ECE327: Digital Systems VLSI

Semester Project Part A

Design, Characterization and Use of Standard Cells

Ioanna-Maria Panagou

2962

Fall Semester 2022-2023

Contents

1 Summary

In this part of the project, we were tasked with designing and testing 4 modules: an XOR gate, an XNOR gate, a positive edge-triggered D flip-lop and a D flip-flop with asynchronous active-high preset and clear.

2 XOR/XNOR

2.1 XOR gate

The truth table for the XOR function ($F = A^{'}B + AB^{'}$) is presented in table 1. The gate can be constructed in two different ways. The first implementation can easily be derived by directly observing the truth table. For $AB = \{00, 11\}$, the output is driven low by the pull down network. Therefore, the pull down network must consist of two different paths; one that is on when both A and B are 0 and one when A and B are equal to 1. In the same vein, the pull up network must drive the output to V_{DD}, when $AB = \{01, 10\}$, so we again need two paths that are on when the two inputs differ. This analysis results in the gate of figure 1

A	**B**	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Table 1: XOR gate truth table

The total number of transistors required are 8 for the gate and 4 additional transistors for the inverters of A and B, so 12 transistors in total.

The pull up and pull down networks should have an equvalent resistance of $6.5k\Omega$. We know that for a ratio $\frac{W}{L}$ equal to 1, the NMOS has a resistance of $13k\Omega$. In the pull down network, there are 2 in series transistors, so each one should have a resistance of $\frac{6.5}{2} = 3.25k\Omega$, which is $\frac{13}{3.25} = 4$ times smaller than the NMOS with equal channel width and height. We also know that $R \propto \frac{L}{W}$, so $\frac{L}{W} = \frac{1}{4} \implies \frac{W}{L} = 4$. The minimum size for L is equal to $2\mu m$, so $W_{NMOS} = 8\mu m$ (choosing the minimum value for W would result in L being smaller than the minimum).

The analysis for the pull up network is similar. The PUN also consists of two paths with 2 in series transistors each, so each one should have a resistance of $3.25k\Omega$, which is $\frac{31}{3.25} \approx 9.54 \approx 10$. Therefore, $\frac{W}{L} = 10 \implies W_{PMOS} = 10 \cdot L_{min} = 10 \cdot 2\mu m = 20\mu m$.

Figure 3 presents the circuit graph of 1. The common Euler path for the PUN and PDN is $A^{'} \to B \to A \to B^{'}$. The choice of input for each gate was intentional; we needed to modify our initial circuit to the circuit of figure 2, so that a common Euler path for the PUN and PDN exists. More specifically, we switched the order of the inputs in series, since that would not alter the functionality of the gate.

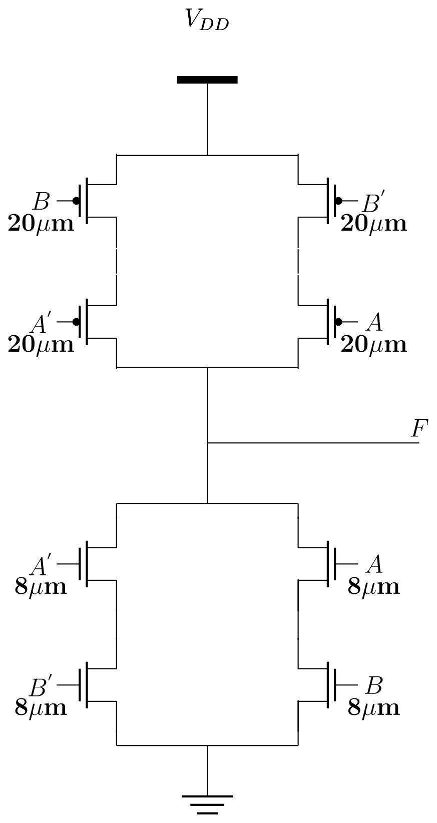

Figure 1: Circuit diagram for XOR gate

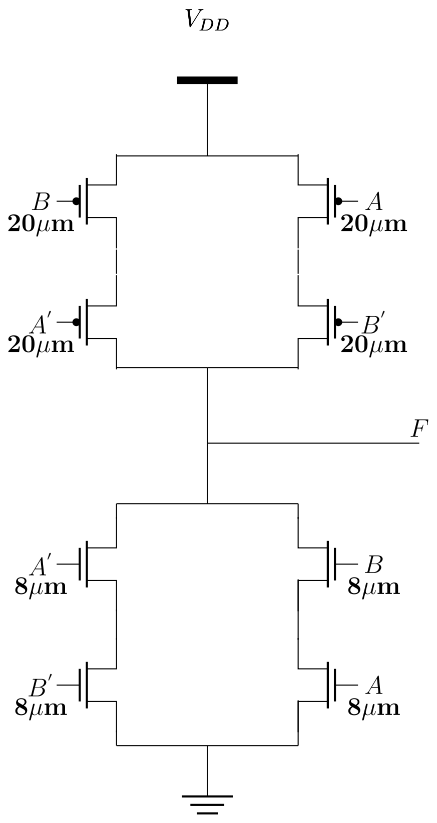

Figure 2: Circuit diagram for XOR gate

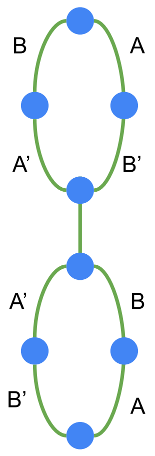

Figure 3: Circuit graph for the XOR gate

Based on the above Euler path, we constructed the stick diagram of 4.

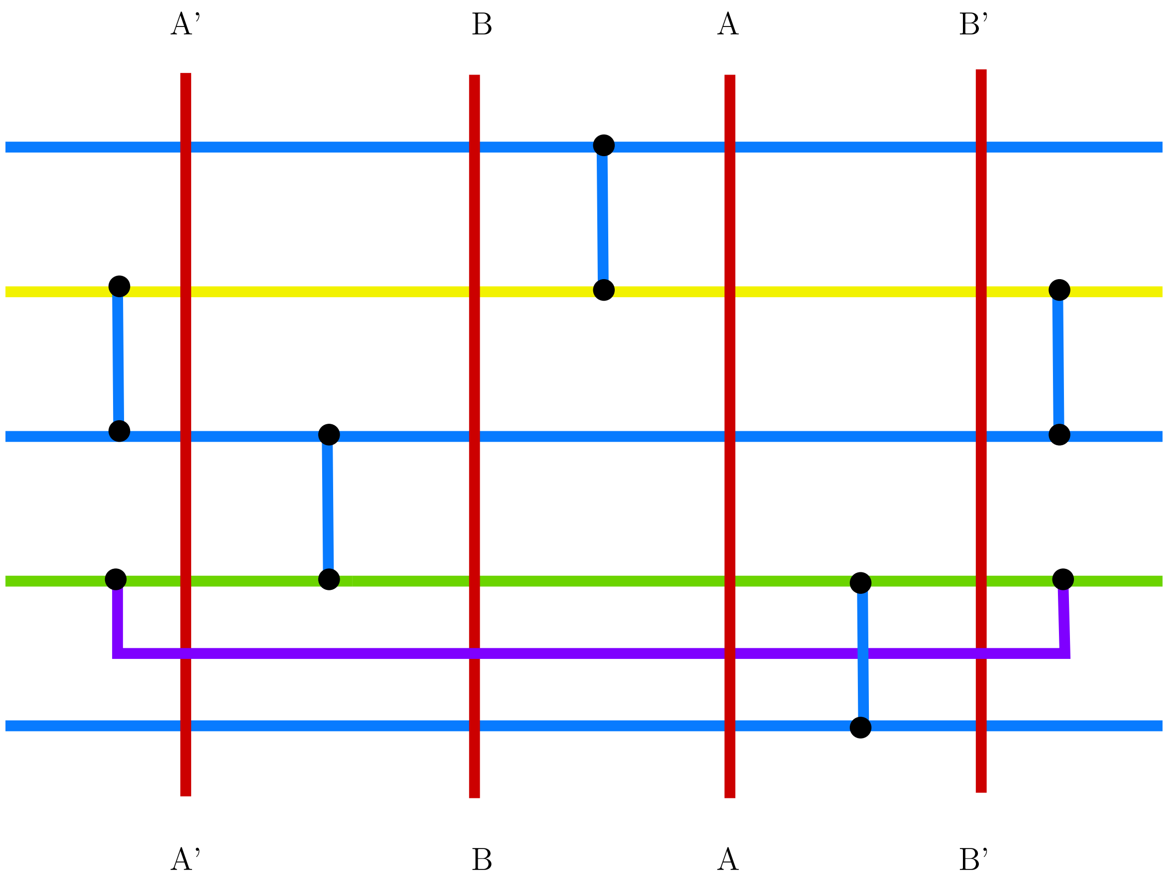

Figure 4: Stick Diagram of XOR gate

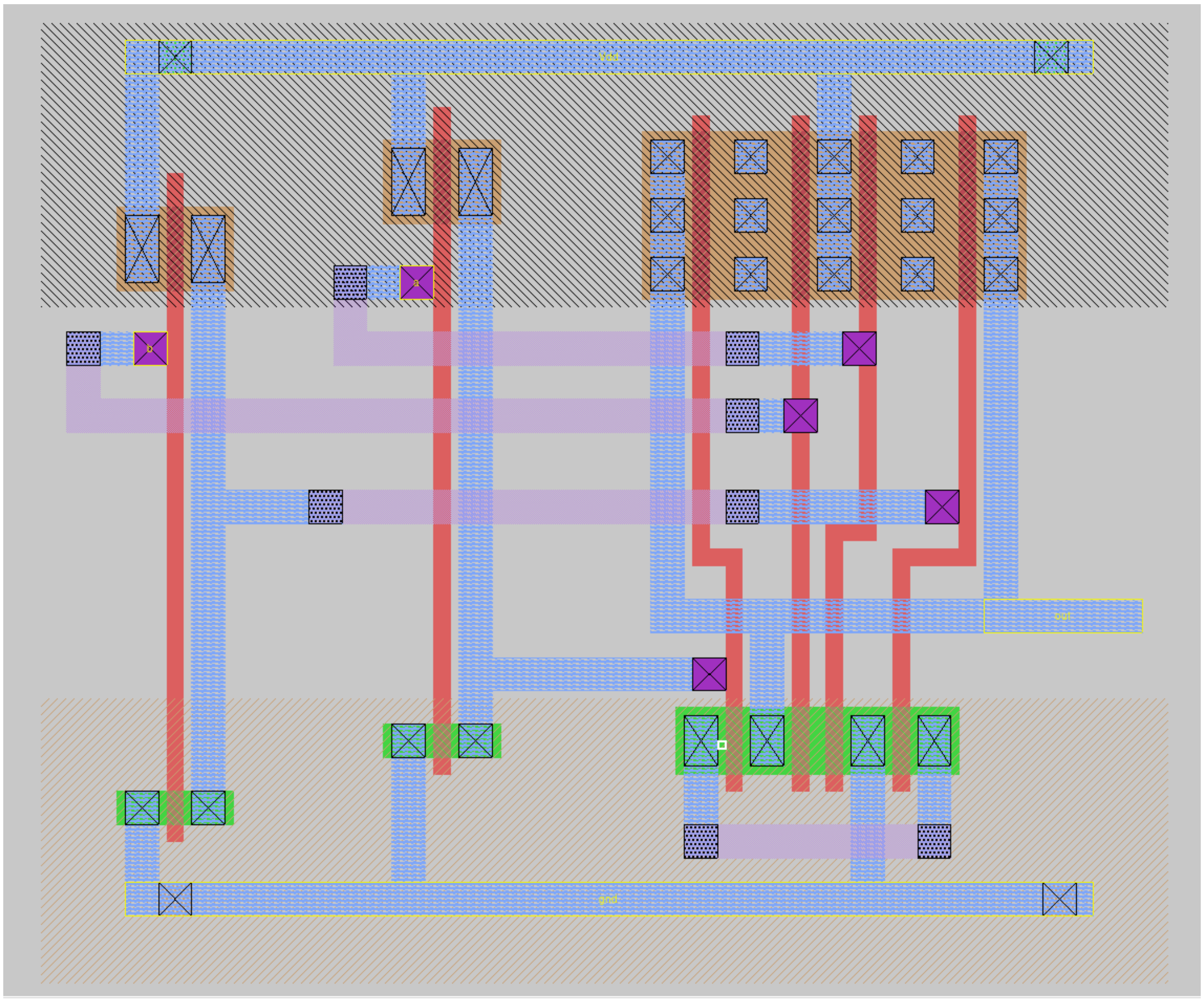

Figure 5: MAGIC Layout for XOR gate

The MAGIC layout of the XOR gate is pictured in figure 5.

Figure 6 verifies the functionality of our gate.

The second implementation of the XOR gate can be derived using DeMorgan's law. We know that static CMOS logic is only used to implement negative gates. We can rewrite $F_{xor} = A'B + AB'$ as $F_{xor} = \overline{(A + B')(A' + B)}$. This expression results in the gate of figure 7. The sizing of the transistors is the same as the previous implementation of the XOR gate, since the PUN is essentially the same and the PDN also has paths with 2 in series transistors. Figures 8, 9, 10, 11 present the circuit graph, the stick diagram, the MAGIC layout and the simulation waveforms of this XOR implementation respectively.

