6   Basic Gates

The basic logic gates are described in this chapter. The principles for building combinational logic circuits are developed. Details on layout implementation are also provided.

1. Introduction

Table 6-1 gives the corresponding symbol to each basic gateas it appears in the logic editor window as well as the logic description. In this description, the symbol & refers to the logical AND, | to Or, ~to INVERT, and ^ to XOR.

Name	Logic symbol	Logic equation
INVERTER	▷∘	Out=~in;
AND	▷	Out=a&b;
NAND	▷∘	Out=~(a.b);
OR	▷	Out=(a|b);
NOR	▷∘	Out=~(a|b);
XOR	▷	Out=a^b;
XNOR	▷∘	Out=~(a^b);

Table 6-1. The list of basic gates

2. Combinational logic

The construction of logic gates is based on MOS devices connected in series and in parallel. If two n-channel MOS switches are connected in series (Figure 6-1), the resulting switch connects ports C1 and C2 if both gates A and B are set to '1'. This yields an AND operator, represented by the symbol '&' in equation 6-1, where C12 is the logical variable which represents the connection between C1 and C2.
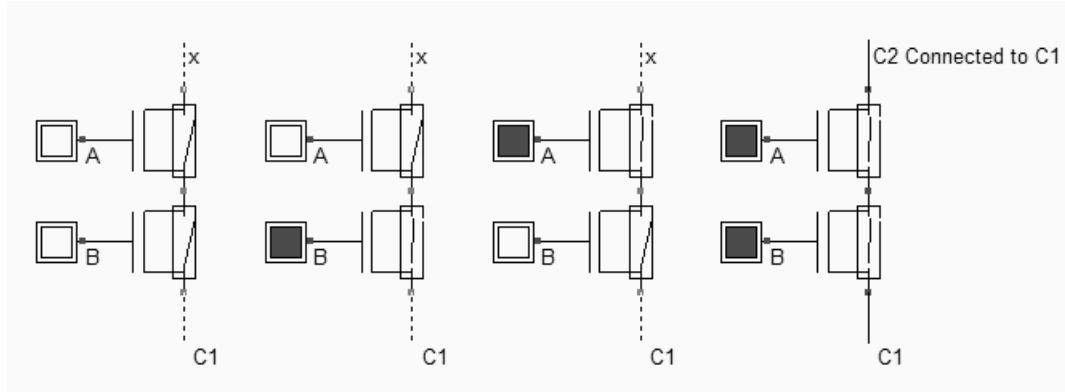
$$C_{12} = A \& B \qquad\qquad (6\text{-}1)$$

Fig. 6-1. Connecting N-channel devices in series creates a path between C1 and C2 when A and B are set to '1'

(BaseCmos.SCH)

When two nMOS switches are connected in parallel (Figure 6-2), the resulting switch is ON if either gates A and B are set to '1'. This yields an OR operator (described as '|' in equation 6-2).

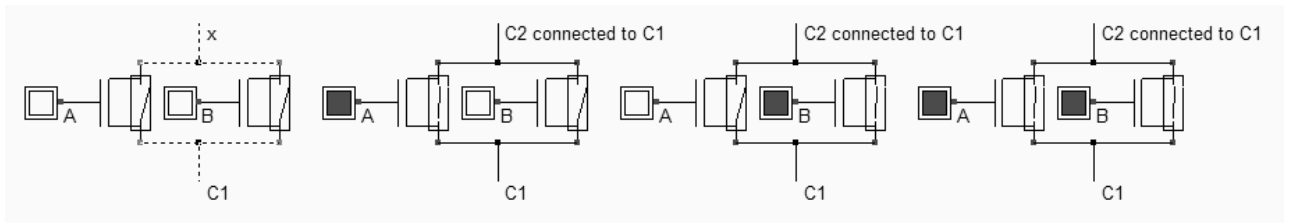$$C_{12} = A \mid B \qquad\qquad \text{(Equ. 6-2)}$$

Fig. 6-2. Connecting N-channel devices in parallel creates a path between C1 and C2 when either A or B are set to '1'

(BaseCmos.SCH)

Considering p-channel devices, we observe that two pMOS switches connected in series (Figure 6-3) behave as an AND between negative logic values: the resulting switch is ON if both gates A and B are set to '0'. The corresponding Boolean operator is as follows.

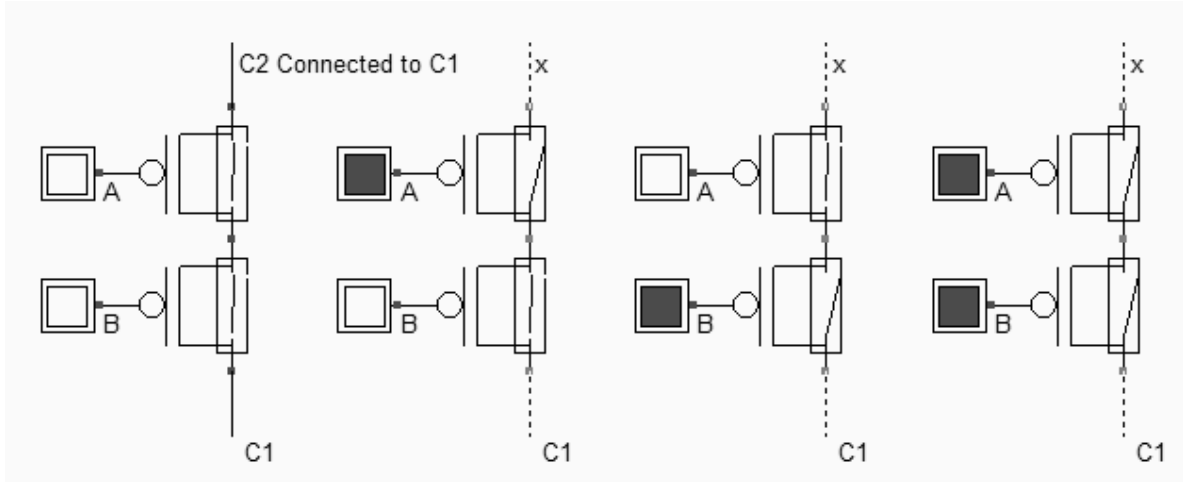$$C_{12} = \overline{A} \,\&\, \overline{B} \quad \text{(Equ. 6-3)}$$

Fig. 6-4. Connecting P-channel devices in series creates a path between C1 and C2 when A and B are set to '0'

(BaseCmos.SCH)

When two pMOS switches are connected in parallel (Figure 6-5), the resulting switch is ON if either gates A and B are set to '0'. The Boolean operator is described by equation 6-4.

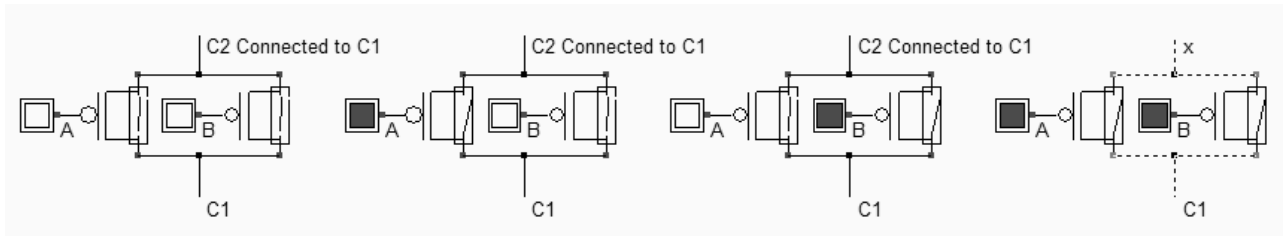$$C_{12} = \overline{A} \mid \overline{B} \quad \text{(Equ. 6-4)}$$

Fig. 6-5. Connecting P-channel devices in parallel creates a path between C1 and C2 when either A or B are set to '0'

(BaseCmos.SCH)

3. CMOS logic gate concept

The structure of a CMOS logic gate is based on complementary networks of n-channel and p-channel MOS circuits. Remember that the pMOS switch is good at passing logic signal '1', while nMOS switches are good at passing logic signal '0'. The operation of the gate has two main configurations:

- the nMOS switch network is closed, the output s=0 (figure 6-6 left)
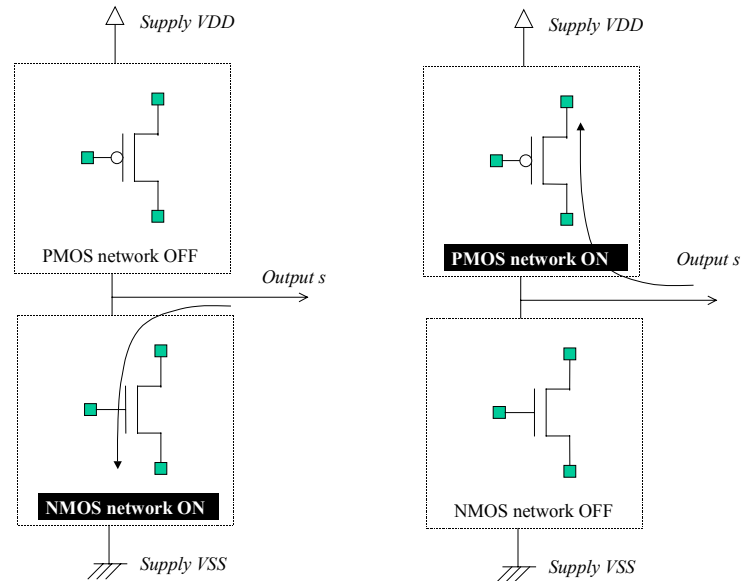- the pMOS switch network is closed, the output s=1 (figure 6-6 right)

