THE UNIVERSITY OF BRITISH COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ELEC 402 Assignment 5: PnR, Delay, Power, and Interconnects

Shidi Xi (90506643)

December 9, 2022

1 Cell library layout

Design concept (taken from project report 1)

The design concept and the real-world functionality of the FSM is based upon a Python script that the author has developed during his Co-op at a video surveillance company. Hence, a quick introduction of the script will be given for better understanding of the design of the FSM.

The company's cameras have embedded firmware on them, and for any scenes or videos that a camera is capturing or streaming, the camera's firmware will analyze the amount of motion and details contained in the scene and output two numbers, namely, the motion score and the detail score. In one of the tests that are performed on cameras, videos which deliver specified motion scores and detail scores need to be made by the test engineer. E.g., when this video is played in front of a camera, the camera must give a motion score of say, 5, and a detail score of say, 60. The Python script can automatically generate videos meeting the motion and detail requirements provided by the engineer. It consists of two modules (classes), shown by Figure 1, one is a class containing individual functions of making a video using provided detail and motion parameters. It is analogous to a datapath in digital systems such as the central processing unit. The other class is, not surprisingly, a controller. It calls the function in the datapath, providing detail and motion parameters. The datapath then makes the video, gets the detail and motion scores from the camera's firmware, and feeds the scores back to the controller, which then processes the scores, and generates new parameters. New parameters will produce a video that is closer to the target. The process repeats until the controller receives detail and motion scores that are equal to the targets.

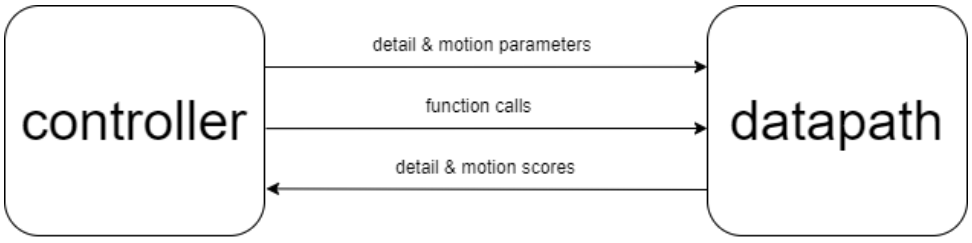


Figure 1: The block diagram showing the two modules (classes) in the Python script.

The FSM designed in this project is in fact the SystemVerilog implementation of the controller class in the Python script. Figure 2 below shows all the inputs and outputs of the FSM module. One can see it has a clock (`clk`) and a reset (`rst`) signal on the input, just like the majority of digital finite-state machines. The 3-bit `aim` signal set the detail target for the FSM, which will be explained later. The 8-bit `motion_score` and `detail_score` signals are from the datapath, if it was to be designed

using SystemVerilog. On the output side, there are five 1-bit enable signals. If there is a datapath, these are the enable signals for the functional units inside the datapath. With the analogy to Python, they behave like function calls. For example, the `make_motion` function will make a video containing a moving object and the `check_motion` function will get the motion scores from the camera's firmware. Finally there are the 8-bit `motion_prmt` and `detail_prmt` which are the parameters the controller provides to the datapath to make videos. For instance, larger `motion_prmt` will make the moving object bigger and larger `detail_prmt` will make the video contains more details.

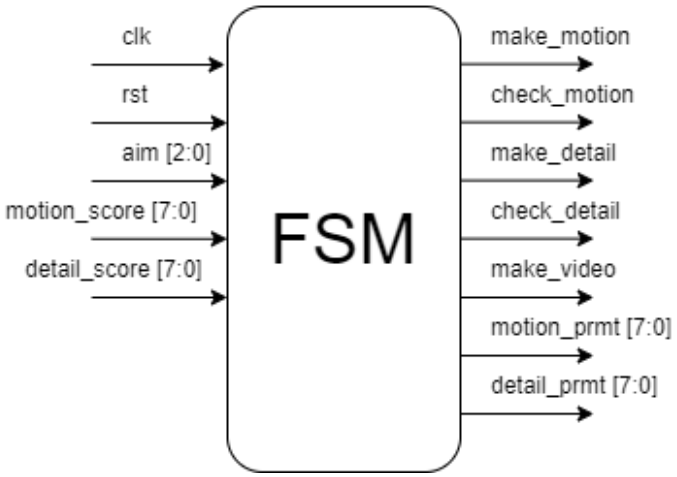


Figure 2: The block diagram of the FSM designed in this project. All I/O ports are labeled.