2.2 XNOR gate

The XNOR gate is very similar to the XOR gate. Although we could construct an XNOR gate by adding an inverter at the output of the XOR gate, that would result in a total of 14 transistors. Following the paradigm of the XOR gate, an XNOR gate could be constructed

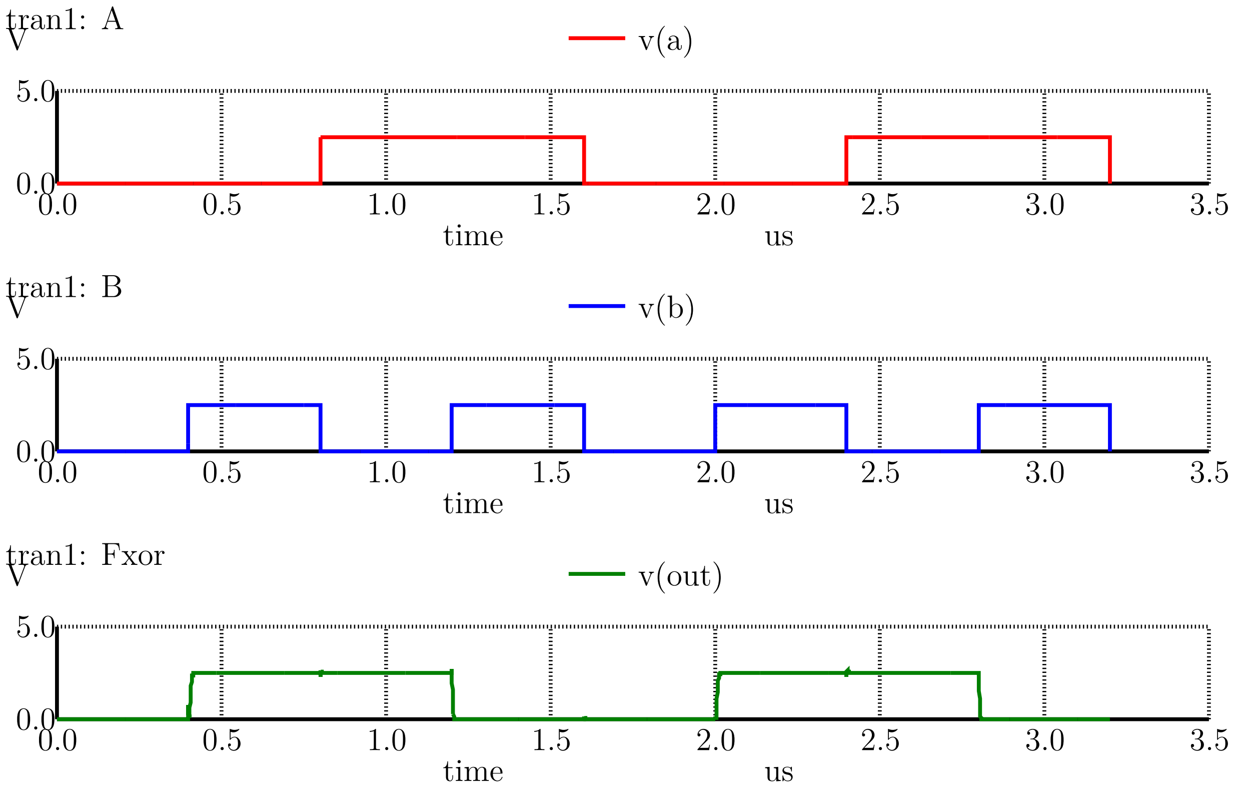

Figure 6: XOR Spice simulation

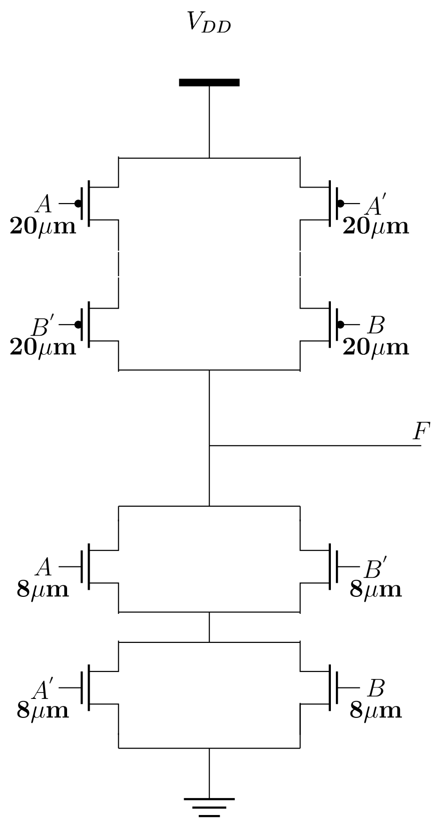

Figure 7: Circuit diagram for XOR gate

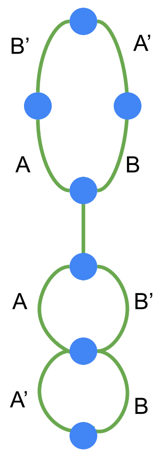

Figure 8: Circuit graph for the XOR gate. The Euler path is $A \rightarrow B^{'} \rightarrow A^{'} \rightarrow B$

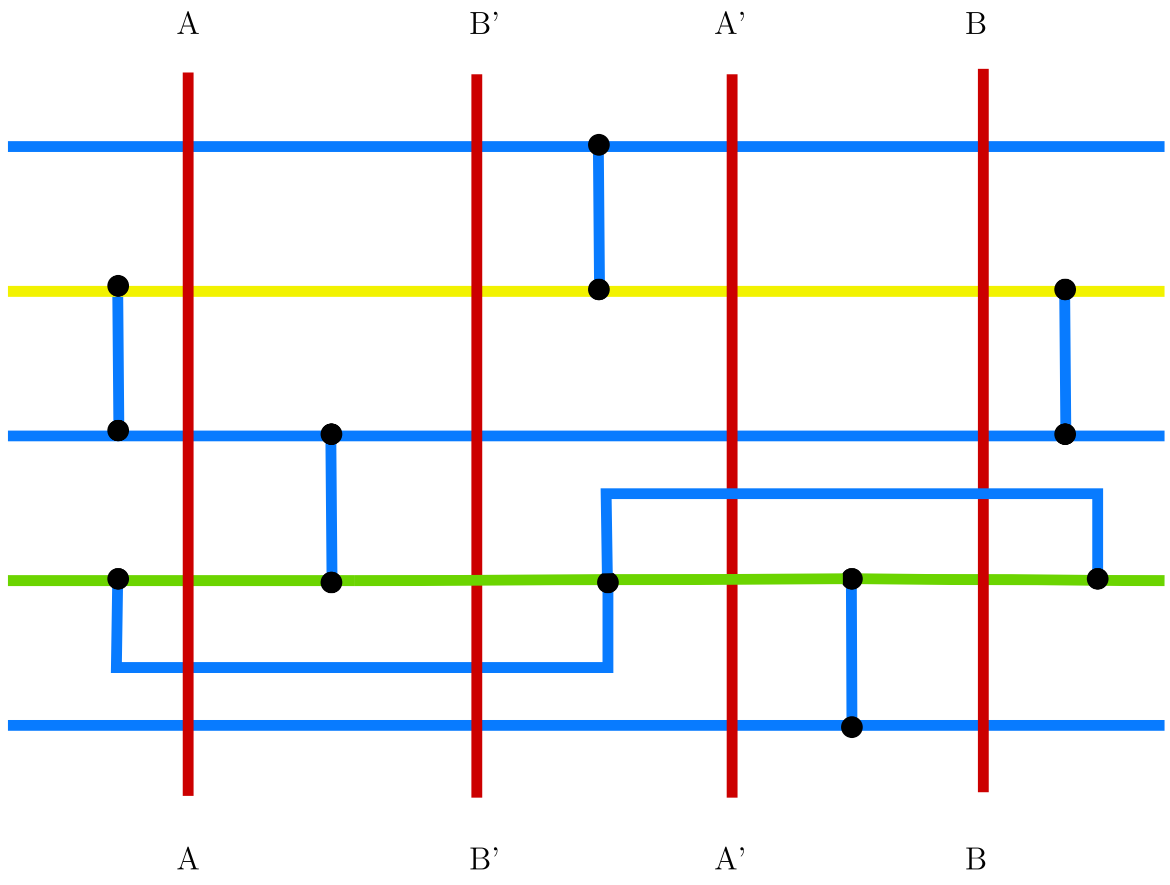

Figure 9: Stick Diagram of XOR gate

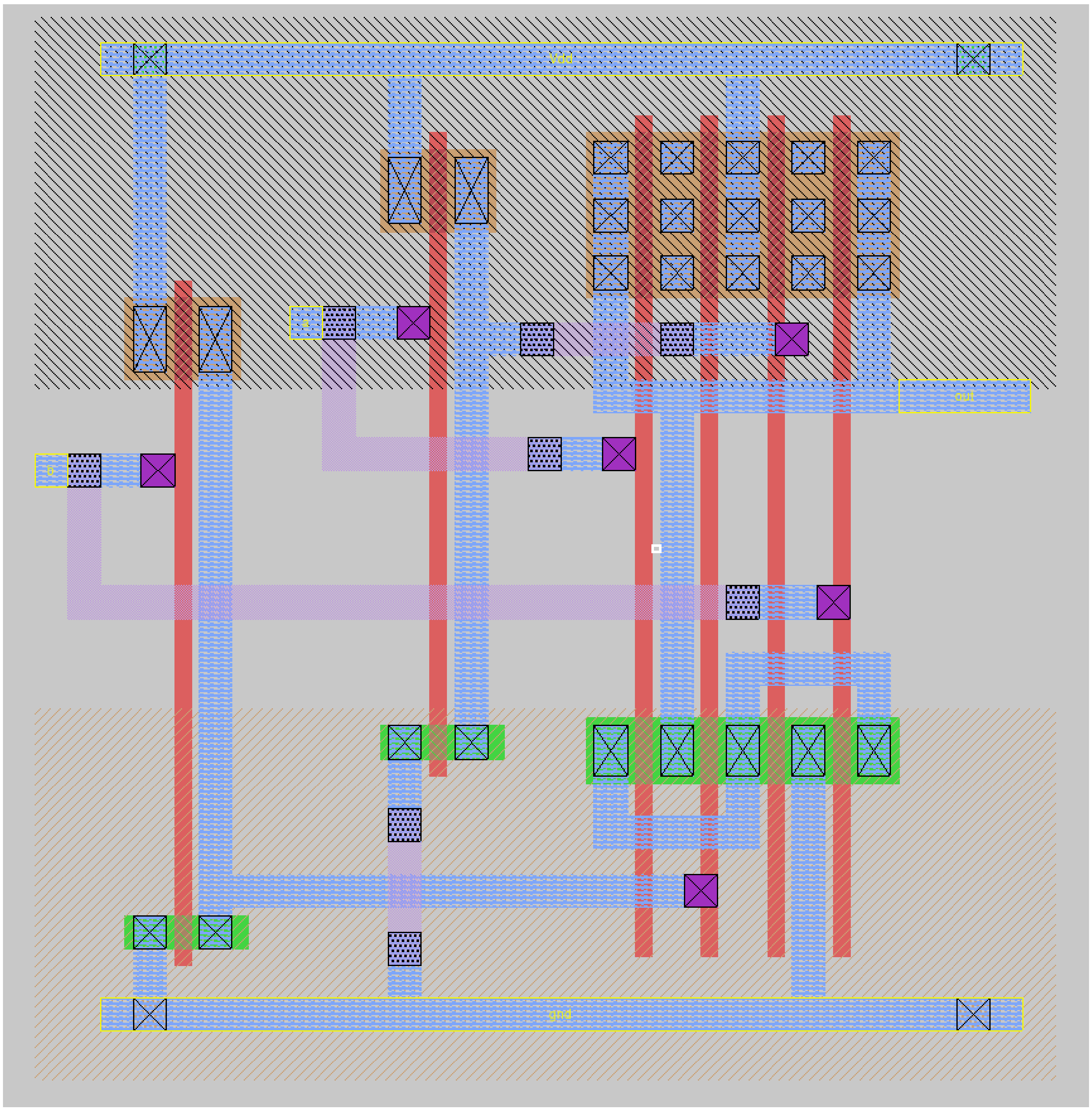

Figure 10: MAGIC Layout for XOR gate

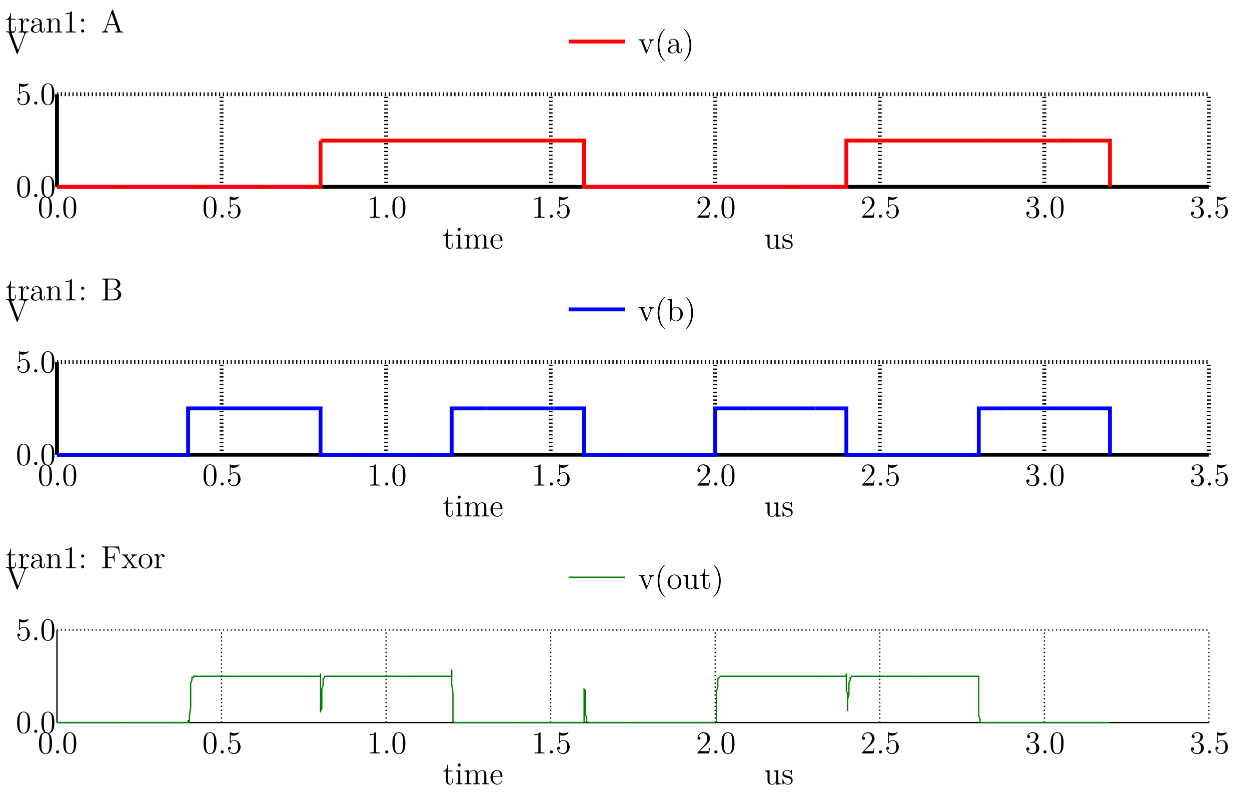

Figure 11: XOR Spice simulation

with just 12 gates. The first implementation stems from directly observing the XNOR truth table of 2. The pull down network should form paths to the ground when the inputs differ, i.e. $AB = \{01, 10\}$ and the pull up network when the inputs are equal i.e. $AB = \{00, 11\}$. The gate that we end up with is pictured in figure 12. The topology of the transistors is the same as the XOR gate, so the sizing remains the same.