Fig. 6-6. General structure of a CMOS basic gate

Using complementary pairs of nMOS and pMOS devices, either the lower nMOS network is active, which ties the output to the ground, either the upper pMOS network is active, which ties the output to VDD. In conventional CMOS basic gates, there should exist no combination when both nMOS and pMOS networks are ON. If this case happened, a resistive path would be created between VDD and VSS supply rails. The situation where neither nMOS and pMOS networks are OFF should also be avoided, because the output would be undetermined. These illegal situations are illustrated in figure 6-7.
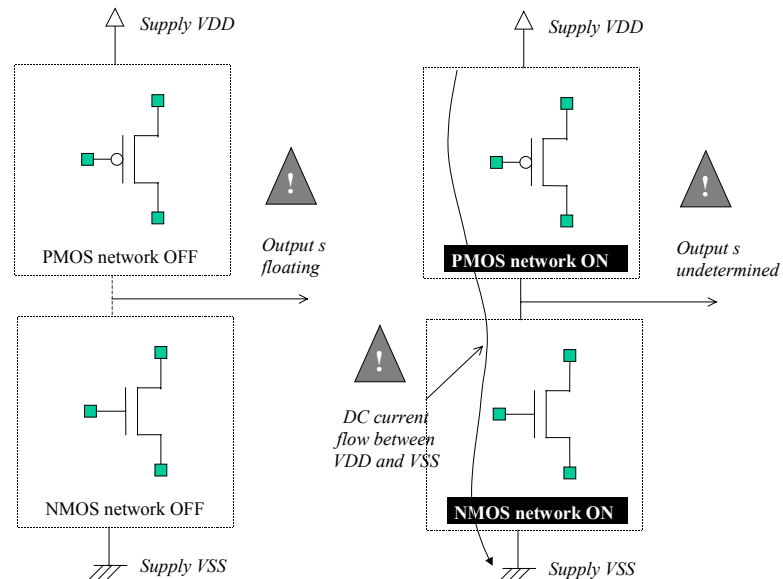

Fig. 6-7. Network configurations to be avoided: both OFF (left) and both ON (right).

4. The Nand Gate

Truth-table

The truth-table and logic symbol of the NAND gate with 2 inputs are shown below. The truth tables use 0 for logic level zero (usually 0V), 1 for logic level 1, (also called VDD, equal to 1.2V in 0.12μm technology) and x for unknown value. Some books [Uyemura] also include the z state, that is the high impedance value. We shall make no difference between the unknown and high impedance state in this book.

```
A B     Out
0 0     1
0 1     1
1 0     1
1 1     0
x 0     1
x 1     x
0 x     1
1 x     x
```
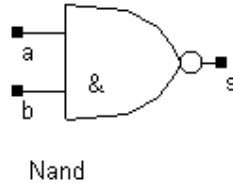

Figure 6-8. The truth table and symbol of the NAND gate

In DSCH2, the NAND gate is part of the symbol palette, which appears at initialization on the right side of the main window. Select the NAND symbol in the palette, keep the mouse pressed and drag the shape to the editing window at the desired location. Also add two buttons and one lamp as shown in figure 6-9. Add interconnects if necessary to link the button and lamps to the cell pins. The icon **Add a Line** is available for this purpose. Notice that the right click with the mouse enables the editing of an interconnect too. You may verify the logic behavior of the cell by a click on **Simulate → Start Simulation** or the icon **Run Simulation**. The logic level '0' corresponds to a white color, or to dot lines, and the logic level '1' is drawn in black, or with solid lines.
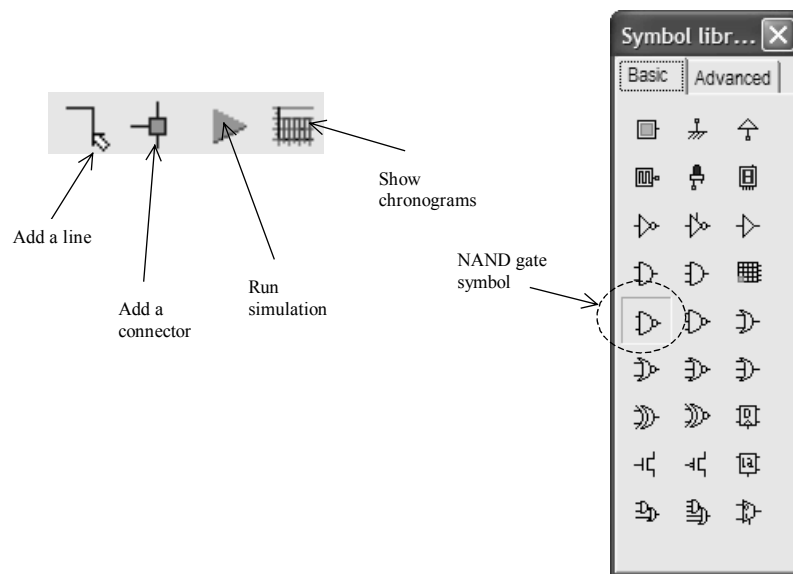

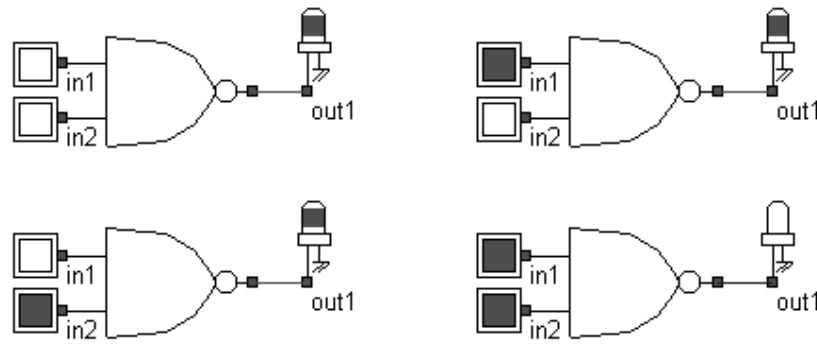*Figure 6-9. Editing commands for simulating the NAND gate*

Figure 6-10. The logic simulation of the NAND gate verifies the truth table (NandTruthTable.SCH)

Logic Design of the CMOS nand gate

In CMOS design, the NAND gate consists of two nMOS in series connected to two pMOS in parallel. The schematic diagram of the CMOS NAND cell is reported below. The nMOS devices in series tie the output to the ground for one single combination A=1 and B=1. For the three other combinations, the nMOS path is cut, but at least one pMOS ties the output to the supply VDD. Notice that both nMOS and pMOS devices are used in their best regime: the nMOS devices let "0" pass, the pMOS let "1" pass.

```
A B     nMOS  pMOS  Out
0 0     off   on    1
0 1     off   on    1
1 0     off   on    1
1 1     on    off   0
```
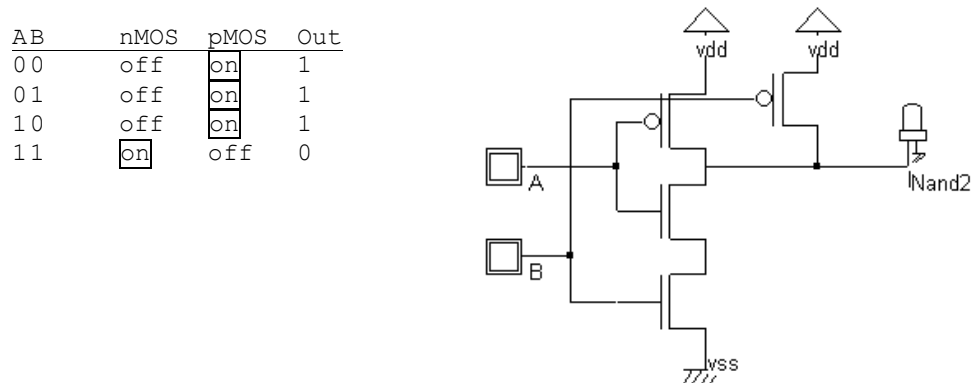

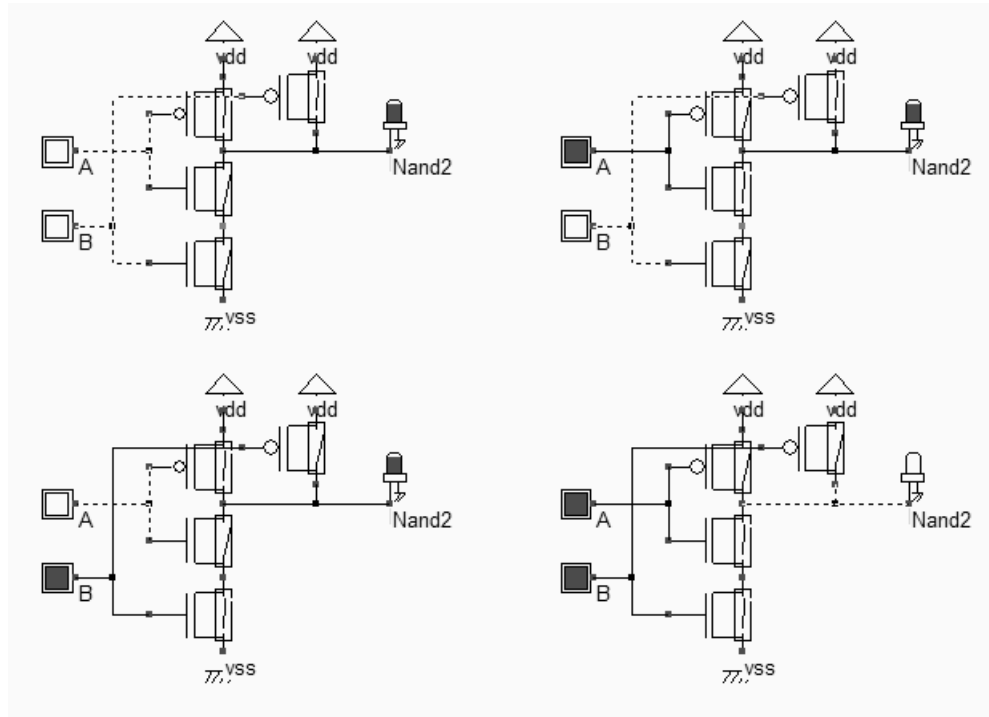*Figure 6-11. Schematic diagram of the CMOS NAND gate (NandCmos.SCH)*

Figure 6-12. The logic simulation of the NAND gate (Nand2Cmos.SCH)

Furthermore, the circuit of figure 6-12 eliminates the static power consumption when A and B are steady by ensuring that the situation "PMOS ON", "NMOS ON" never happens.