RTL simulation (taken from project report 1)

Figure 3 shows the connection between the testbench module and the device under test (DUT, which is the FSM) it instantiates. Basically the testbench mimics the behavior of the datapath that the FSM is designed to control. It sends `motion_score` and `detail_score` feedback to the FSM, which compares these scores with the targets and makes new `motion_prmt` and `detail_prmt`. Figure 4 shows the simulation waveform of the FSM in ModelSim.

Layout

Figure 5 shows the layout of the FSM generated by Cadence Innovous. The total area occupied is $41.8 \times 37.1\ \mu m^2$.

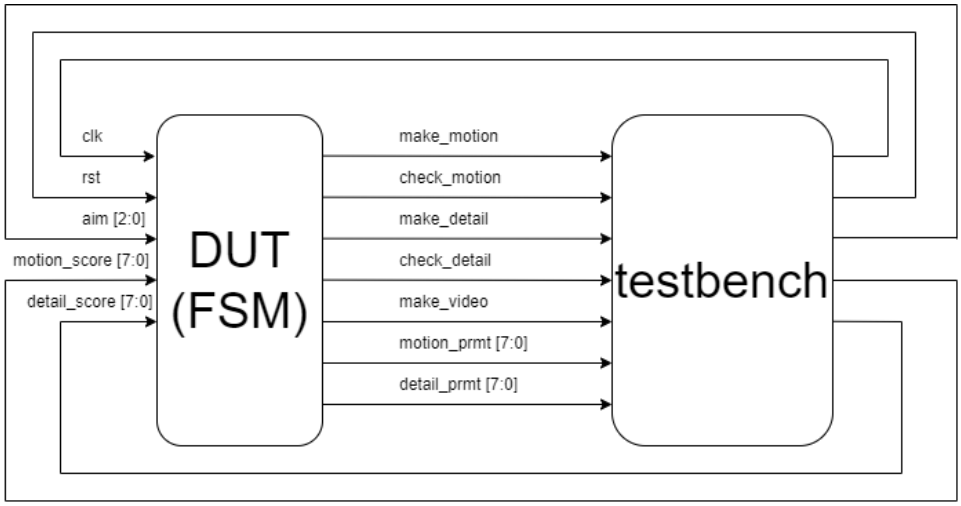
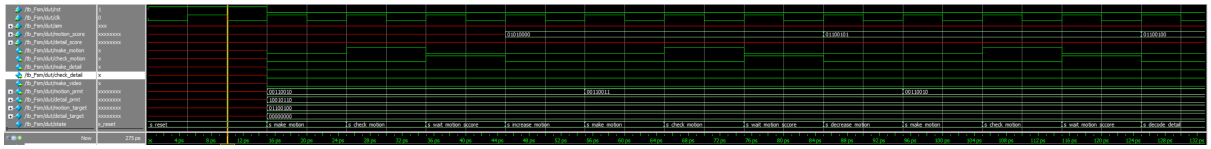
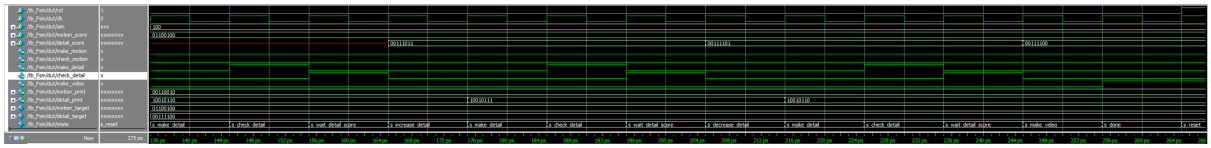


Figure 3: A block diagram showing how the testbench is connected to the DUT (FSM).



(a)



(b)

Figure 4: ModelSim simulation waveform of the FSM.

Post-synthesis simualtion

In project 2, post-synthesis simulation was run in Modelsim. In this project, it is run in Candence using the Spectre SPICE. Unfortunately however, the schematic of the testbench and the waveform were unable to be obtained due to an X2Go service shutdown before the due time.