A	**B**	**F**
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

Table 2: XNOR gate truth table

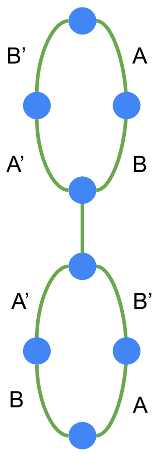

Figure 13: Circuit graph for the XNOR gate. The Euler path is $A \rightarrow B' \rightarrow A' \rightarrow B$

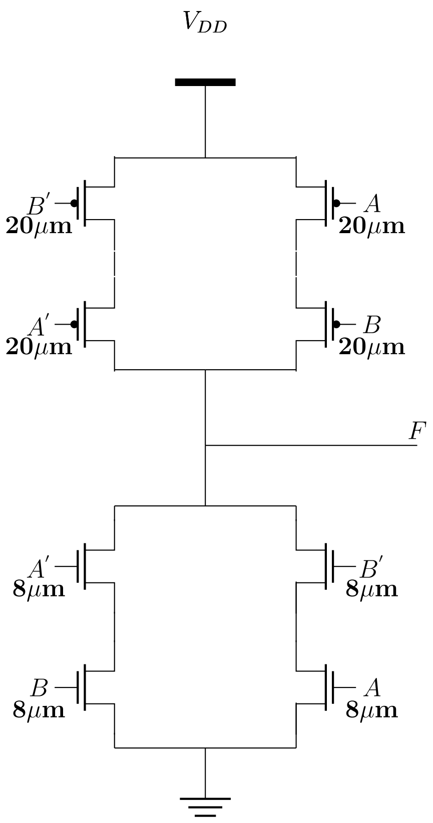

Figure 12: Circuit diagram for XNOR gate

Figures 13, 14, 15 and 16 present the circuit graph, the stick diagram, the MAGIC layout and the SPICE simulation waveforms for the above implementation of the XNOR gate.

The XNOR gate can also be constructed utilizing DeMorgan’s law: $F_{xnor} = AB + A'B' \implies F_{xnor} = \overline{(A' + B')(A + B)}$, as presented in 17, 18, 19, 20 and 21.

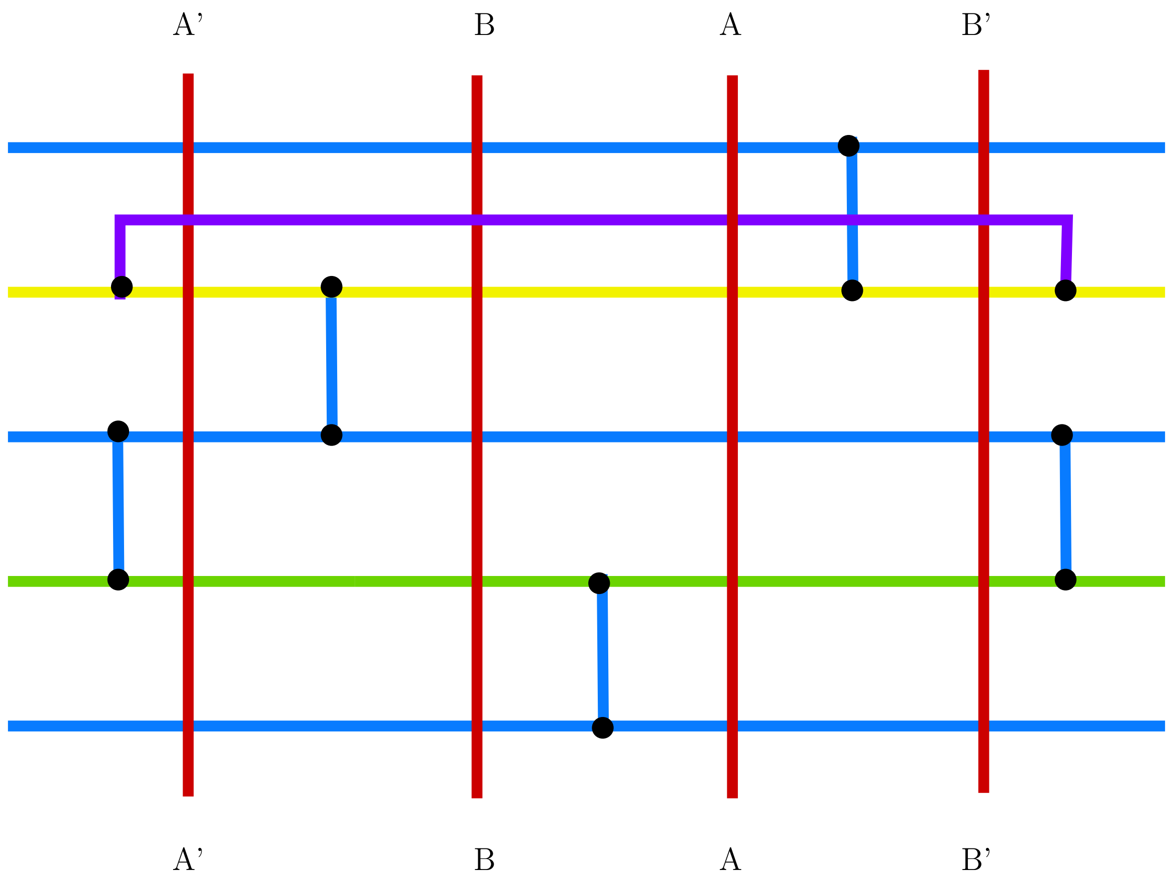

Figure 14: Stick Diagram of XNOR gate

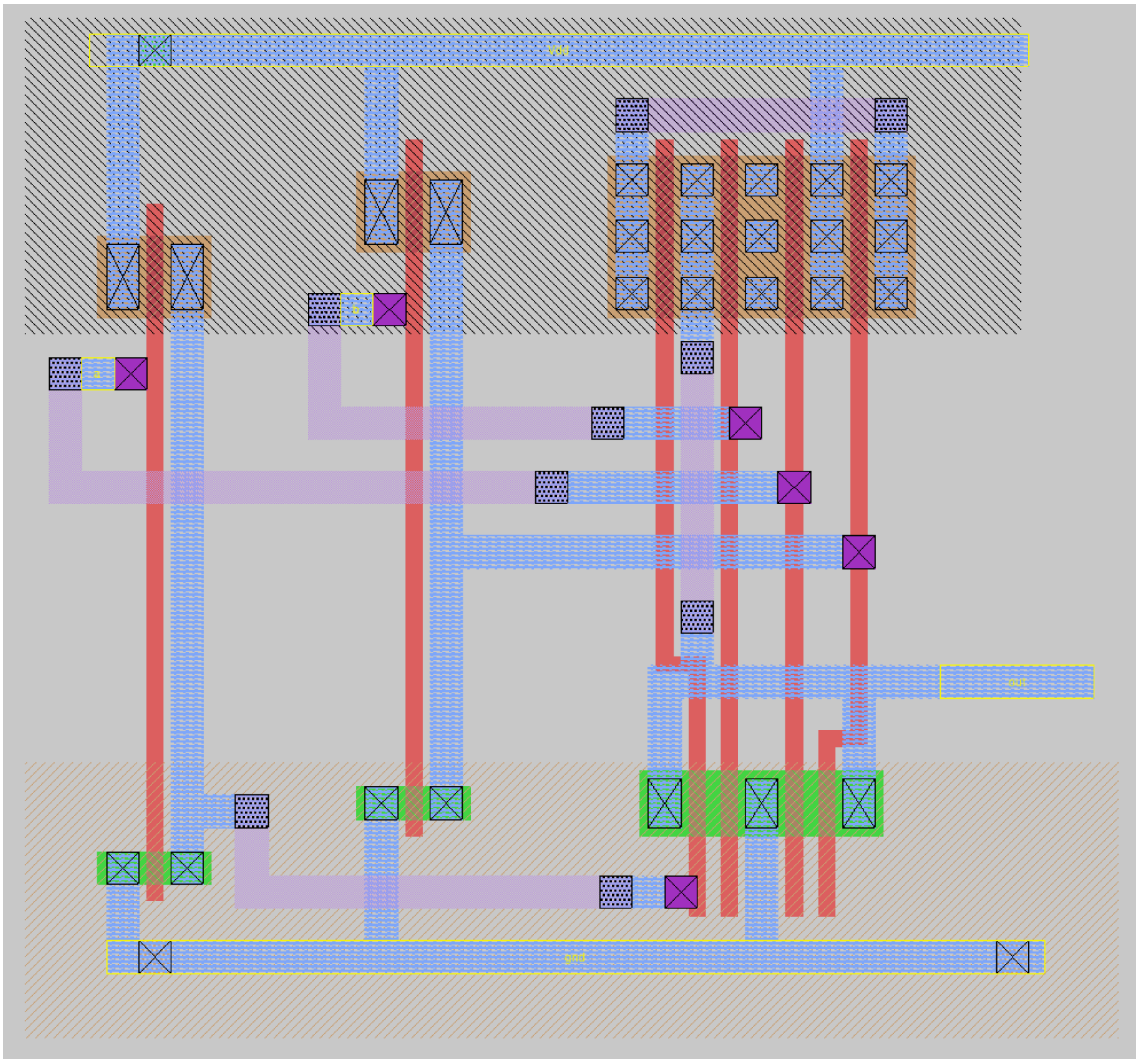

Figure 15: MAGIC Layout for XNOR gate

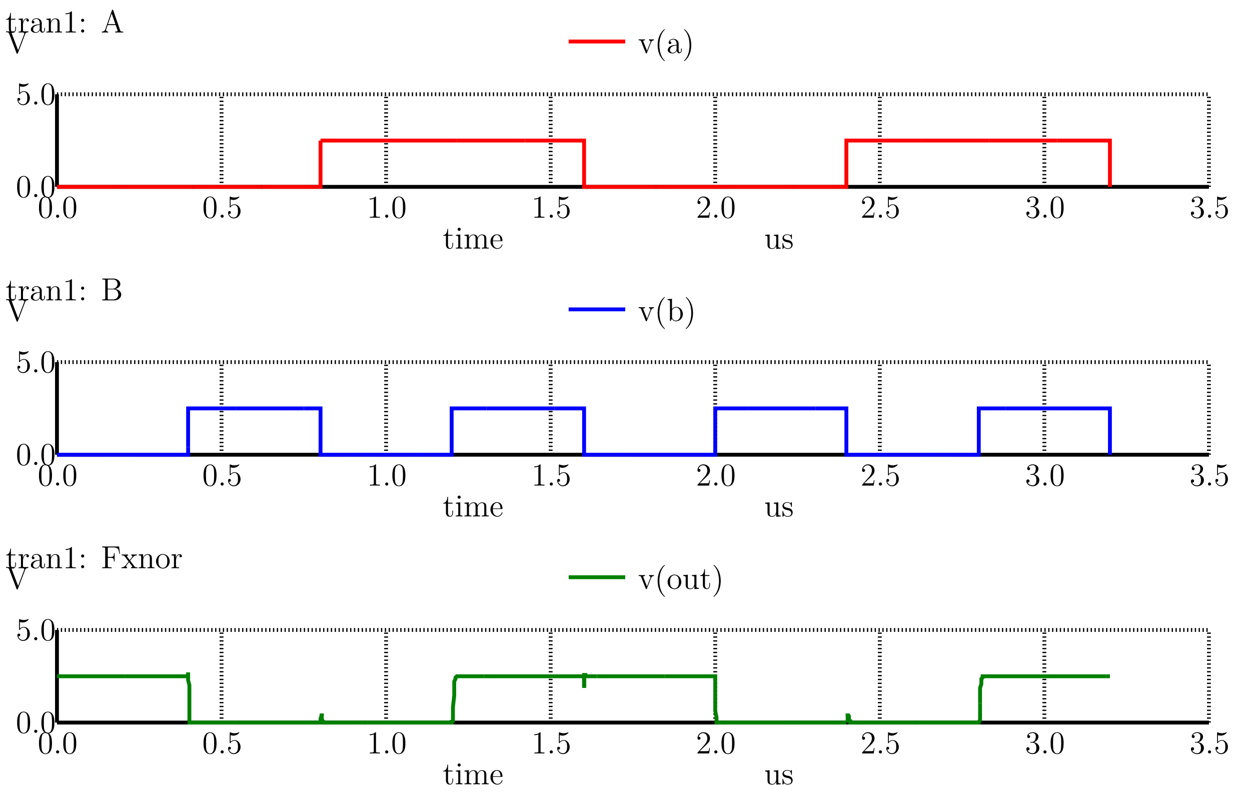

Figure 16: XNOR Spice simulation

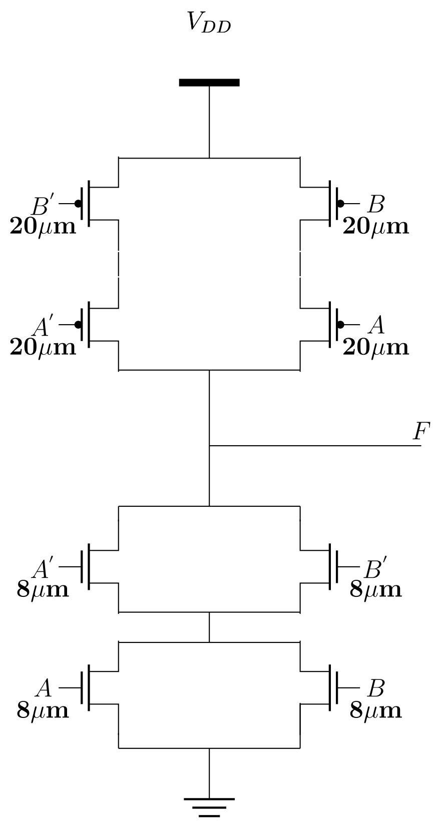

Figure 17: Circuit diagram for XNOR

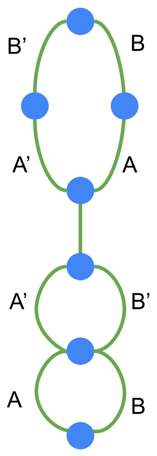

Figure 18: Circuit graph for the XNOR gate. The Euler path is $A \rightarrow B' \rightarrow A' \rightarrow B$