Automatic Generation of the NAND layout

Microwind features a built-in cell compiler that can generate the NAND gate automatically. In Microwind2, click on **Compile→Compile One Line.** Either select the line corresponding to the 2-input NAND description (Figure 6-13) or type the logical expression that describes the link between one output and two inputs. The '~' operator represents the NOT operator, the '&' symbol represents the AND operator.
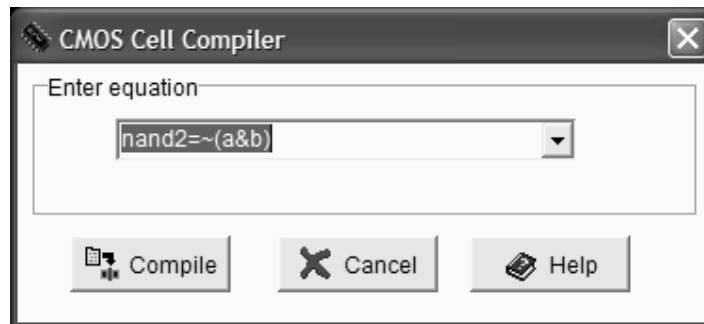

Figure 6-13. The CMOS cell compiler is used to generate a NAND gate

When you click "Compile", the layout of the NAND gate appears in the screen, as drawn in figure 6-14. The compiler has fixed the position of the VDD power supply in the upper part of the layout, near the p-channel MOS devices. The compiler has also fixed the ground VSS in the lower part of the window, near the n-channel MOS devices. The texts *A*, *B*, are placed in the layout, on the gates, and the text *nand2* is fixed on the output, at the upper location, near the contact. The layout generation is driven by the default design rules, corresponding in this case to a CMOS 0.12µm technology. Depending on the design rules and the technology generation the layout may look a little different. The implantation of the four devices is detailed in the schematic diagram at the right side of figure 6-14. The MOS devices have been arranged in such a way that the diffusion regions are merged, both for the two nMOS in series and the two pMOS in parallel. Sharing a common diffusion always leads to more compact designs, which saves silicon area and minimizes parasitic capacitance.
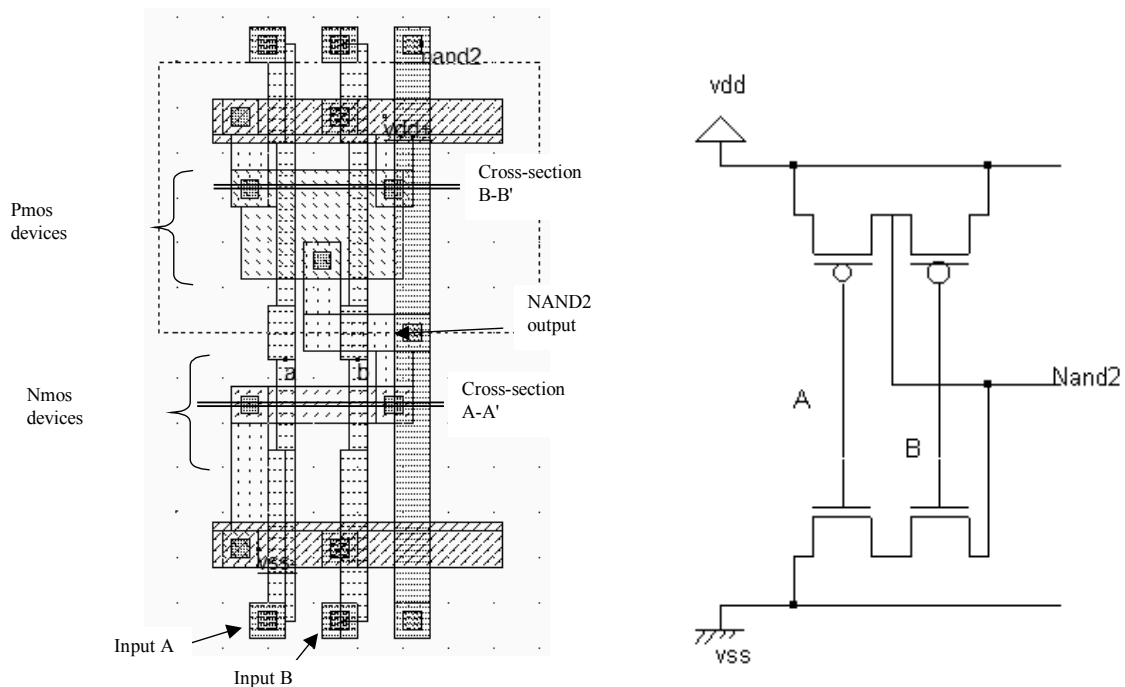

Figure 6-14. The layout of the NAND gate generated by the CMOS cell compiler (Nand2.MSK)

Inside the Nand gate

The 2D-process viewer is a useful tool to display the two nMOS in series and the two pMOS in parallel. Select the corresponding icon and draw a horizontal line in the layout in the middle of the nMOS channels, at location A-A' shown in figure 6-14. The figure below appears. The path from *vss* to the output *nand2* goes through two transistors connected in series, one controlled by A, the other controlled by B. In Figure 6-15, the output **nand2** may be tied to VDD either through the pMOS device controlled by A or through the pMOS device controlled by B. Notice the n-well under the pMOS devices, polarized to VDD.
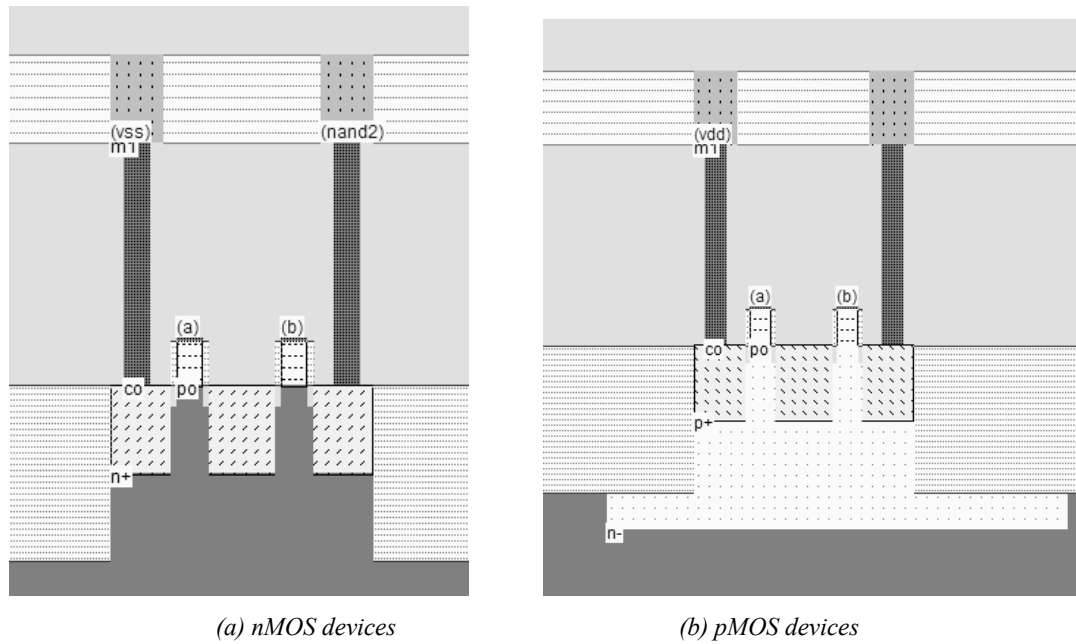
(a) nMOS devices   *(b) pMOS devices*

Fig. 6-15. The nMOS devices in series and the pMOS devices in parallel (Nand2.MSK)

.

Adding Simulation properties

The simulation icons add properties to the nodes. Properties are applied to the electric nodes of the circuit in order to serve as simulation guides. The list of properties required to perform the analog simulation of the NAND gate are shown in figure 6-16. First of all, the NAND gate should be supplied by a 0V (VSS) and 1.2V (VDD). The VDD and VSS properties are already placed in the layout by the CMOS cell compiler.
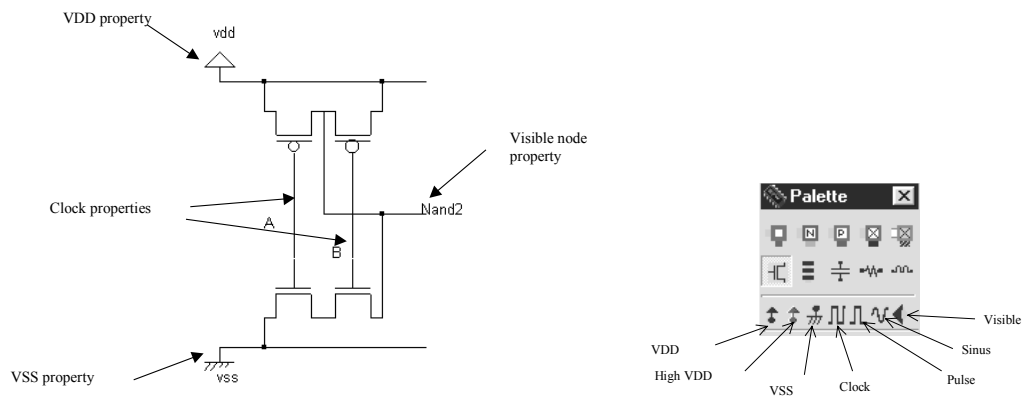

Fig. 6-16. Adding simulation properties to simulate the NAND gate (Nand2.MSK)