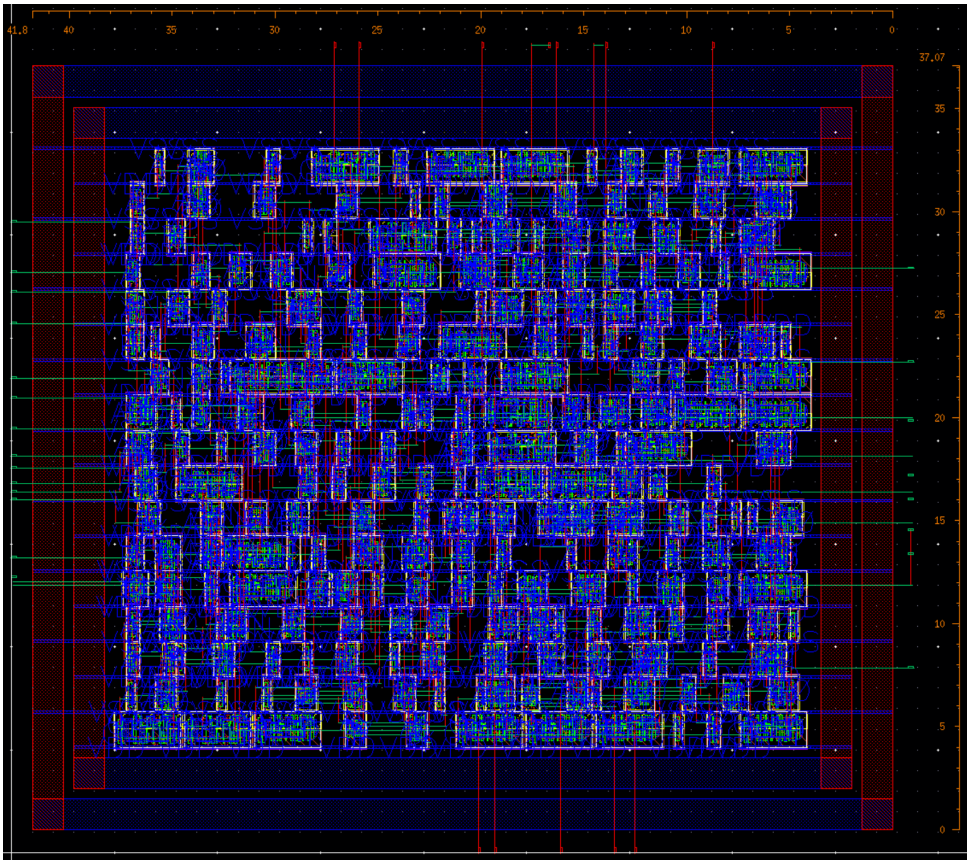


Figure 5: Layout of the synthesized FSM circuit.

2 Domino logic

1

$$OUT = AB(C + D)$$

2

The worst case charge-sharing occurs when ABCD=1100. In that case, considering shared diffusion,

$$C_A = 0.3C_{eff} = 0.3\ fF$$
$$C_B = 0.6C_{eff} = 0.6\ fF$$
$$C_L = 0.5C_{eff} + 0.6C_g = 1.7\ fF$$
$$V^* = \frac{C_A V_A + C_B V_B + C_L V_L}{C_A + C_B + C_L} = 0.65\ V$$

However, if we assume $V_{DD} = 1\ V$ and $V_{Tn} = 0.4\ V$, then V_A and V_B cannot exceed $V_{DD} - V_{Tn} = 0.6\ V$. We then have,

$$V_L = \frac{V_L^0 C_L - 0.6(C_A + C_B)}{C_L} = \frac{(1 \times 1.7) - [0.6 \times (0.6 + 0.3)]}{1.7} = 0.682\ V$$

3 Capacitance and delay calculations

Minimum delay

The minimum delay occurs when $AB = 00$ or $AC = 10$. The equivalent circuit is shown in Figure 6.

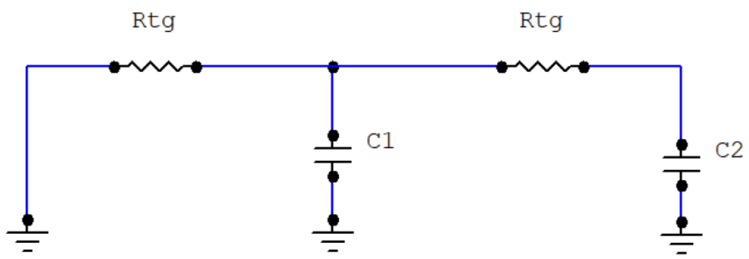


Figure 6: Minimum delay equivalent circuit.

$$W = 2\lambda = 0.13\ \mu m$$
$$C_{eff} = 1\ fF/\mu m$$
$$C_g = 2\ fF/\mu m$$
$$C1 = 2C_{eff}W \times 3 + C_gW \times 2 = 1.3\ fF$$
$$C2 = 2C_{eff}W \times 2 + C_gW = 0.78\ fF$$
$$R_{TG} = R_{eq}^N \frac{L}{W} = 15\ k\Omega$$
$$\tau = 2R_{TG}C2 + R_{TG}C1 = 42.9\ ps$$

Hence, minimum delay $= 0.7\tau = 30.0\ ps$.