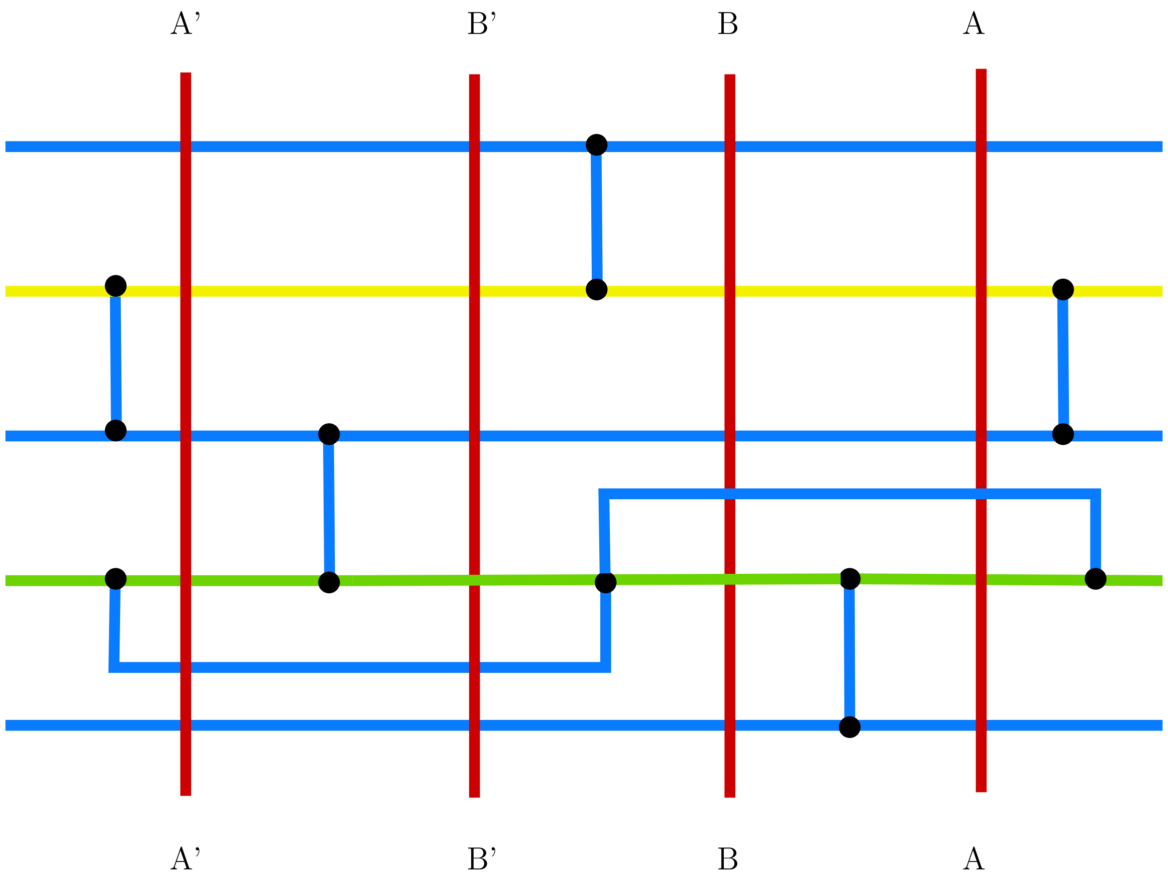

Figure 19: Stick Diagram of XNOR gate

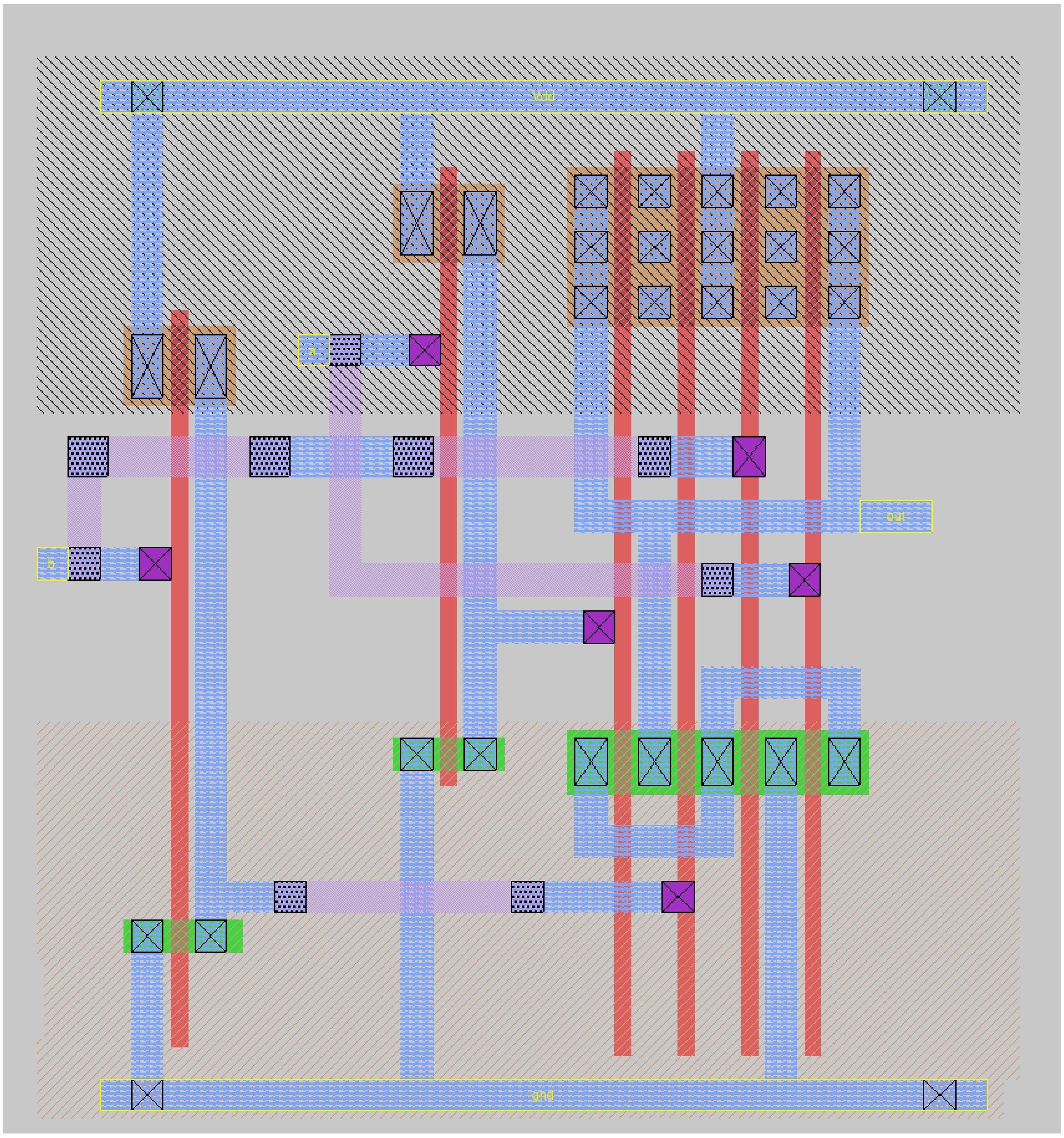

Figure 20: MAGIC Layout for XNOR gate

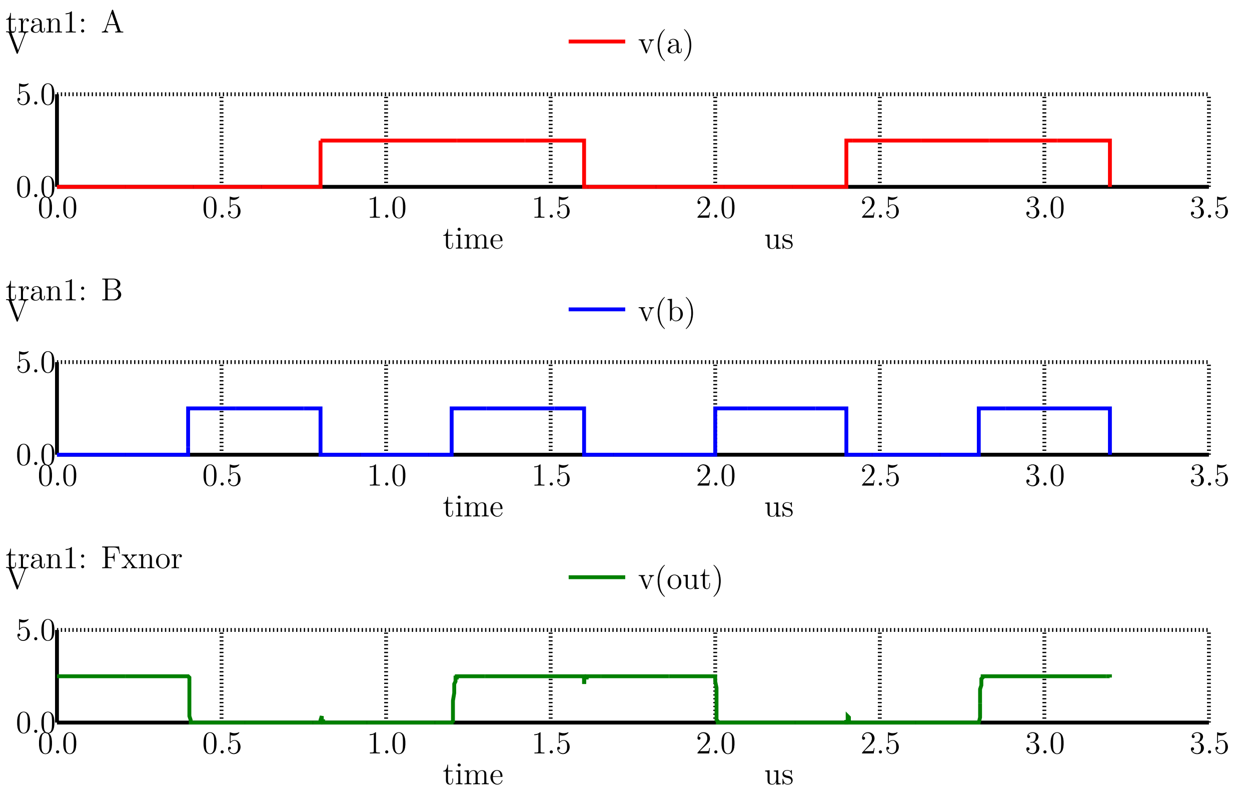

Figure 21: XNOR Spice simulation

3 D Flip-Flop (DFF)

The D flip flop we had to design for this part of the project is a positive-edge triggered flip flop. The architecture we chose for our D flip flop is depicted in figure 22 and requires only 14 transistors (including the 2 transistors needed for the inverter of the output Q). Other design options are presented in figures 23, 24 and 25.

The flip flop of figure 23 is comprised solely of NAND2 gates and inverters. Each NAND gate in static CMOS logic would need 4 transistors, resulting in 32 transistors only for the NAND gates and 4 more for the two inverters, which is not acceptable in terms of area.

The flip flop of figure 24 requires way less transistors than figure 23 (24 transistors), but it has 4 transmission gates ($T_1 - T_4$), so there is a high capacitative load to the clock signal, which is important as it directly impacts the power dissipation of the clock network.

The flip flop of figure 25 has even less transistors than our implementation, since it uses NMOS pass transistors instead of transmission gates. However, the use of only NMOS pass transistors results in the passing of a degraded high voltage of $V_{DD} - V_T$ to the input of the first inverter. This impacts both noise margins and the switching performance. It also causes static power dissipation in the first inverter, since the PMOS device of the inverter is never turned off, resulting in a static current flow.