Secondly, clocks should be assigned to the input gates *A* and *B*. The clock icon is the fourth icon from the right in the palette menu. Simply activate the clock icon, and click in the letter *a* in the layout window. The following screen appears (Figure 6-17).
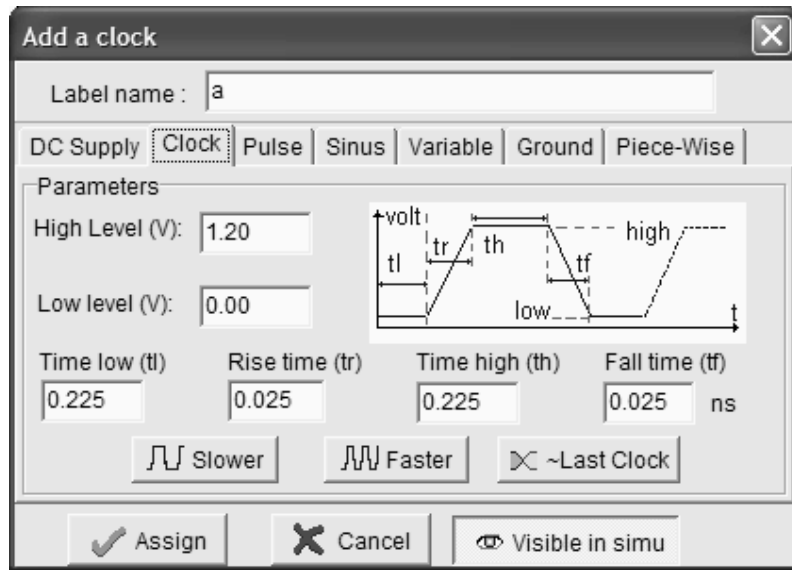
Fig. 6-17 Clock property added to node A (Nand2.MSK)

The parameters of the clock are divided as follows : time at low level (*tl*), rise time(*tr*), time at high level (*th*) and fall time(*tf*). All values are expressed in nanosecond (ns). Clock **Assign** to assign a clock to label *a*. Click again the clock icon, and this time click on label *b* in the layout. As you ask for a second clock, the period is automatically multiplied by two.

♦ You may alter level 0 and level 1 by entering a new value with the keyboard.

♦ To generate a clock which works in opposite phase, click **~Last Clock**.

♦ Use **Slower** to multiply the clock period by two.

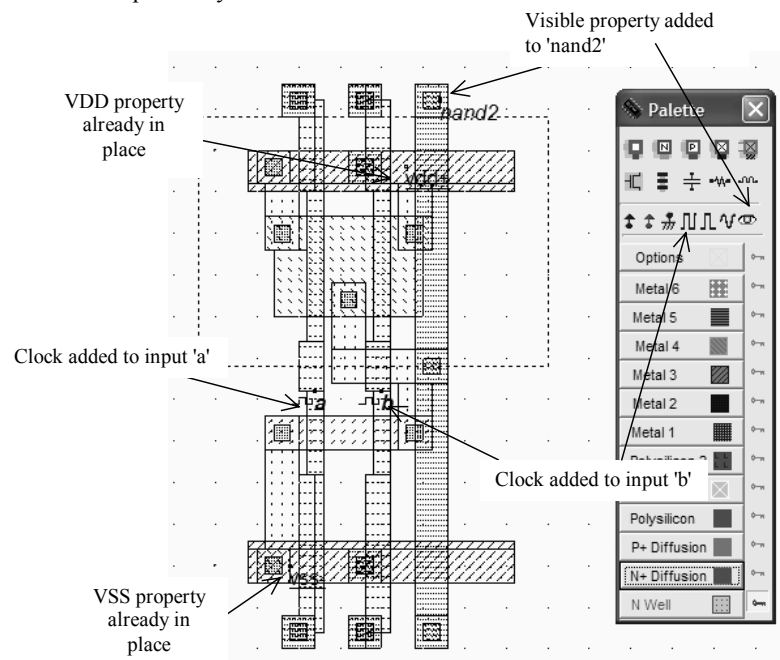♦ Use **Faster** to divide the clock period by two.

Fig. 6-18 Simulation properties added to the NAND gate (Nand2.MSK)

Finally, click on the "eye" in the palette, and click on the text *nand2* in the layout to make the chronograms of the node appear. Initially, all nodes are invisible. However, the nodes with clocks, impulse and sinus properties are subsequently made visible.

Electrical structure of the Nand Gate

The icon above is useful to get an insight of the electrical node structure of the layout. Select the icon and simply click inside the layout at the desired location. Information corresponding to the parasitic capacitance, resistance, as well as simulation properties are also displayed in a separate window, called navigator. In figure 6-19, the ground node and supply node are illustrated. In the Navigator window, the electrical properties of the node are displayed.
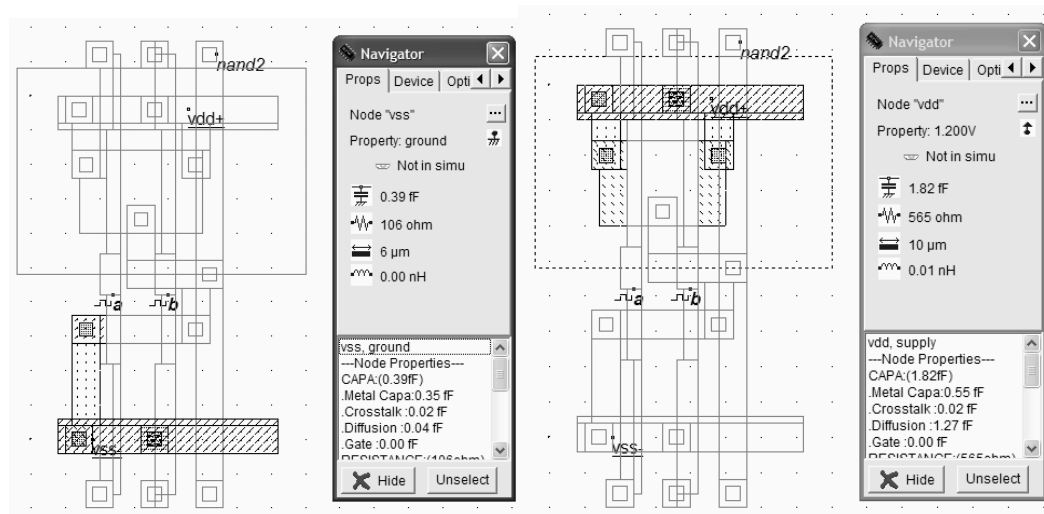

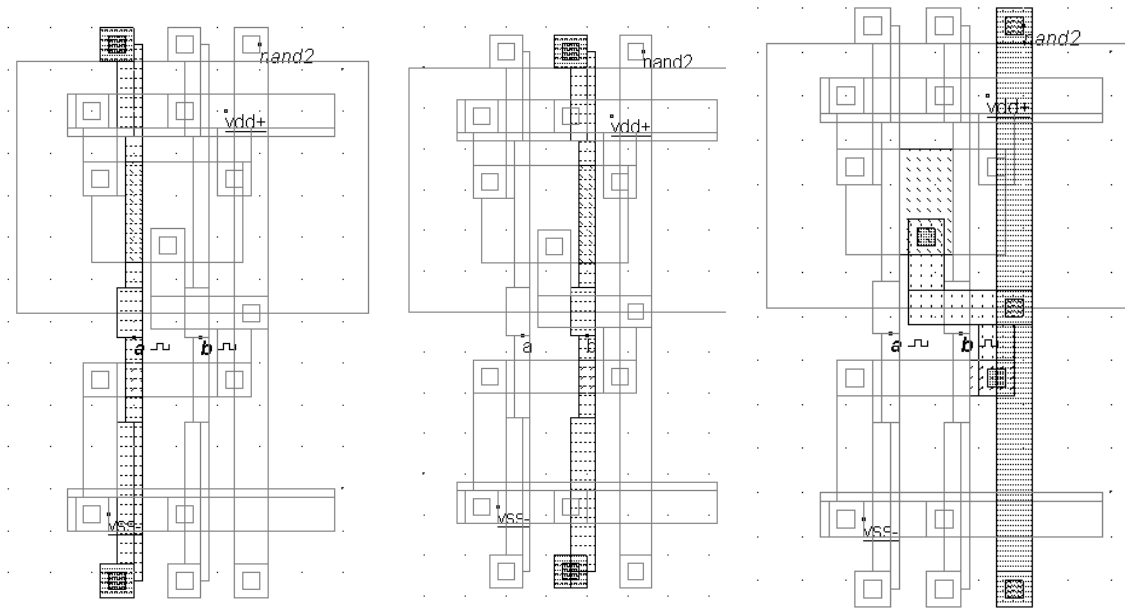*Fig. 6-19. The VSS node and VDD node with their associated properties (Nand2.MSK)*

Fig. 6-20. The structure of the input nodes A,B and the output node nand2 (Nand2.MSK)