Maximum delay

The maximum delay occurs when $AB = 01$ or $AC = 11$. The equivalent circuit is shown in Figure 7.

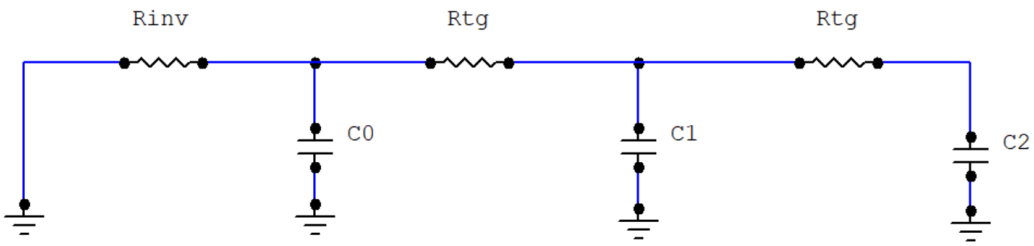


Figure 7: Maximum delay equivalent circuit.

$$C0 = 2C_{eff}W + C_gW + 3C_{eff}W = 0.91\ fF$$
$$R_{INV} = R_{eq}^N \frac{L}{W} = 15\ k\Omega$$
$$\tau = (2R_{TG} + R_{INV})C2 + (R_{TG} + R_{INV})C1 + R_{INV}C0 = 89.3\ ps$$

Hence, maximum delay $= 0.7\tau = 62.5\ ps$.

4 Power consumption

Pseudo NMOS inverter

$$P_{static} = V_{DD}I_D$$
$$I_D = I_{sat}^P = Wv_{sat}C_{ox}\frac{(V_{GS} - V_T)^2}{V_{GS} - V_T + E_cL}$$
$$W = 68\ nm \times n_{fins} = 68 \times 4 = 272\ nm$$
$$v_{sat} = 8 \times 10^6\ cm/s$$
$$C_{ox} = 1.6 \times 10^{-6}\ F/cm^2$$
$$E_cL = 24\ V/\mu m \times 15\ nm = 0.36\ V$$
$$V_{GS} - V_T = 1 - 0.4 = 0.6$$
$$P_{static} = 0.131\ mW$$

Figure 8 and Figure 9 show the testbench and the waveform of the transient simulation implemented to find the static current I_D when input remains high. We get $I_D = 0.7\ mA$ and hence $P_{static} = 0.7\ mW$. Comparing the simulation value with the calculated value, we see a noticeable difference. This is likely due to the fact that parameters from the 130 nm technology were used in calculation rather than 15 nm parameters.

$$
\begin{aligned}
&P_{dynamic} = \alpha C V_{DD}^2 f \\
&\alpha = 1 \\
&C = 14 C_{eff} + C_L \approx 10.4\ fF \\
&V_{DD} = 1\ V \\
&f = 100\ MHz \\
&P_{dynamic} = 1.04\ \mu W
\end{aligned}
$$

The circuit shown in Figure 8 was modified into a CMOS topology while maintaining the sizing of the transistors, in order to find the dynamic power. The result we get from the simulation is $P_{dynamic} = 1.23\ \mu W$.