The penalty we, however, need to pay for the reduced clock load and minimal power dissipation is increased design complexity. If we wish to switch the state of the cross-coupled inverter,

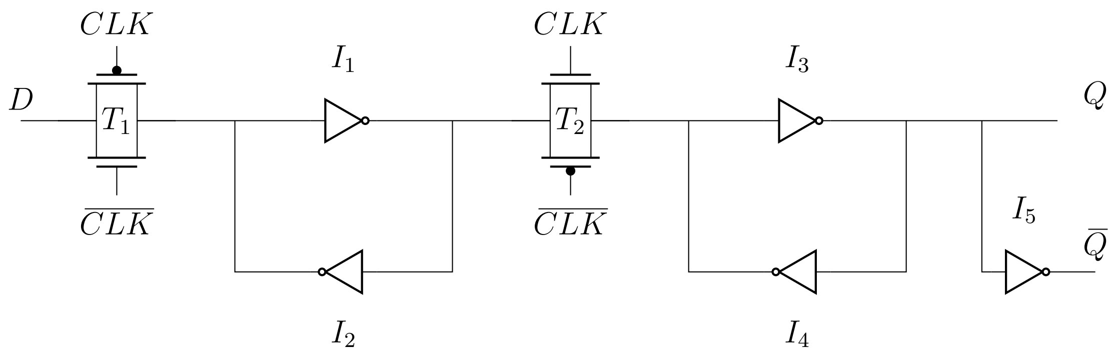

Figure 22: Reduced Load Clock Static Master-Slave Flip Flop

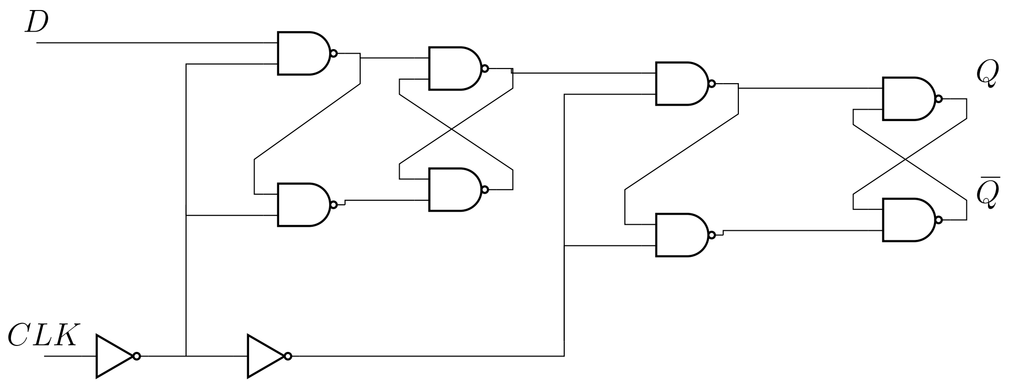

Figure 23: D Flip Flop using NAND2 gates

then the devices I_2 and I_4 should be weaker. Initially, we designed all devices (inverters and transmission gates) to have $W_{NMOS} = 4\mu u$, $W_{PMOS} = 8\mu m$ and $L = 2\mu m$, except for the inverters I_2 and I_4 for which we set L equal to $4\mu m$ to make them weaker (fig. 26). However, our flip flop failed to successfully switch states, since I_2 and I_4 were not weak enough compared to the rest of the circuit (fig. 27). Therefore, we doubled the size of the transistors in the transmission gates and got a properly functioning flip flop (figures 28 and **??**.

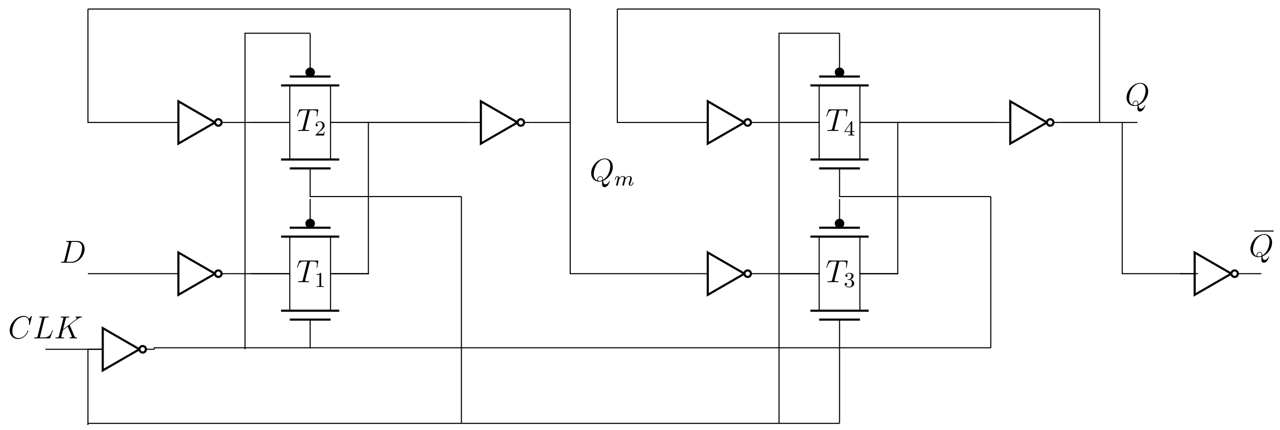

Figure 24: Master-slave positive edge-triggered register using multiplexers.

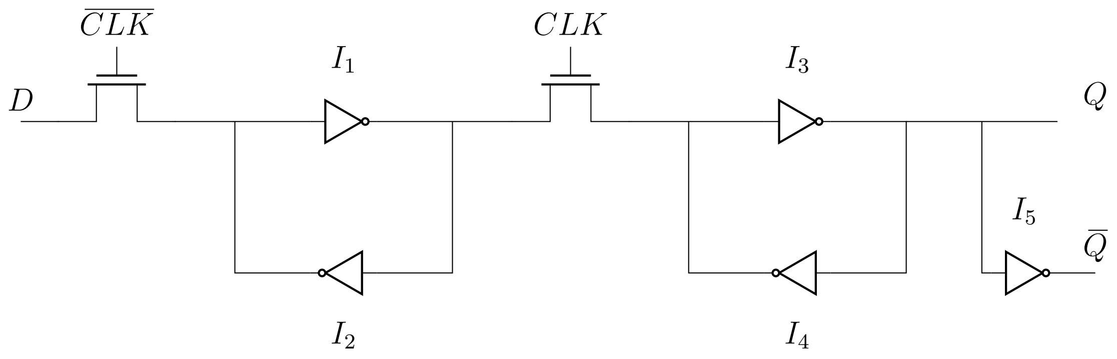

Figure 25: Reduced Load Clock Static Master-Slave Flip Flop using only NMOS pass transistors