The electrical connection of the inputs a,b and the output nand2 is revealed in figure 6-20. The inputs correspond to the nMOS gates, to a small portion of polysilicon between the pMOS and nMOS areas, and to the pMOS gates. The contacts situated on the lower and upper parts of the cell serve as a simple connection point to upper metal layers. The output node is quite complex. It consists of a drain area in the n-channel MOS region, connected to the central part of the pMOS diffusion area. The output is also connected to routing contacts up and down, thanks to a vertical metal bar in metal2.
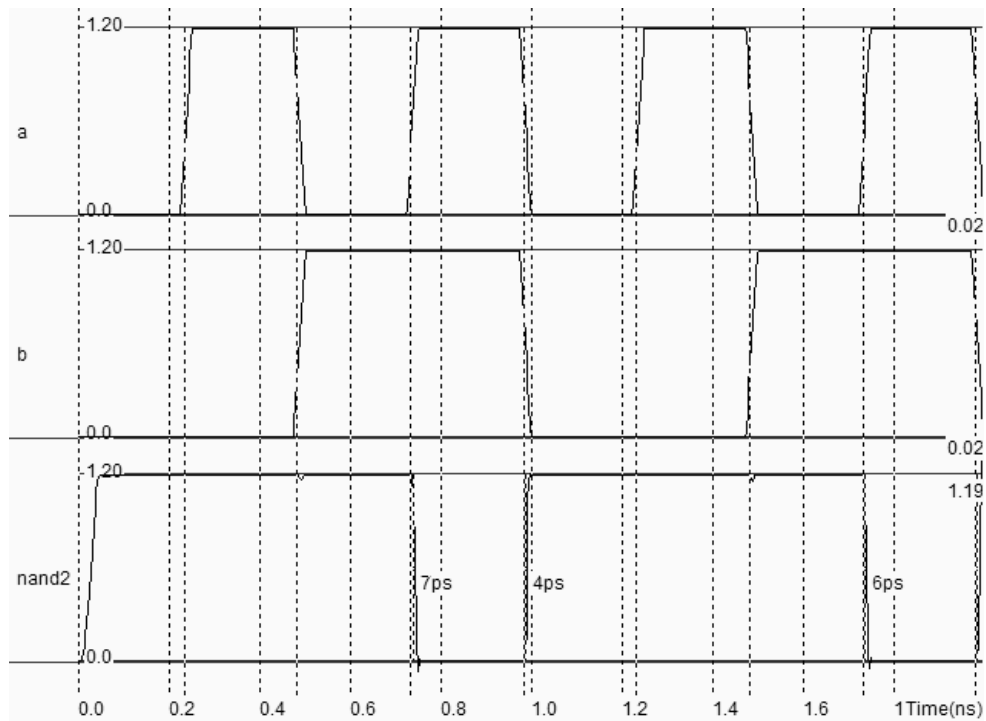
Analog Simulation of the NAND gate

Figure 6-21. Simulation of the NAND gate (NAND2.MSK)

The simulation of figure 6-21 is obtained by the command **Simulate → Run simulation**, or the above icon. We verify that *nand2* is equal to 0 when both *a*=1 and *b*=1, according to the truth-table, otherwise the output is at 1. The rise time and fall time are computed according to the following scenario. The simulator starts computing the delay when the selected signal, chosen here as *a*, crosses VDD/2. The simulator stops when the output nand2 also crosses VDD/2 (Figure 6-22). We should not emphasize too much the extremely small switching delay (7ps for the fall edge, 4ps for the rise edge). There are two reasons for such a high speed: one is the delay computation mode, based on VDD/2 which is always optimistic compared to a 10%-90% delay evaluation, the second is the absence of any output load, which gives a best-case estimation of the switching delay.
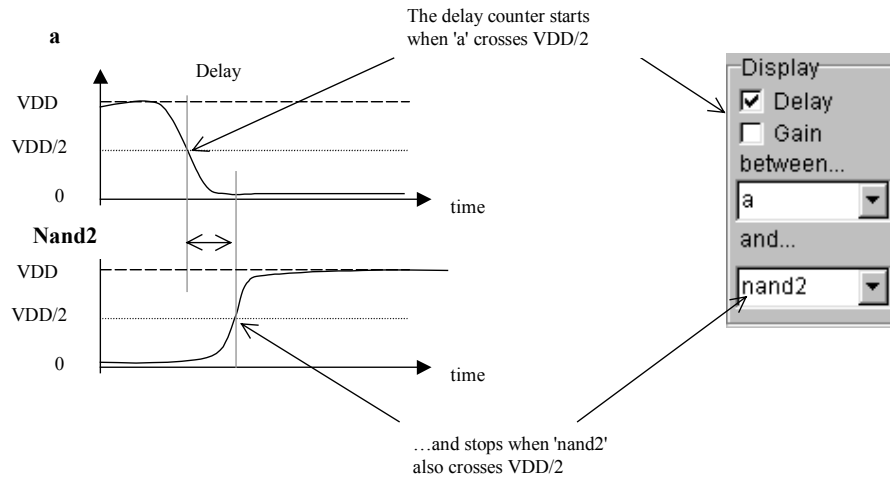
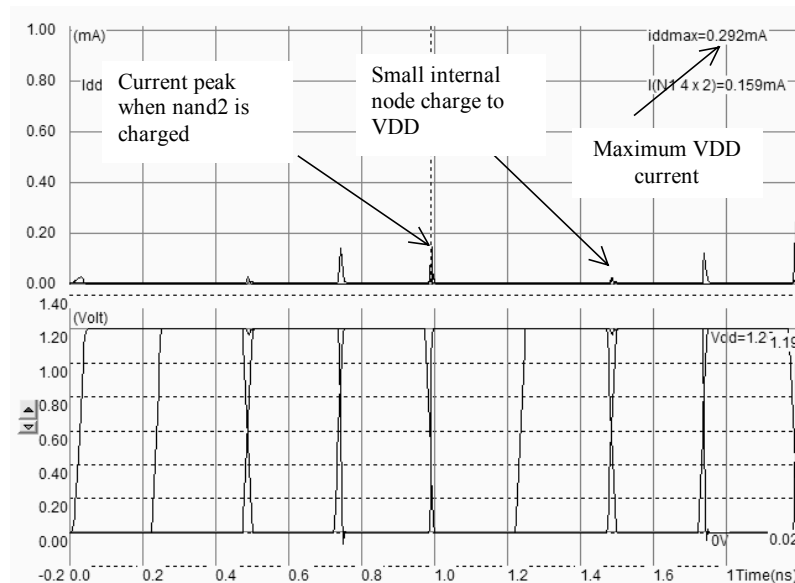*Figure 6-22. Delay computation in the simulation menu*

Figure 6-23. Current consumption on VDD supply line vs. time (Nand2.MSK)

Let us consider now another simulation mode, Voltage and Currents, accessible through the main menu via the command **Simulate → Run Simulation → Current, Voltage vs. Time**. The simulator displays all voltages in the lower window, and a selection of currents in the upper window (The IDD current, which is the sum of currents flowing from the supply VDD, and the current of one selected MOS device). The ISS currents can also be displayed. A transient current peak appears on IDD when the output node *nand2* is charged, as shown in figure 6-23. The current consumption is important only during a very short period corresponding to the charge of the output node.

Without any switching activity, the current is very small and cannot be seen accurately in linear scale. When looking at the same diagram in logarithmic scale (Assert **Scale I in log** in the simulator parameter window), we observe that the NAND gate consumes around 1nA of standby current. The default simulation model is Model 3, which does not

account accurately for leakage currents. The BSIM4 model, accessible by the command **Simulate → Using Model → BSIM4**, configures the simulation with BSIM4 model parameters.
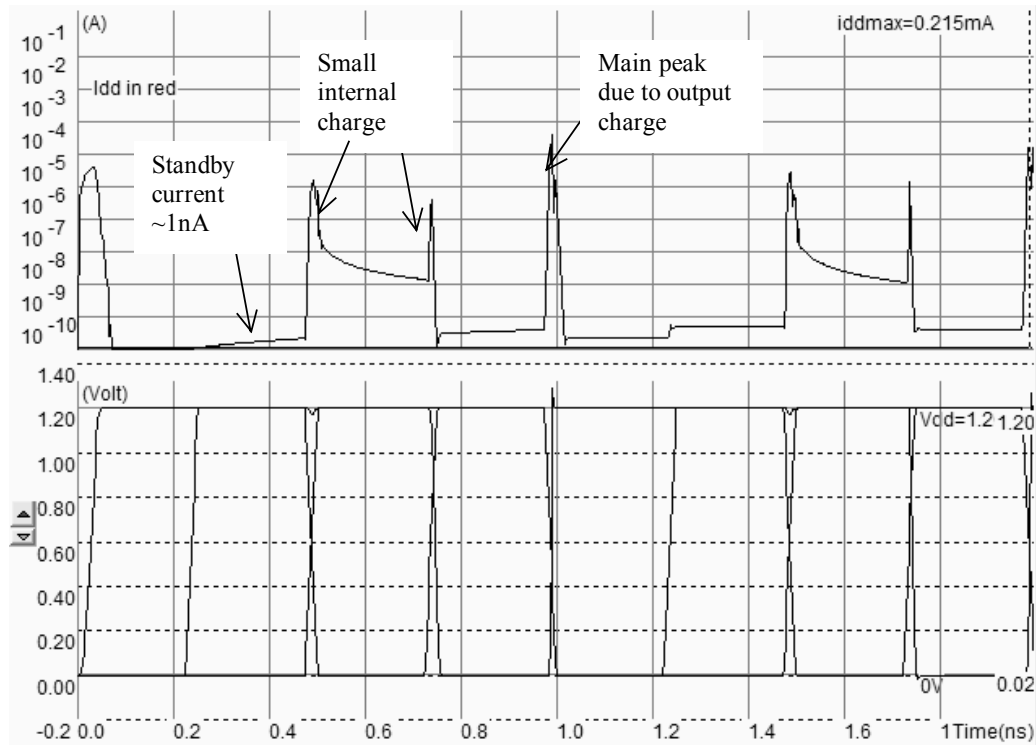

Figure 6-24. Current consumption in log scale showing the stand-by current (Nand2.MSK)

At time 0.5ns, the internal node situated in the n-channel MOS area between gates *a* and gate *b* is charged with a peak of current around 1µA. The area consists of a small n+ diffusion which is shown in figure 6-25. This diffusion region creates a N+/P-substrate junction, polarized in invert, which may be considered as a parasitic capacitance. This capacitance is charged and discharged under certain conditions. For example, at time 0.75ns, the short-circuit current resulting from a temporary situation where both n-channel MOS and p-channel MOS devices are ON, produces a current consumption also around 1µA.
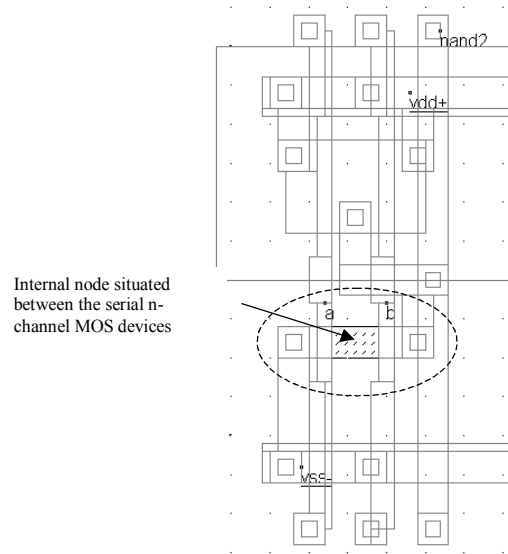
Internal node situated between the serial n-channel MOS devices