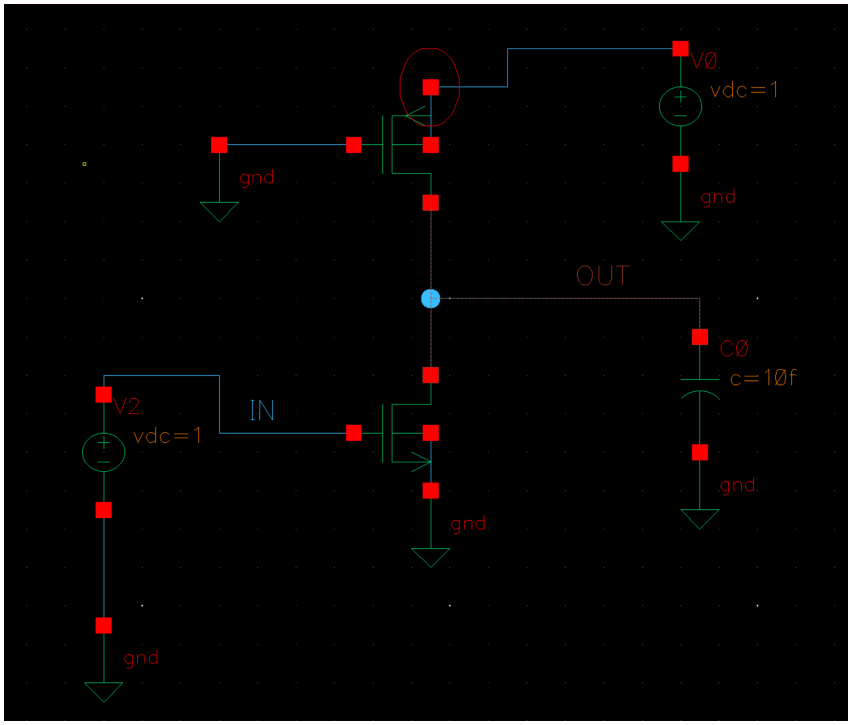


Figure 8: Testbench used to find the static power of the pseudo NMOS inverter.

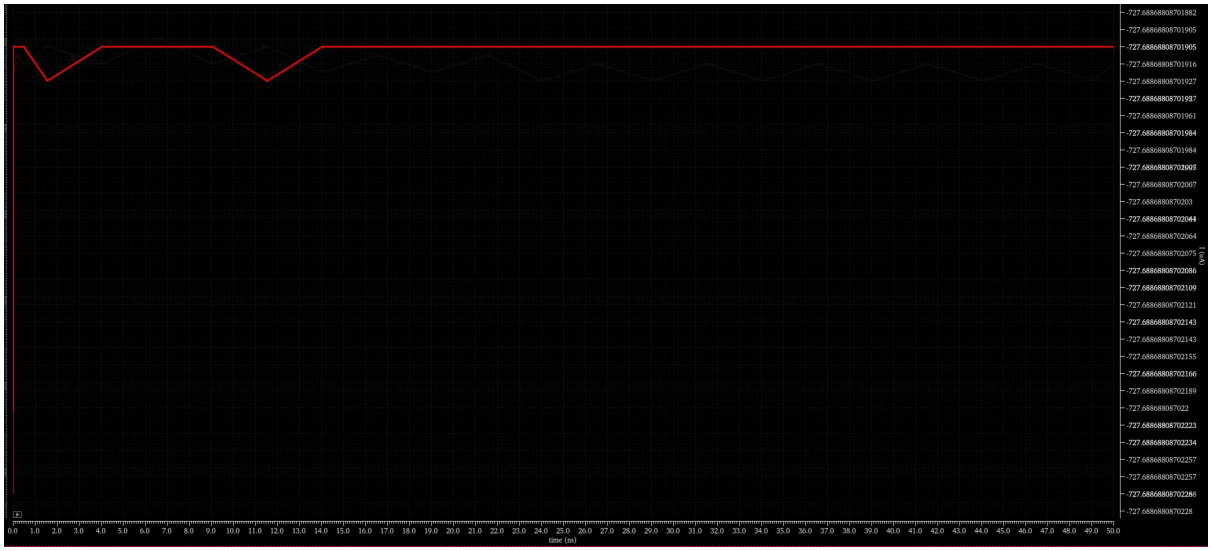


Figure 9: Transient simulation waveform of I_D of the PMOS when the input remains high.

CMOS inverter

$$
\begin{aligned}
&P_{dynamic} = \alpha C V_{DD}^2 f \\
&\alpha = 1 \\
&C = 6C_{eff} + C_L \approx 10\ fF \\
&V_{DD} = 1\ V \\
&f = 100\ MHz \\
&P_{dynamic} = 1\ \mu W
\end{aligned}
$$

Figure 10 and Figure 11 show the testbench and the waveform of the transient analysis implemented to fine the dynamic current passing through the drain of PMOS due to the charging of the capacitors at the output node. From this current, we can get the average power using Cadence Calculator, which yields $1.09\ \mu W$. This value is very close to our calculations, yet still, it is slightly bigger as there are non-zero rise and fall time of the input in the simulation, which can lead to the presence of the short-circuit current, shown in Figure 11 as the small peaks.