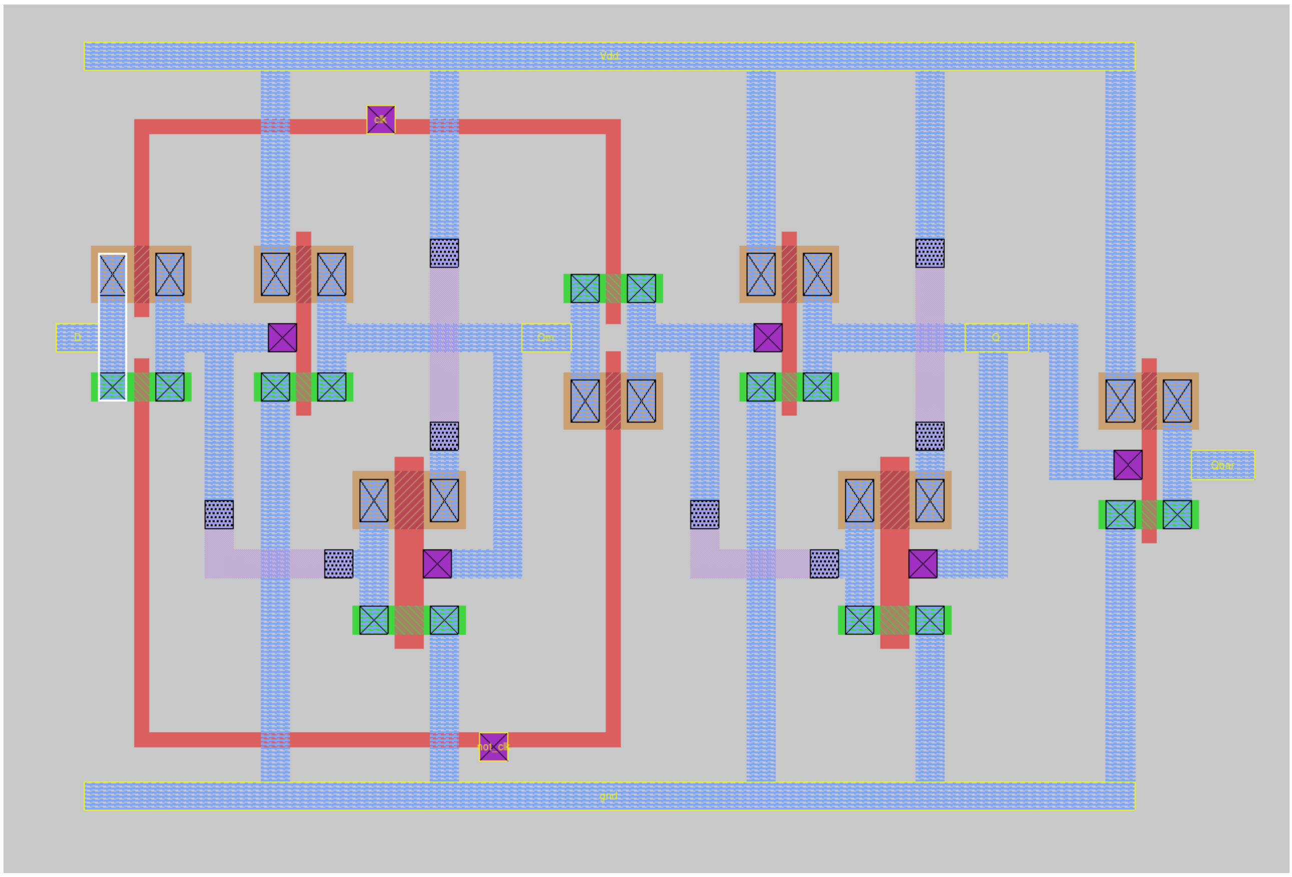

Figure 26: Unsuccessful D flip flop implementation

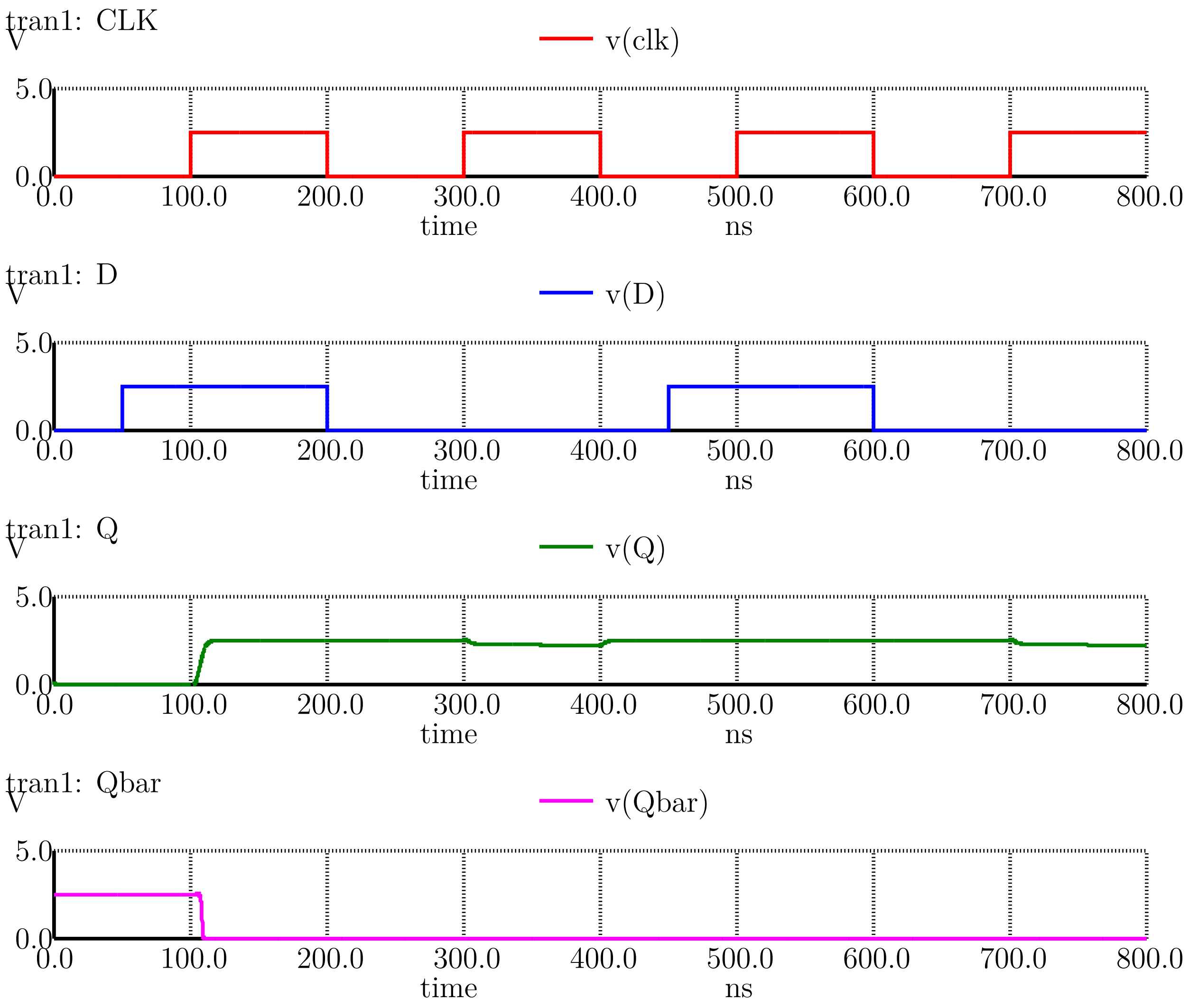

Figure 27: SPICE simulation for the unsuccessful D flip flop implementation

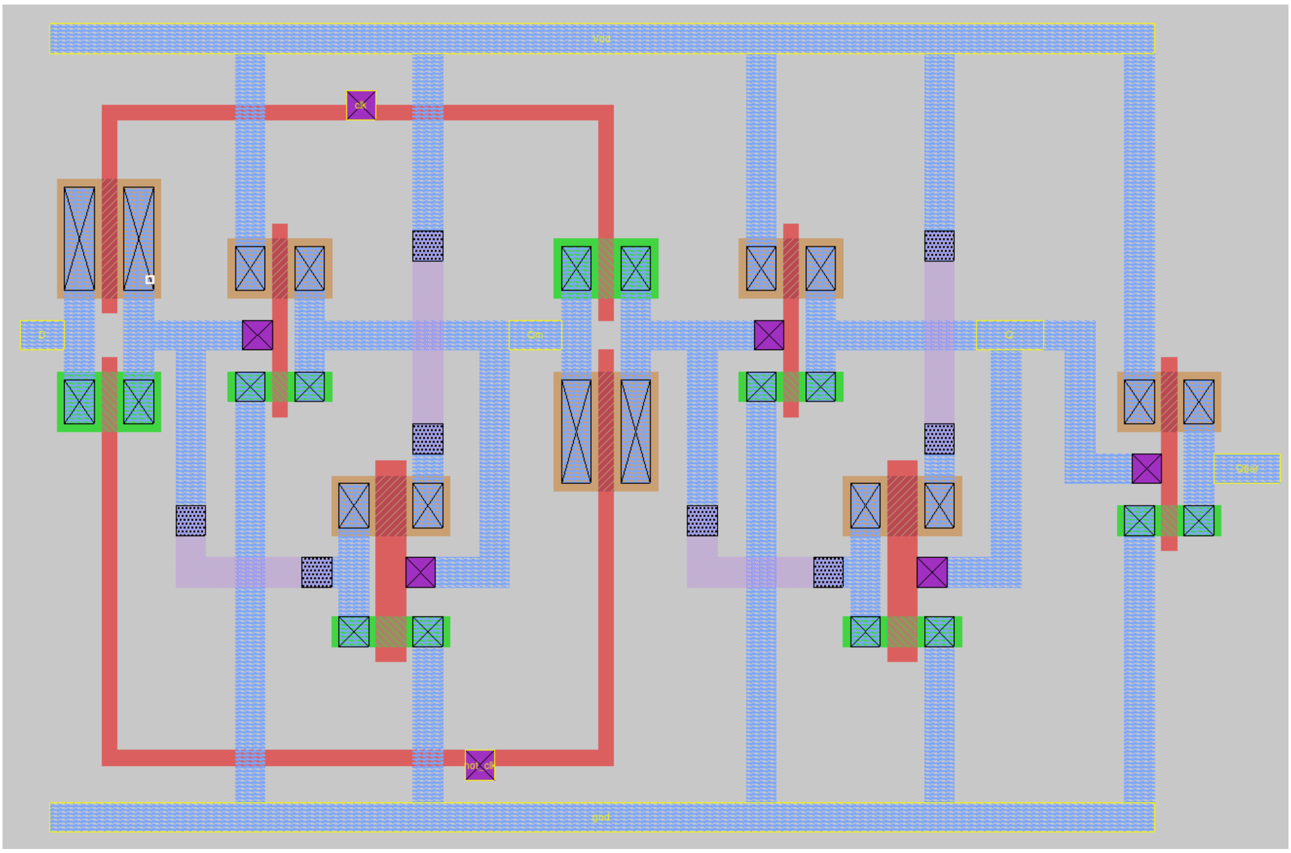

Figure 28: Functionally correct D flip flop implementation

4 D Flip-Flop with Asynchronous Preset & Clear (DF-FRS)

For this part of the project, we had to design a D flip-flop with asynchronous active-high preset and clear. The truth table for this D flip-flop is presented in table 3.

CLK	D	S	R	Q	$\bar{Q}$
X	X	1	0	1	0
X	X	0	1	0	1
X	X	1	1	?	?
↑	1	0	0	1	0
↑	0	0	0	0	1

Table 3: D Flip-flop with asynchronous preset and clear truth table

Our first attempt at designing this flip flop was inserting two multiplexers right before the output as shown in figures 30 and 31 . If S is equal to 1, then the first multiplexer selects 1 to propagate to the output else it propagates the output of the flip flop. Similarly, when R is equal to 1, the output Q is going to be set to 0 by the second inverter. In the case where S and R are both equal to 1, the output will be dictated by the second multiplexer and will be equal to 0.

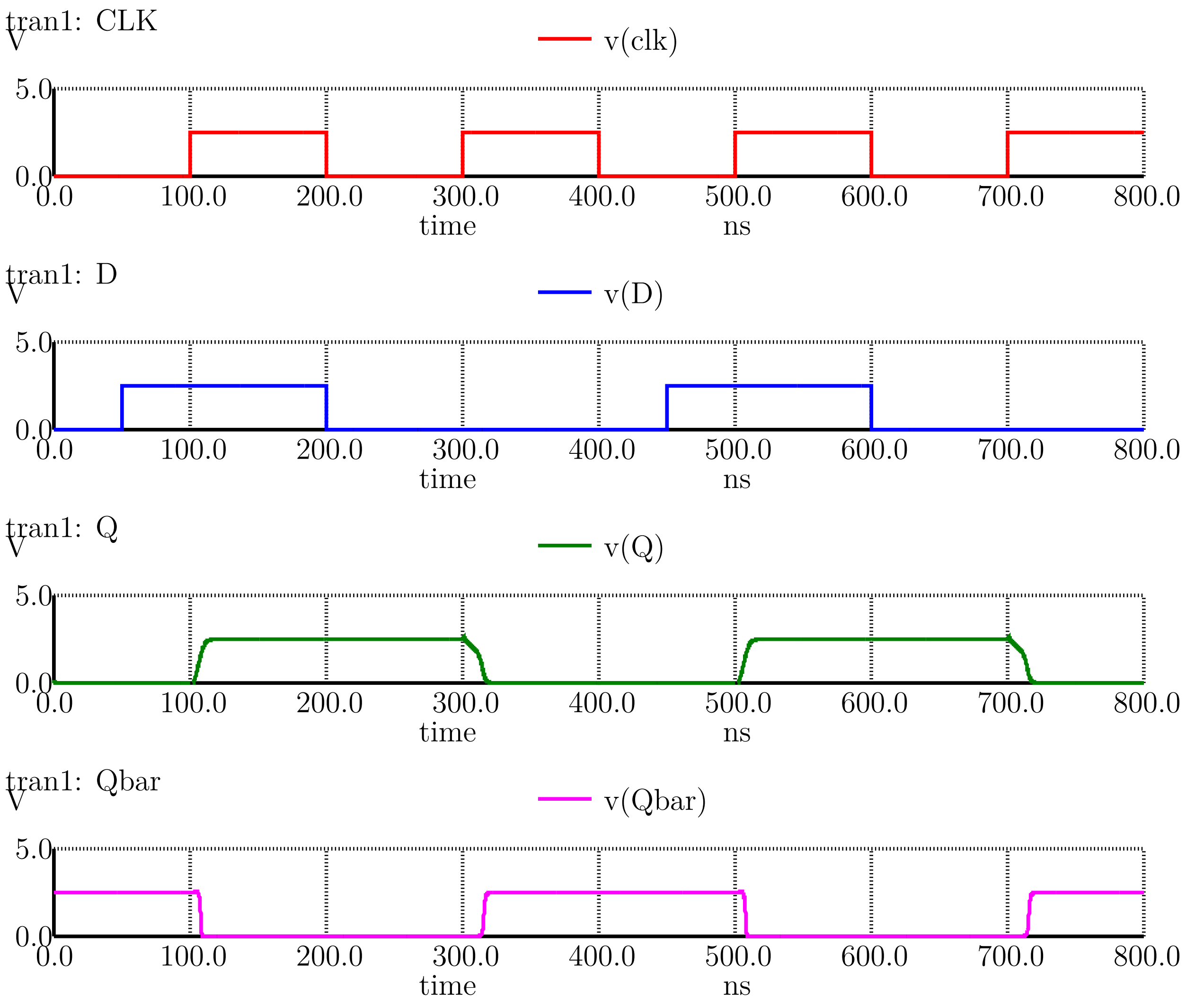

Figure 29: SPICE simulation for the correct D flip flop implementation

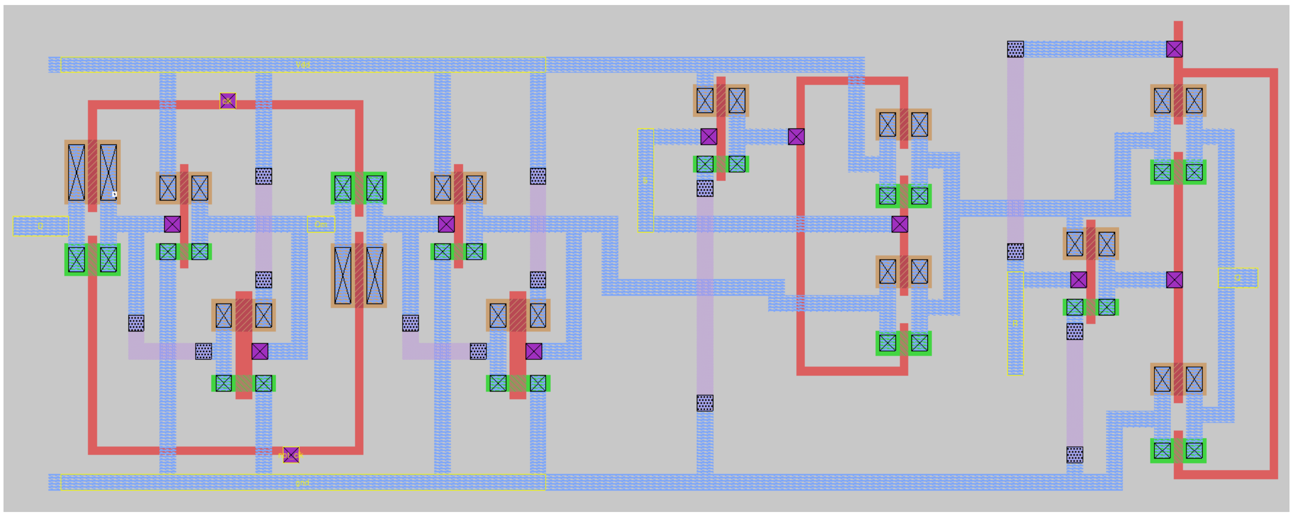

Figure 30: D flip-flop with asynchronous preset and clear using multiplexers

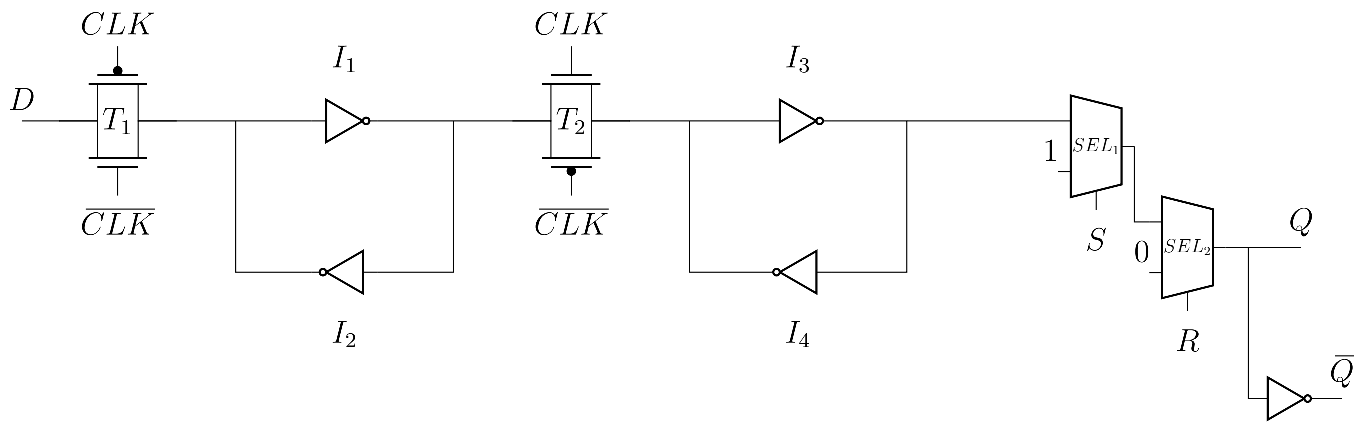

Figure 31: D Flip-flop with multiplexer-based asynchronous preset (S) and clear (R)

Figure 32 shows the simulation waveforms for this implementation. We can see that as soon as S or R becomes 1, the output goes to 1 or 0 respectively, with R having a higher priority over S. However, as soon as R or S are unset, the flip flop returns to the state that it had previously stored. For example, at around 33ns, R is set and Q correctly becomes low. However, after 38ns and before S is set, Q returns to high, whereas it should have stayed low until S becomes high at 38ns. This hinted at the fact that the set and reset should actually switch the stored value. More specifically, we have to change the stored value at the master latch, so that the slave latch passes through that value.

The first step to constructing the flip flop was to try and enforce the desired state at the master latch. We began with just the preset. We didn't tamper with the slave latch nor the output, so the flip flop had a synchronous set.

The idea was to insert a combinational logic element between transmission gate T_1 and inverter I_1 and between I_1 and I_2. The latter was necessary because inverter I_2 would try to impose the old state and set would not be successful. In order for Q to be equal to 1, the output of I_1 should be 0, and, therefore the output of the combinational logic between T_1 and I_1 should be 1 in case $S = 1$. On the other hand, the output of inverter I_2 should be 1, so its input should be driven to 0 in case $S = 1$. The truth tables for both combinational logic elements are shown in tables 4 and 5. It turns out that they represent an OR2 and an AND2 gate respectively. Constructing those gates using static CMOS logic would require 4 transistors each. However, considering that they are followed by an inverter, which is a voltage rectifier, we could use NMOS pass transistor logic for each gate that needs only 2 transistors (figures 35 and 34).

The simulation waveforms of figure 36 prove that our flip flop with synchronous set works correctly; Q becomes high when $S = 1$ on the positive edge of the clock and stays high even after S has gone to 0.