Figure 6-25. At time 0.5ns, an internal node is charged and induces a 1µA current on VDD supply line (Nand2.MSK)

MOS sizing

In the time-domain chronograms of figure 6-21, we observed a 7ps fall time and a 4ps rise time. This is due to the non-symmetrical structure of the NAND gate regarding low-to-high and high-to-low output switching. A fall edge of *nand2* is provoked by a discharge current IVSS through the two n-channel MOS in series, meaning an equivalent of 2xRN, where RN is the nMOS resistance when the channel is ON. In the case where both *a* and *b* are set to 0, the rise edge of *nand2* is provoked by the charge current IVDD through the two n-channel MOS in parallel. In this case, the equivalent resistance of the path is RP/2. Consequently, there is a switching speed difference due to the asymmetry inherent to the NAND gate structure. This asymmetry can be improved by resizing the nMOS or pMOS devices.
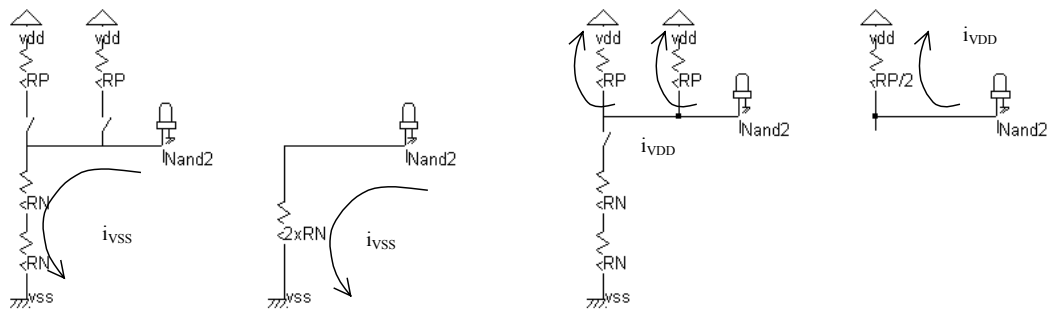

Figure 6-26. The unsymmetrical charge and discharge paths for the currents Ivdd and Ivss lead to a faster rise time of the output.

Optimization of the Nand Surface

The main advantage in joining the MOS devices diffusion whenever possible, rather than implementing all devices separately, is the silicon surface reduction, and consequently cost savings. A second advantage is the speed improvement. Joint diffusions lead to smaller areas, meaning lower parasitic capacitance, and thus shorter charge/discharge delays. The origin of the parasitic capacitance is mainly the N+/P-substrate junction capacitance due as the diode is polarized the other way round (P at low voltage VSS, N at higher voltage). Furthermore, the direct link between diffusions leaves space for metal routing.
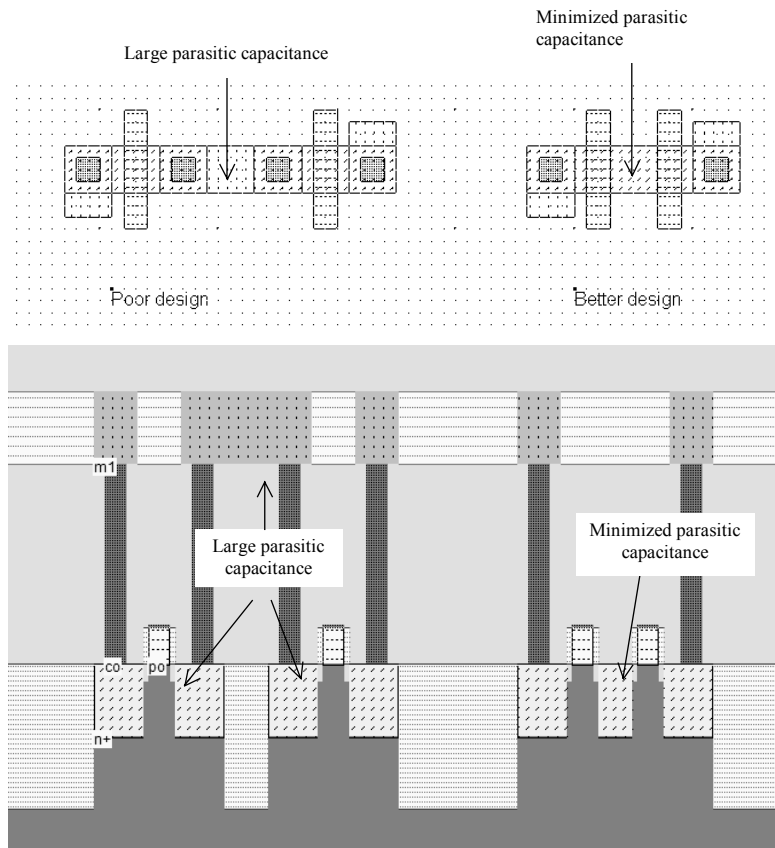

Fig. 6-27: Joined diffusions lead to compact designs and speed improvements (NandComp.MSK)

Optimum pMOS placement

There are two solutions in implementing the pMOS devices, according to the schematic diagram of figure 6-28. One solution consists in placing the pMOS with two connections to VDD (Left circuit), the second one with two connections to the output (Right circuit). Both solutions work fine. However, from the simulation of both structures, it can be seen that the left structure with minimum diffusion and metal connected to the output switches a little faster than the right structure which includes two diffusion areas.
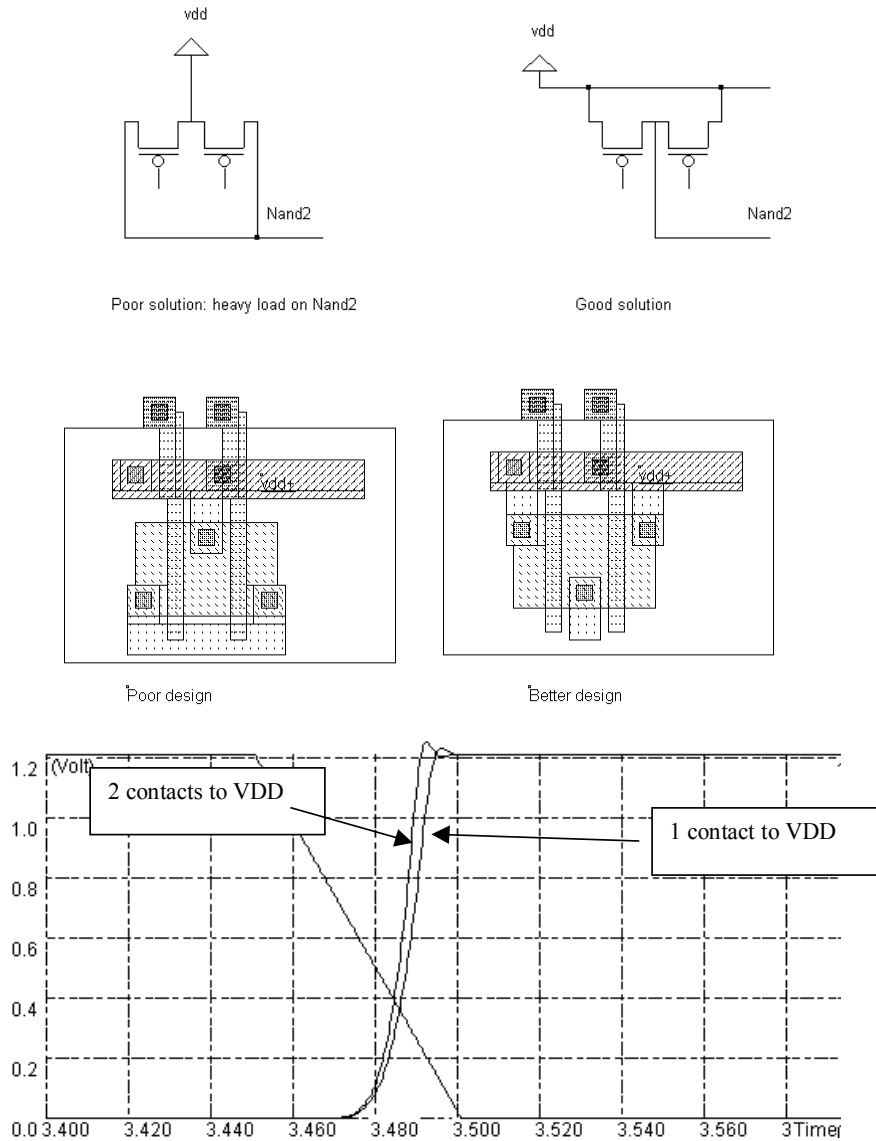
Fig. 6-28: A minimum path for the output is preferred for an optimum speed (NandComp.MSK)