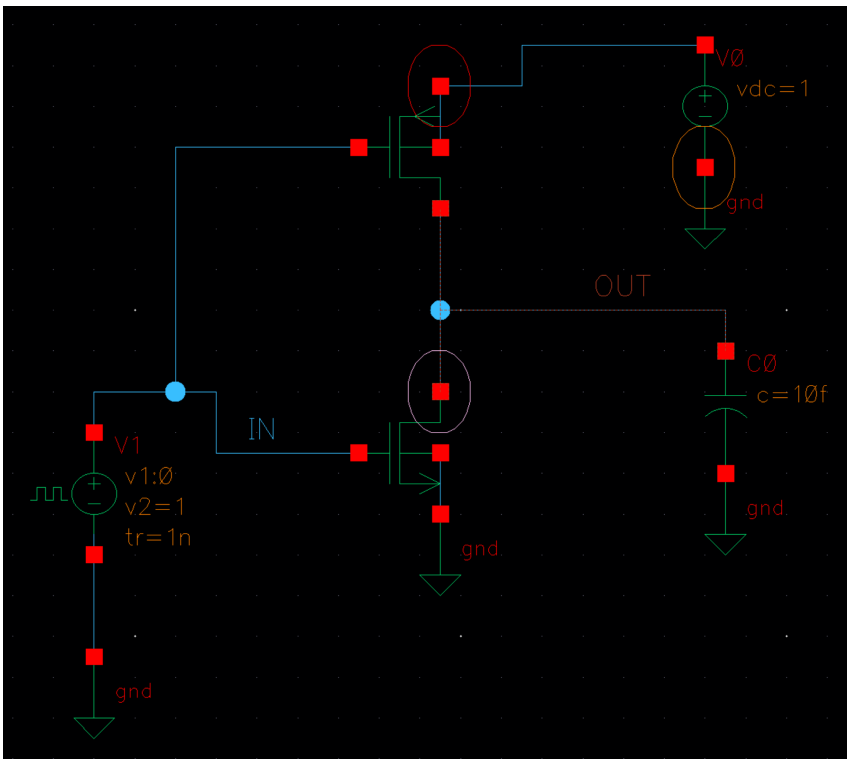


Figure 10: Testbench used to find the dynamic power of the CMOS inverter.

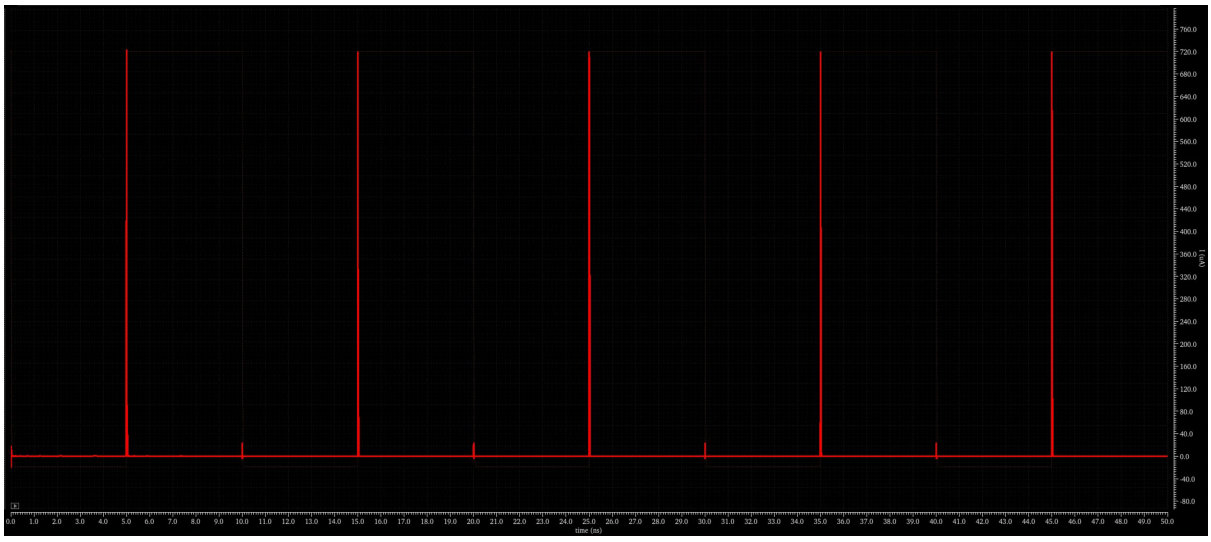


Figure 11: Transient simulation waveform of I_D of the PMOS.