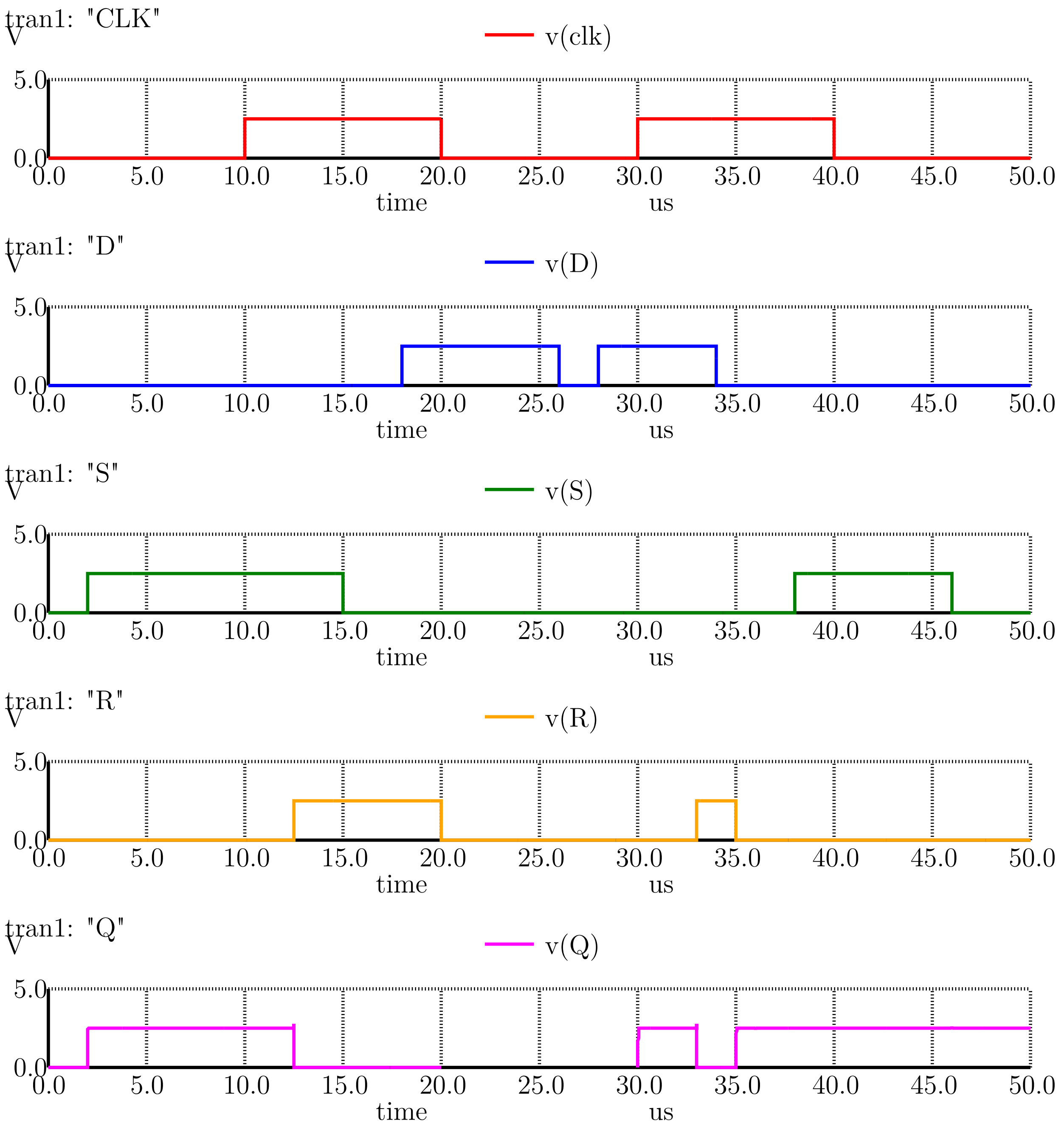

Figure 32: SPICE simulation for the D flip-flop with asynchronous preset and clear using multiplexers

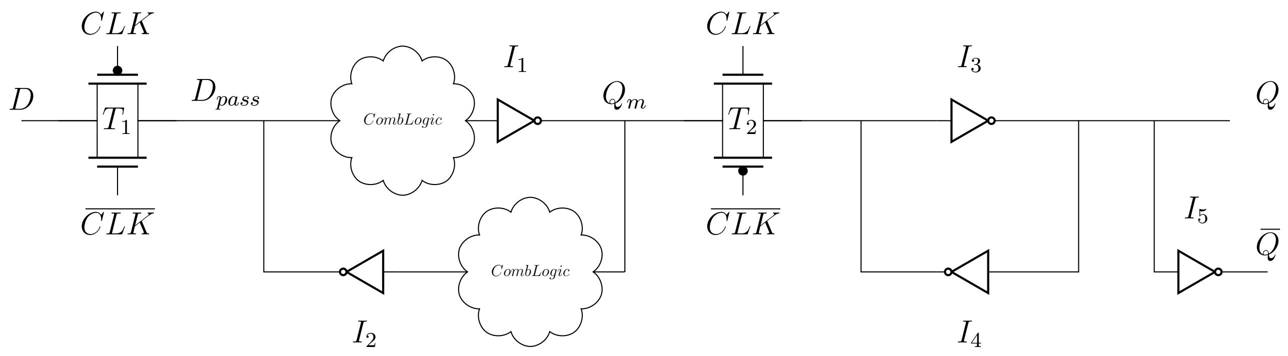

Figure 33: Reduced Load Clock Static Master-Slave Flip Flop with Synchronous Set

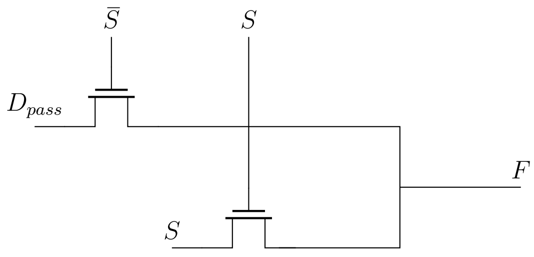

Figure 34: OR gate

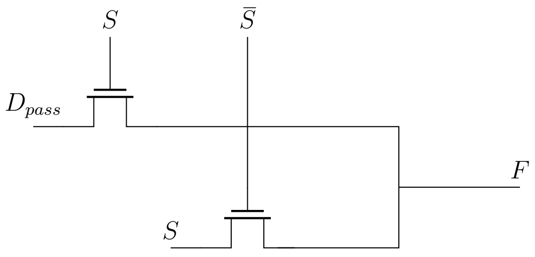

Figure 35: AND gate

$\mathbf{D_{pass}}$	**S**	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Table 4: OR gate truth table

$\mathbf{Q_m}$	**S**	**F**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Table 5: AND gate truth table

The exercise, however, calls for an asynchronous set, meaning that we must set the output Q to 1, regardless of the clock. So, we need a multiplexer before inverter I_3 that selects among the output of the pass gate T_2, which is only updated on the positive edge of the clock, and Q_m, which immediately switches to 0, after set has been set to 1 (38, 37, 39). By the time S becomes low again, the new state has already been stored in the bistable element of the slave latch and stays there until the next positive clock edge, when it changes to the value of D. Had we placed the multiplexer after inverter I_3 and before output Q, Q would revert to its stored state once S returns to low.

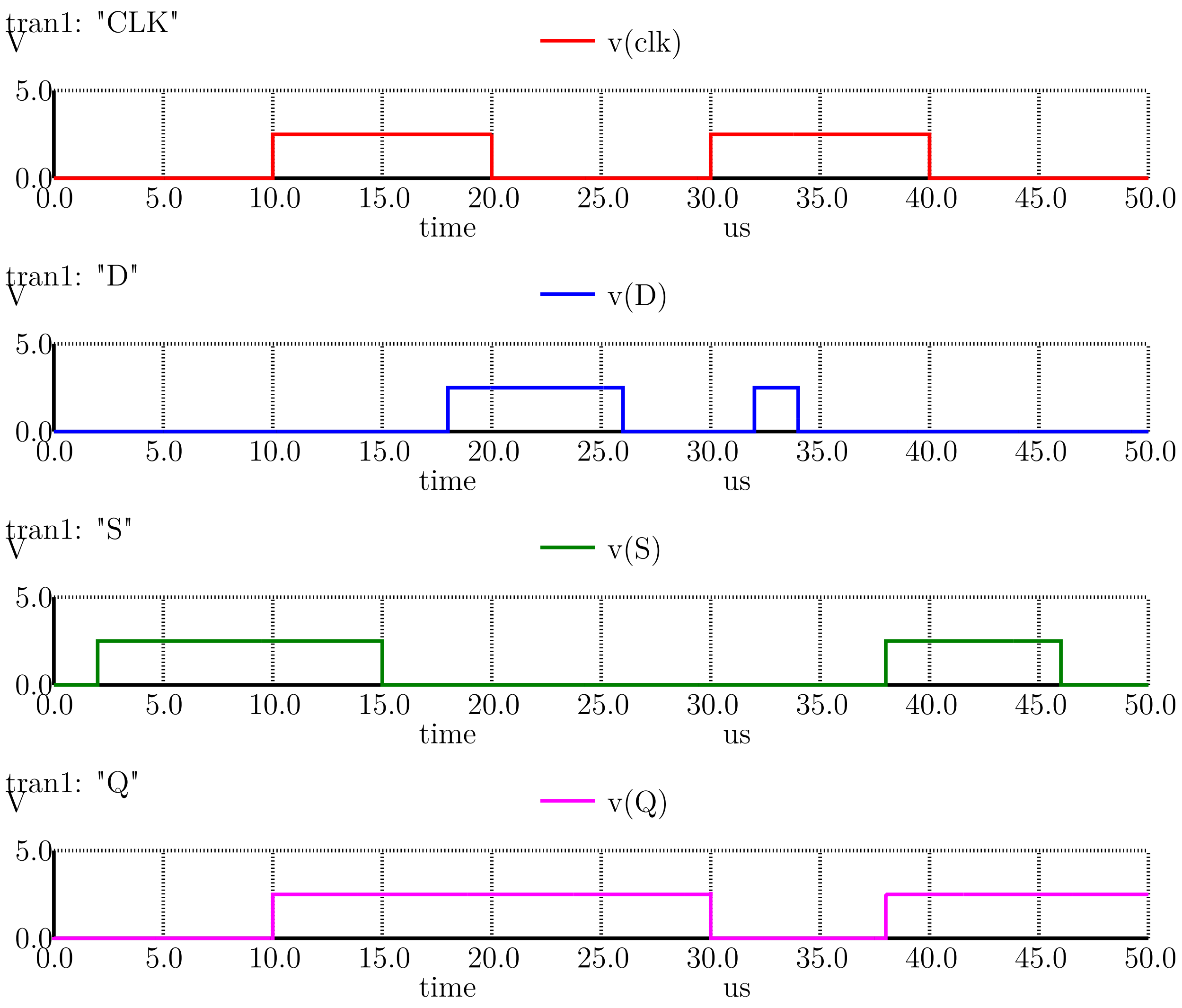

Figure 36: SPICE simulation for the D flip flop implementation with synchronous set

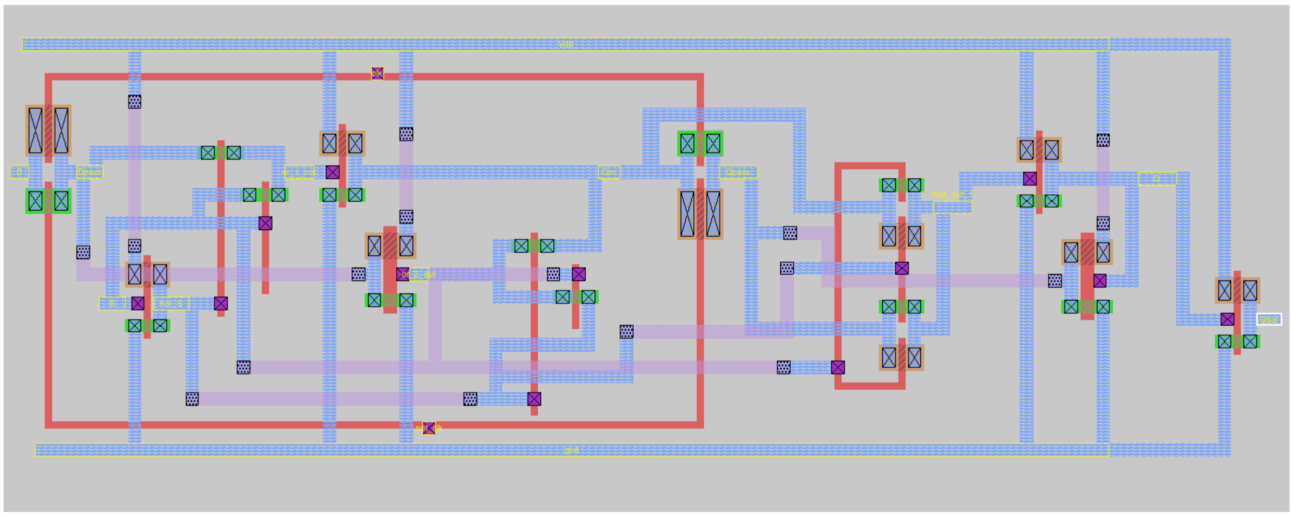

Figure 37: D flip-flop with asynchronous set

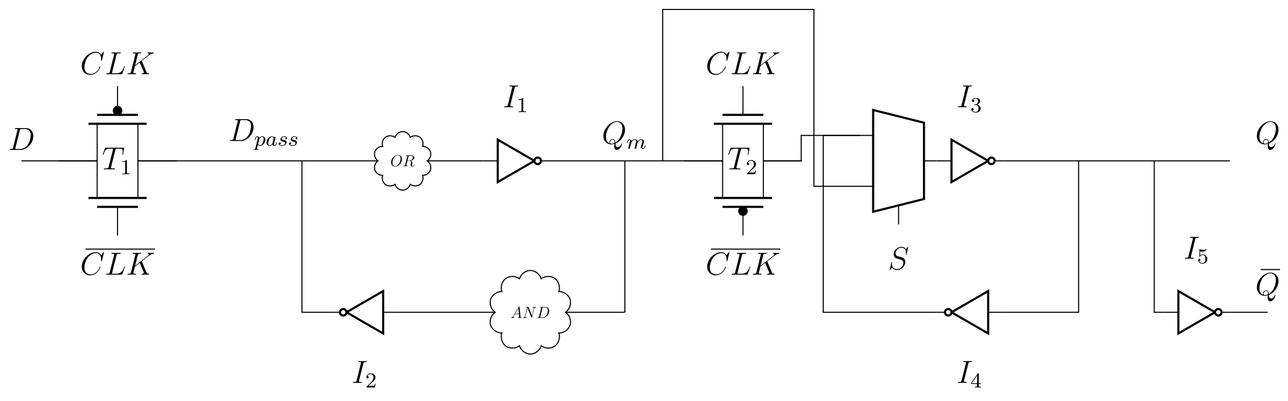

Figure 38: Reduced Load Clock Static Master-Slave Flip Flop with Asynchronous Set