Sub-micron vs. Deep sub-micron technology

In 0.8μm technology, the design rules concerning the design of a link between polysilicon and metal2 layers lead to the complicate and area consuming layout displayed on the left side of figure 6-29. The cross-section of this poly/metal2 contact is shown in figure 6-30. Notice that the via plug between metal1 and metal2 is slightly larger (7x7 lambda) than the contact from polysilicon to metal (6x6 lambda). In 0.12μm technology, the contacts have the same small dimensions (4x4 lambda), and can be stacked on top of each other. Consequently, the routing can be more dense and the cell design can be compacted.
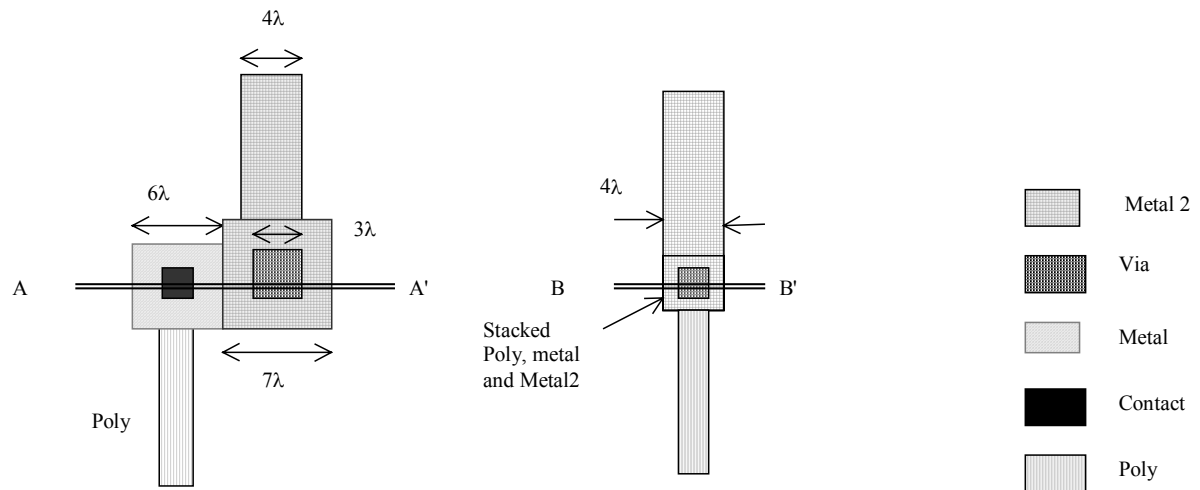
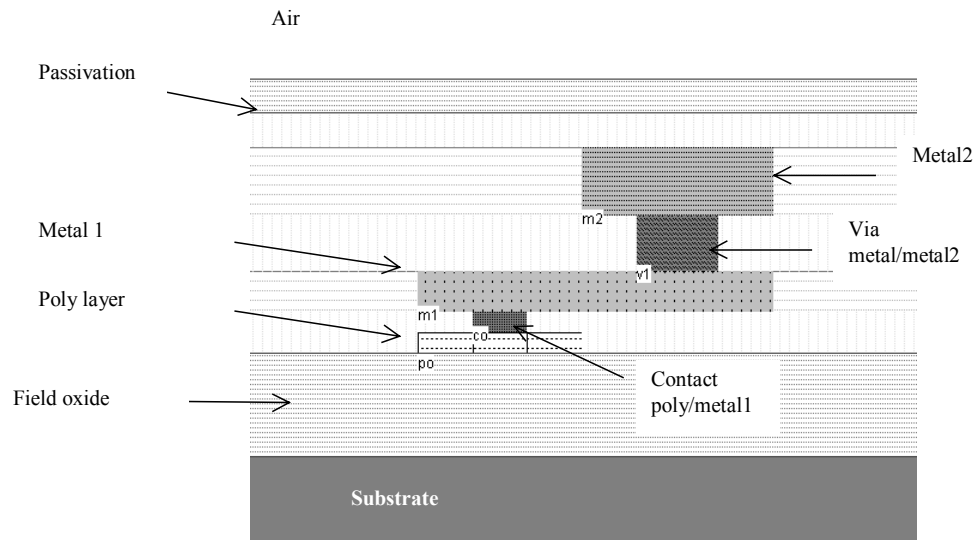*Fig. 6-29 : Sub-micron via (left) and deep sub-micron stacked vias (right)*

Fig. 6-30 : 2D cross-section of the poly/metal12 contact in 0.8µm (A-A' of figure 6-29)
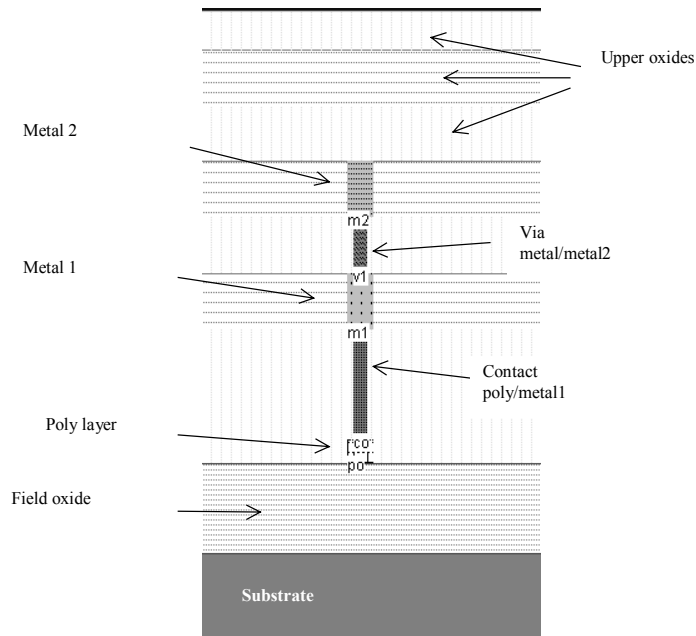
Fig. 6-31 : 2D cross-section of the poly/metal12 contact in 0.12μm (B-B" of figure 6-29)

The benefits of deep sub-micron technology for a complete logic gate are illustrated in figure 6-32. On the left side, the NAND gate compiled with the 0.8μm technology parameters (cmos08.rul) is drawn. On the right side, the same NAND gate compiled with the 0.12μm technology parameters (cmos012.rul) is reported. Although both layout designs are in lambda scale, the gain in surface is obvious.
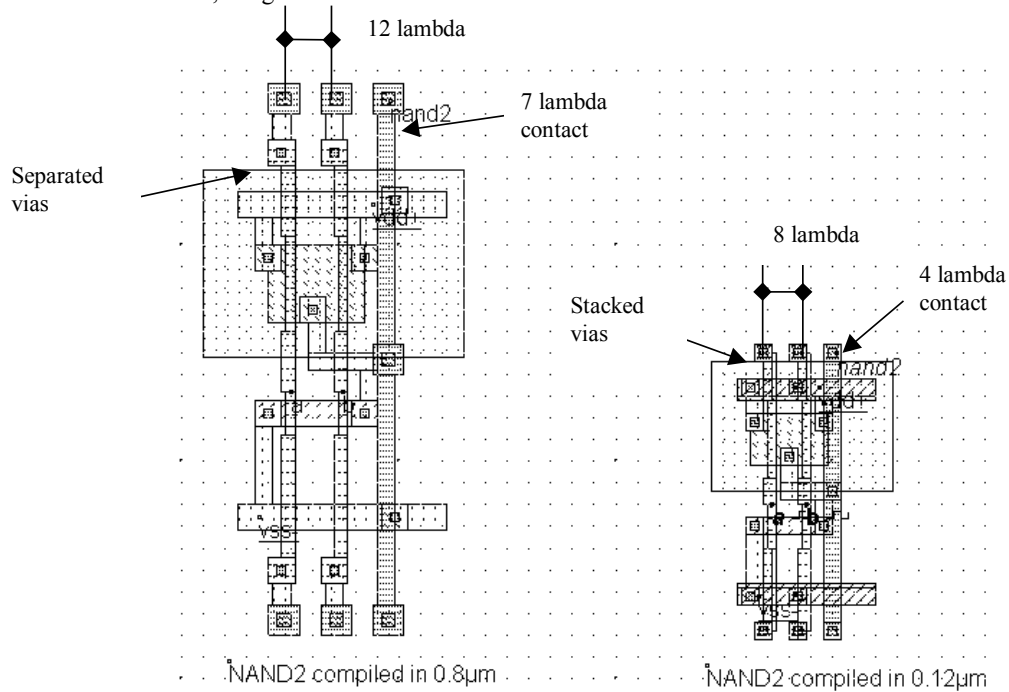

Fig. 6- 32: silicon surface improvement in deep-submicron technology (NandCompo.MSK)

The major reasons for this improvement are:

- a smaller routing pitch: 12 lambda in 0.8µm, 10 lambda in 0.35µm, 8 lambda in 0.12µm thanks to more aggressive rules, specifically for the size of contacts.
- the possibility to stack via: 2 routing pitches are required in 0.8µm to connect polysilicon to metal2, one single pitch is required starting 0.35µm technology.