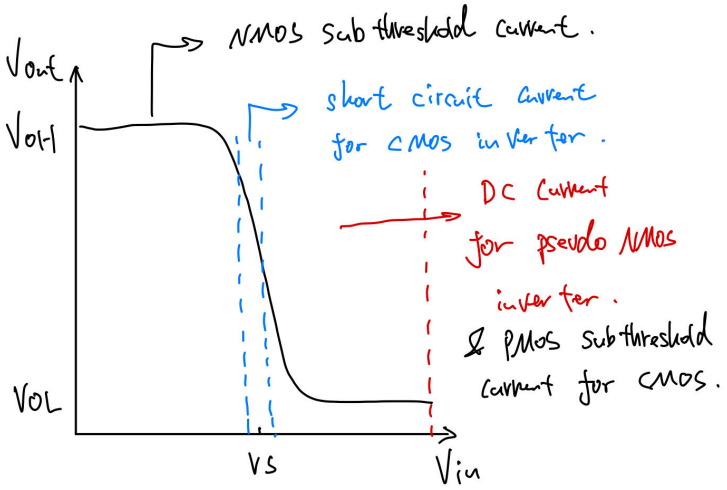


Figure 12: VTC of the inverter.

5 Activity factor

Two complete toggles in 10 cycles.

$$\alpha = \frac{2}{10} = 0.2$$

6 Interconnects

(a)

$$R_{pl} = \frac{\rho}{A} = \frac{0.017}{0.4 \times 0.8} = 0.05\ \Omega/\mu m$$
$$C_{pl} = C_{area} + C_{lateral} + C_{fringe}$$
$$= 3 \times 88.5 \times 10^{-4} \times [2\frac{0.4}{0.5} + 2\frac{0.8}{2} + 4ln(1 + \frac{0.8}{0.5})] = 0.165\ fF/\mu m$$

(b)

$$R_W = R_{pl}L = 900\ \Omega$$
$$C_W = C_{pl}L = 2.97\ pF$$

The equivalent circuit is shown in Figure 13.

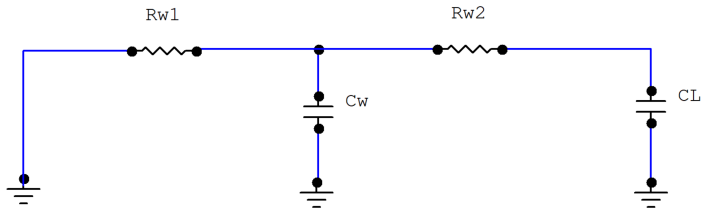


Figure 13: Equivalent circuit of the wire with a perfect voltage source as the driver. The T-model is employed. $R_{W1} = R_{W2} = R_W/2$.

$$t_{pd} = 0.7(R_W C_L + \frac{R_W}{2} C_W) = 0.967\ ns$$

(c)

The equivalent circuit is shown in Figure 14.

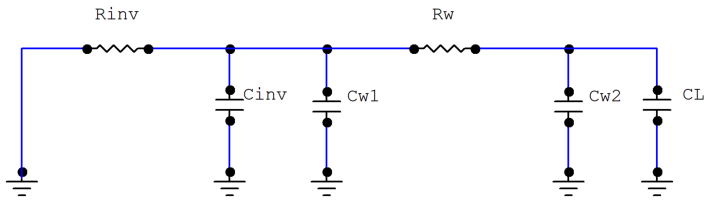


Figure 14: Equivalent circuit of the wire with a 25X inverter as the driver. The π-model is employed. $C_{W1} = C_{W2} = C_W/2$.

$$R_{inv} = R_{eq}\frac{L}{W} = 15000 \times \frac{40}{25 \times 80} = 300\ \Omega$$

$$C_{inv} = C_{self}W = 2\ fF$$

$$t_{pd} = 0.7[(C_L + \frac{C_W}{2})(R_{inv} + R_W) + (C_{inv} + \frac{C_W}{2})R_{inv}] = 1.41\ ns$$