reduced to analog circuits only. At that time i wrote my schematic editor.

Commercial and free editors are always loaded with **environments** and other junk. I wanted just schematic editor, and *zDraw* is just that, without any baggage. I would call it "VI" of schematics editors. Among others things, it has a feature that the file is humanly readable and sometimes writable. And unlike commercial editors, no license, no setup, no libraries, no nothing. You get in, draw something, curse the bugs and their creator, get the job done and get out.



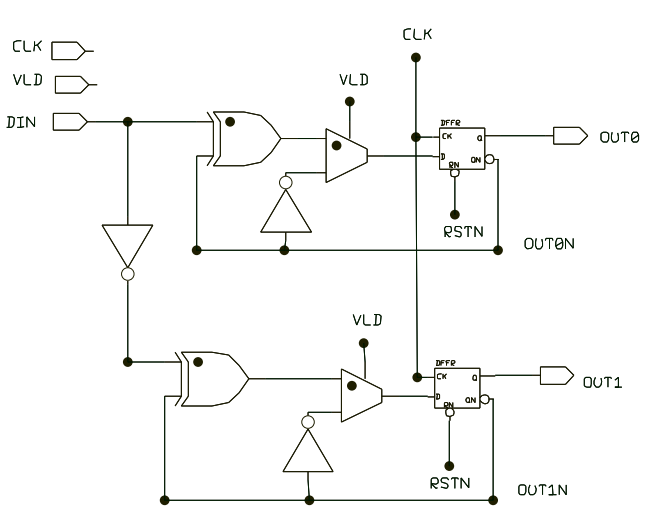
For me, it's main usage was to quickly create Spice circuits to try out things.
Side use was to assist documentation. It is much easier to create and maintain diagrams of circuits with schematic editor.

The original version was written in Prolog. Another dinosaur. Over the years, i rewrote it into Python, and used it from time to time. Like when we had problems with IO cells behaviour, and to make sense of it, i read LVS spice and created an accurate schematic. For most part, zDraw code was gathering dust and bugs in the cloud.

But something changed quite recently. Companies like documentation. Some more than others.
One company wants exact diagrams of some critical logic regions. Things like resets and clocks management.
They want to review the picture, instead of a thousand verilog words.
It makes sense, but keeping the two in sync is a chore/ burden/ grind.
Frequent changes in logic (RTL) and Documentation caused ping-ponging between the two and frustration.
I tried to use DOT. DOT (GraphViz) extracted by script from RTL. It proved to be doable but ugly.

Then i remembered zDraw.
I came to realize that some RTLs are simple and can be expressed as a limited-complexity schematic.
Schematic nice enough to serve as documentation and accurate enough to serve as source for RTL.

So the zDraw code was brushed up (updated in gitHub). Here is an example of how it looks like.
this example has no connection to any company.



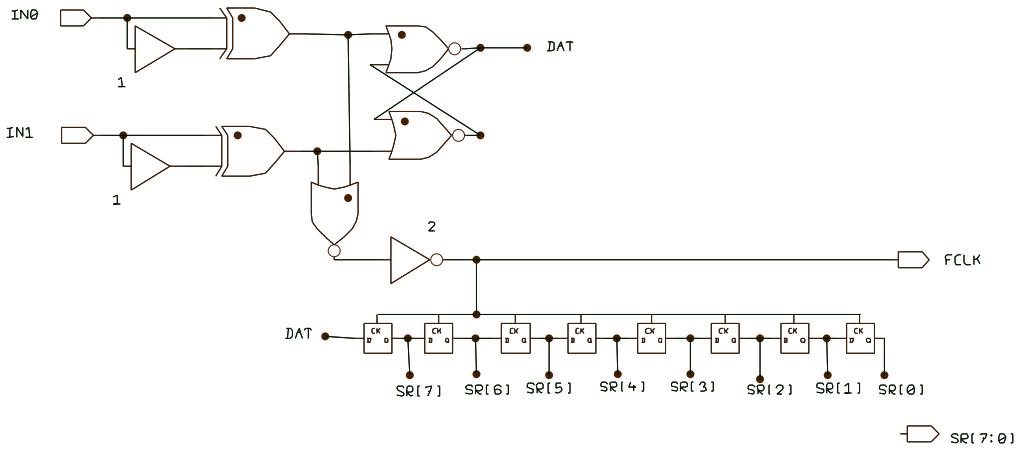
The rtl code created by zDraw:

```
module spacewiretx (   input din  ,input vld, input clk ,output out0 ,output out1);
reg dffr_0; assign out1 = dffr_0;
always @(posedge clk or negedge rstn) if (!rstn) dffr_0 <= 0; else dffr_0 <=
```

```
wire43;
reg dffr_1; assign out0 = dffr_1;
always @(posedge clk or negedge rstn) if (!rstn) dffr_1 <= 0; else dffr_1 <=
wire33;
assign wire37 = din ^ out0n;
assign wire48 = wire25 ^ out1n;
assign wire33 = vld ? wire38 : wire37;
assign wire38 = ~out0n;
assign wire47 = ~out1n;
assign wire43 = vld ? wire47 : wire48;
assign wire25 = ~din;
endmodule
```

I might spend some time to improve the readability. But the main point is , that after we review this schematic, i don't have to worry about coherency.

Here is another example:



And corresponding RTL (also in need of tidying up):

```
module spacewirerx (   input in0 ,input in1 ,output fclk ,output sr[7:0]);
assign #1 wire1 = in0;
assign wire9 = in0 ^ wire1;
assign wire10 = in1 ^ wire5;
assign #1 wire5 = in1;
assign wire13 = ~(dat | wire10);
assign dat = ~(wire9 | wire13);
reg flop_0; assign sr[7] = flop_0;
always @(posedge wire24) flop_0 <= dat;
```

```
reg flop_1; assign sr[6] = flop_1;
always @(posedge wire25) flop_1 <= sr[7];
reg flop_2; assign sr[5] = flop_2;
always @(posedge wire25) flop_2 <= sr[6];
reg flop_3; assign sr[4] = flop_3;
always @(posedge wire27) flop_3 <= sr[5];
reg flop_4; assign sr[3] = flop_4;
always @(posedge wire28) flop_4 <= sr[4];
reg flop_5; assign sr[2] = flop_5;
always @(posedge wire29) flop_5 <= sr[3];
reg flop_6; assign sr[1] = flop_6;
always @(posedge wire30) flop_6 <= sr[2];
reg flop_7; assign sr[0] = flop_7;
always @(posedge wire30) flop_7 <= sr[1];
assign wire35 = ~(wire9 | wire10);
assign #2 fclk = ~wire35;
endmodule
```

Do You think it is a crazy idea? I mean, turntables and vinyls are making comeback.
So far the feedback i am getting is clear: *We dont want to trade Teslas for Horses.*
On the other hand, some people think SV/UVM is a progress over Python for verification.

But would You agree that some critical logic "clusters" can actually benefit from this approach?

P.S the examples of spacewire were not debugged!