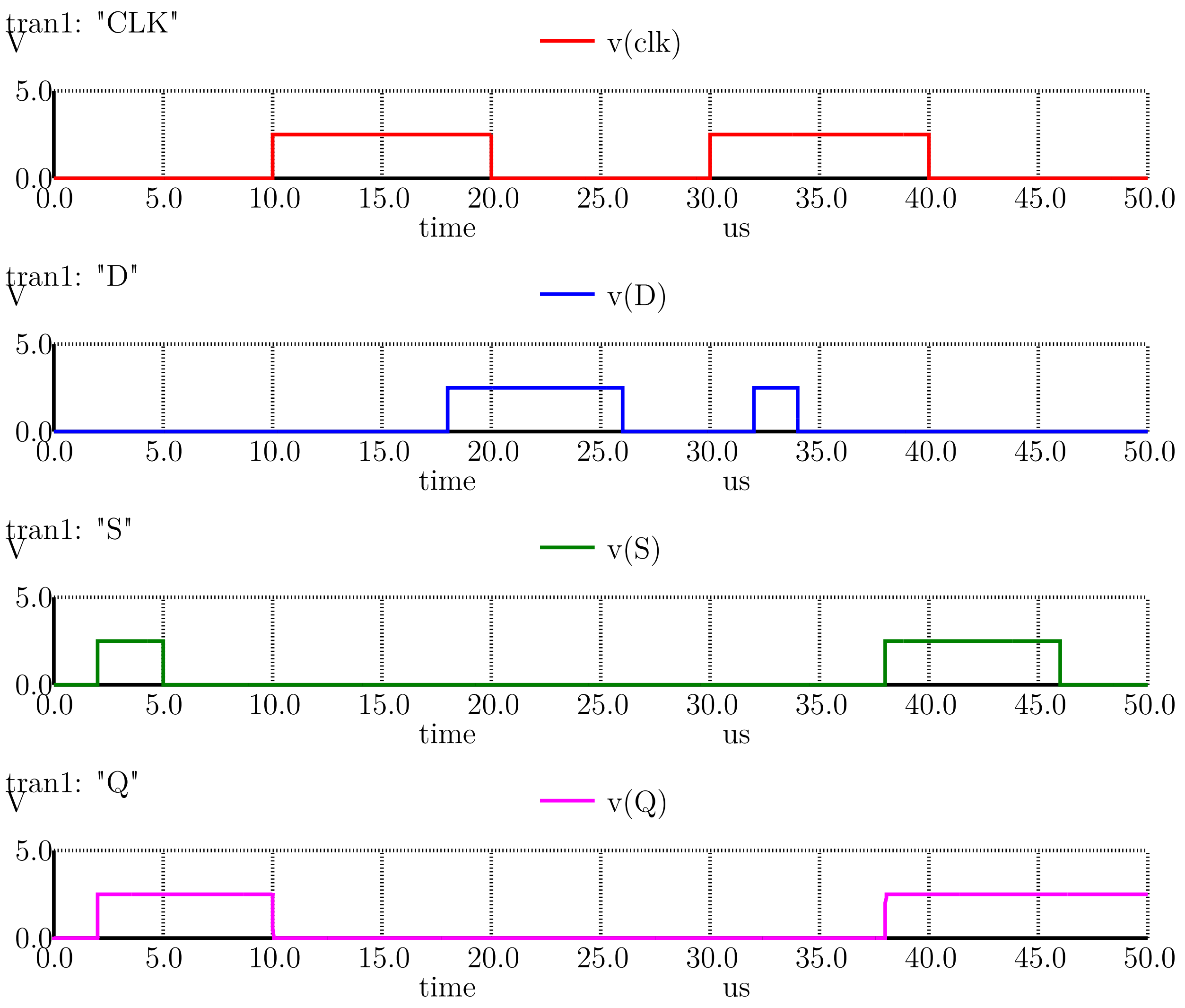

Figure 39: SPICE simulation for the D flip flop implementation with asynchronous set

Our next step was to modify our previous implementation to include the asynchronous reset. Let $Dset_1$ be the input to I_1 (after the OR gate) and $Dset_2$ be the input to I_2 (after the AND gate). Then, if we want to reset the output Q to zero when $R = 1$, then the output of I_1 should be 1, the input to I_1 should be 0, the input to I_2 should be 1 and the output of I_2 should be zero. Those observations can be summed in the truth tables 6 and 7. We can conclude that we need an AND gate before I_1 and an OR gate before I_2. In addition, since we want the reset to be asynchronous, we should modify the multiplexer to pass Q_m to I_3, if $S = 1$ or $R = 1$, which means that the select signal will now be the output of an OR NMOS pass transistor gate that has S and R as inputs. The final D flip-flop with asynchronous preset and clear is pictured in figure 41 and 42. Figure 40 verifies the functionality of our flip flop.

$\mathbf{D_{set_1}}$	**R**	**F**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Table 6: AND gate truth table

$\mathbf{D_{set_2}}$	**R**	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Table 7: OR gate truth table

We used transistors 34 in total. More specifically:

1. 4 transistors for the transmission gates T_1 and T_2
2. 16 transistors for the inverters (including the transistors needed to invert S and R and the one that is used to invent the output of the OR gate that has them as inputs).
3. 6 transistors for the OR gates
4. 4 transistors for the AND gates
5. 4 transistors for the multiplexer

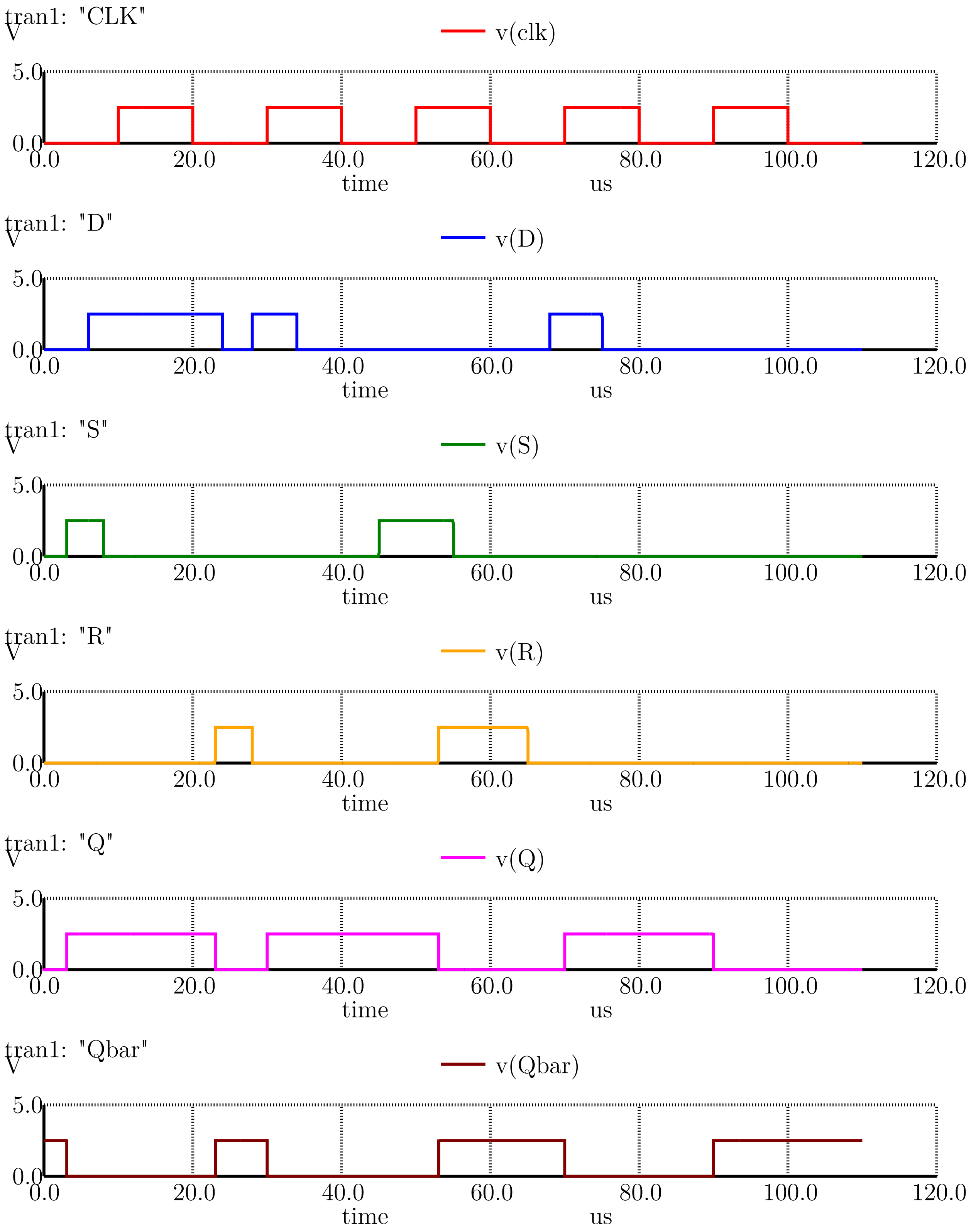

Figure 40: SPICE simulation for the D flip flop implementation with asynchronous preset and clear

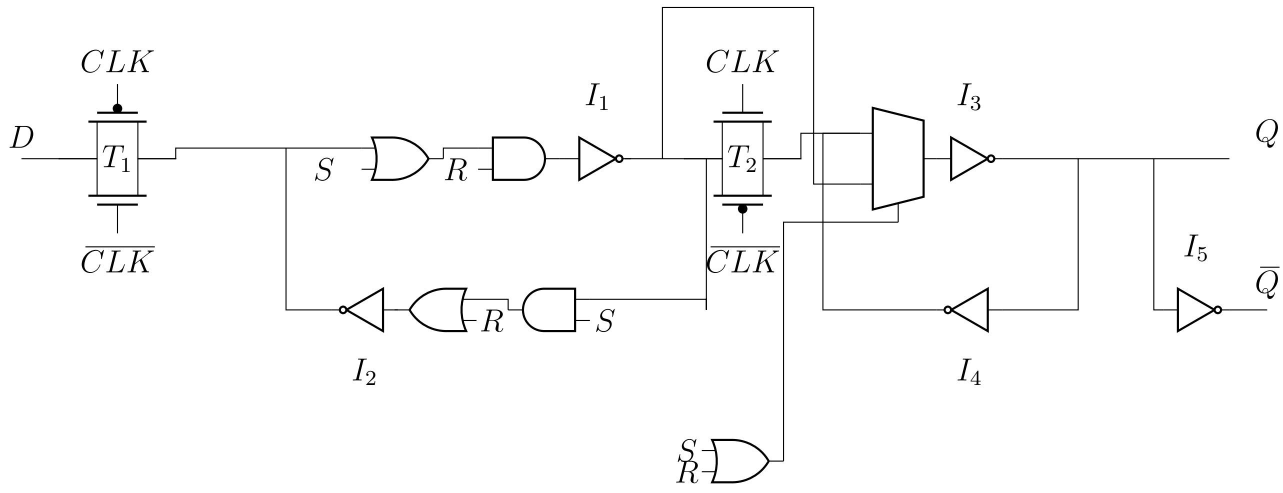

Figure 41: Reduced Load Clock Static Master-Slave Flip Flop with Asynchronous Set and Reset

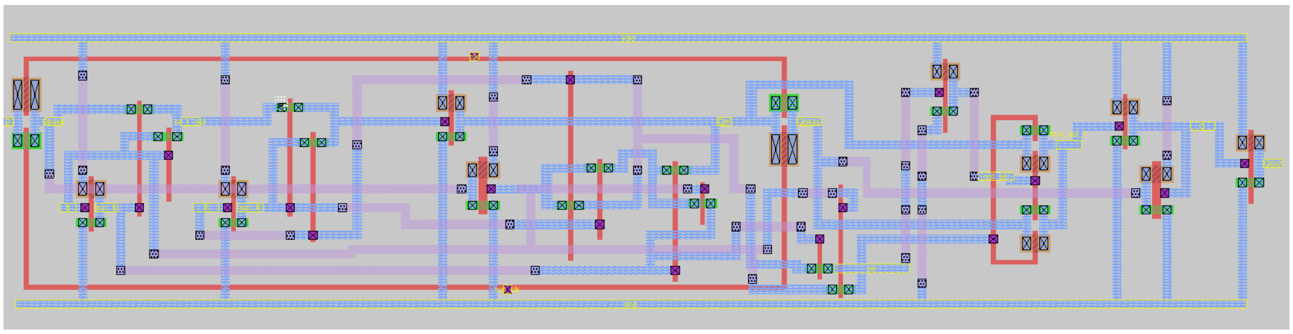

Figure 42: D Flip-flop with asynchronous set and reset MAGIC layout

5 Conclusion

In this part of the project, we mainly used the MAGIC tool and NgSpice for the simulations. The layouts required a lot of planning-ahead, because there were parts of the circuit that needed to be connected to both ground and V_{DD}. Also, some signals had to be propagated, such as S had to be propagated almost entirely across the flip flop, which required changing metal levels and using a lot of metal contacts. We tried to make the flip flop as compressed as possible.