3-input NAND Gate

The schematic diagram of the n-input NAND gate is derived from the architecture of the NAND2 gate, with n-channel MOS devices in series and p-channel MOS devices in parallel. Using the built-in cell compiler, you can generate the 3 or 4 input NAND gate. The command is **Compile→Compile One Line.** Enter the text `nand3=~(a&b&c)` and click **Compile**.
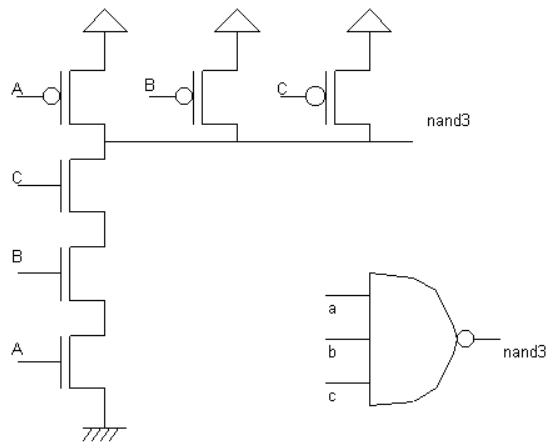

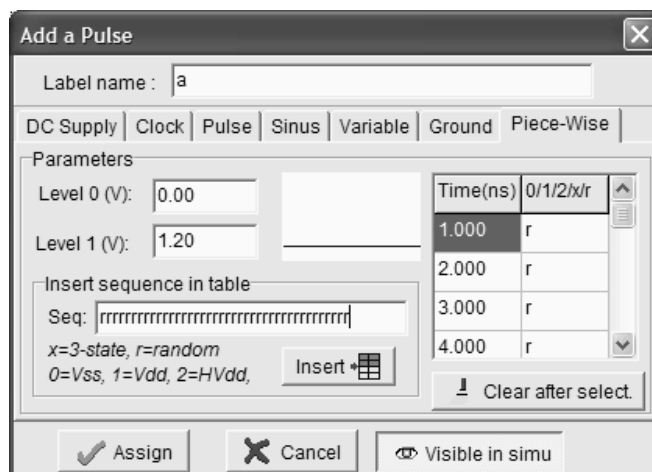*Fig. 6-33: Symbol and implementation of the 3-input NAND gate (nand3Cmos.SCH)*

Fig. 6- 34: Editing the pulse properties to generate a random logic signal (Nand3Rand.MSK)

A convenient way to observe the switching performances of the NAND3 gate is to stimulate the inputs using random patterns and observe the simulation results in **eye diagram** mode. The way to proceed consists in changing clock

properties into pulse properties, and to assign the value 'r' in place of '1' or '0', which is understood as a random logic value. The procedure is as follows: enter an "rrrrrrrrrrrrrrrr" sequence and click **insert** to transfer these values into the table. Then click **Assign**. The result is a series of random logic values each 1ns, as shown in the window reported in figure 6-35. The inputs (Figure 6-35) consist of a random series of logic values. Click **Reset** and a new and uncorrelated series of samples will be generated. The output change may be seen in this particular example at time 3ns.
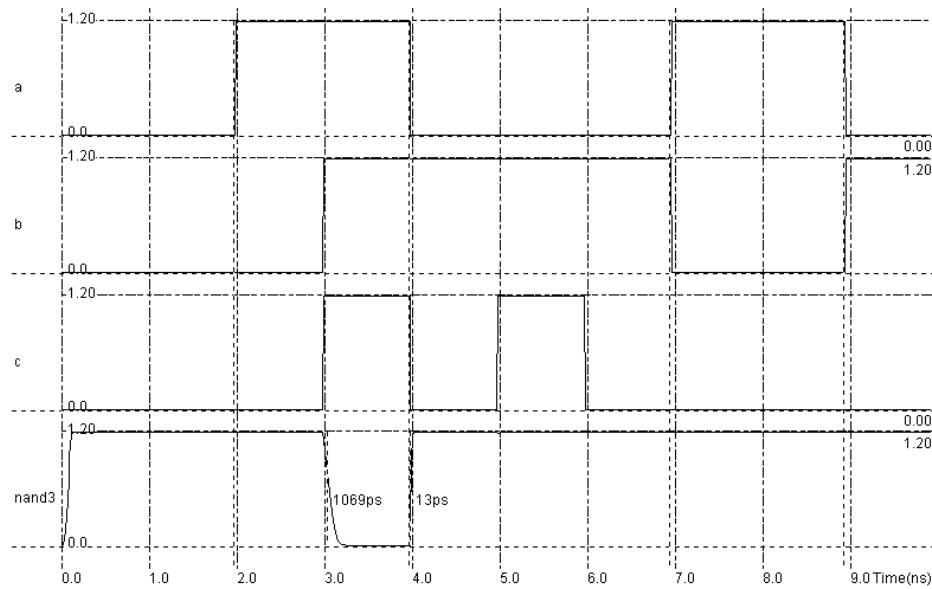

Fig. 6 35: Random stimulation of the 3-input NAND gate (nand3Rand.MSK)

Eye Diagram Simulation

The eye diagram is created by a press on the "**eye diagram**" in the choice bar situated in the lower menu of the simulation window. Each time the input data changes, the simulation chronograms of the output are replaced in the left corner of the simulation window. This makes it possible to compare the switching delays. What we observe is a constant fall delay when discharging a 10fF load. Depending on the value of A,B and C, the rise delay may change significantly, as one, two or three p-channel MOS devices may be put in parallel to charge the 10fF load. Notice that the case A,B,C "on" simultaneously is a very rare event that has not been observed in the figure 6-36.
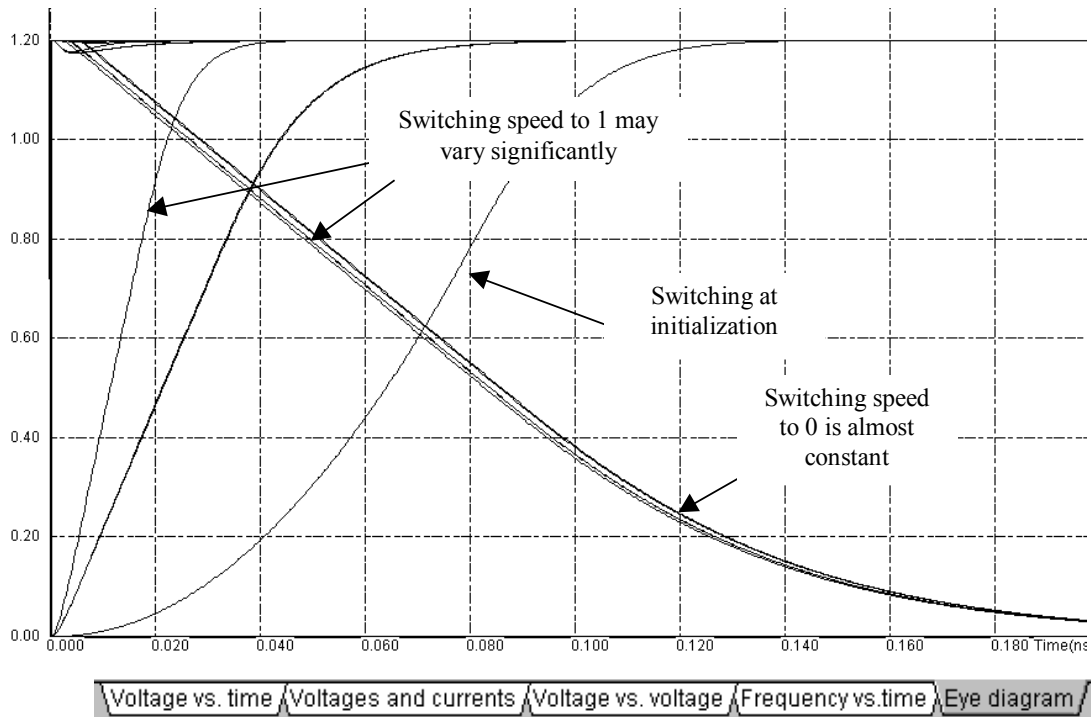
Fig. 6- 36: Eye diagram of the random simulation showing significant variations of the rise delay in the 3-input NAND gate (nand3Rand.SCH)
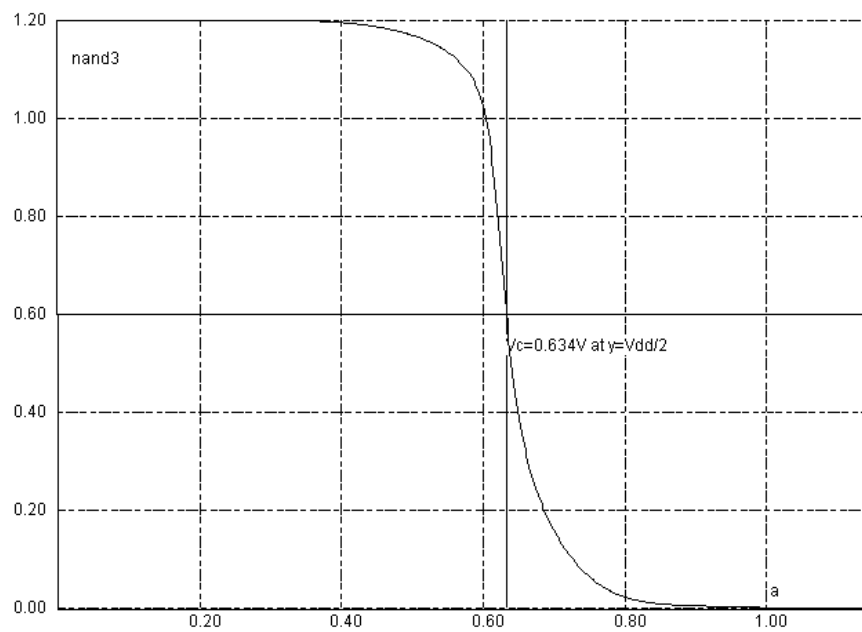

Fig. 6- 37: Static characteristics of the 3-input NAND gate (Nand3.MSK)

The switching characteristics of the 3-input **nand** gate may be improved by reducing the width of the p-mos devices and increasing the width of n-mos devices. However, the commutation point of the cell is not significantly different from

VDD/2. As seen in figure 6-37, the transfer characteristics simulated using BSIM4 between input a and the output *nand3* exhibit a commutation point *Vc* near 0.63V which is close to VDD/2.

5. The AND gate

The truth-table of the AND gate is reported in figure 6-38. In CMOS design, the AND gate is the sum of a NAND gate and an inverter. More generally, the negative gates (NAND, NOR, INV) are simpler to implement in CMOS technology, than the non-negative gates (AND, OR, Buffer). In the logic simulation at switch level (Figure 6-39), the NAND output serves as the input of the inverter output stage, and verifies the truth-table.

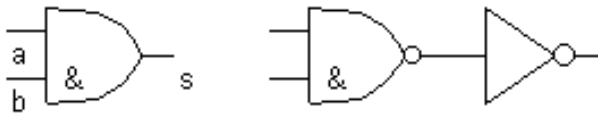| A	B	AND2
0	0	0
0	1	0
1	0	0
1	1	1
X	0	0
X	1	X
0	X	0
1	X	X

Fig. 6- 38 Truth-table of the AND gate

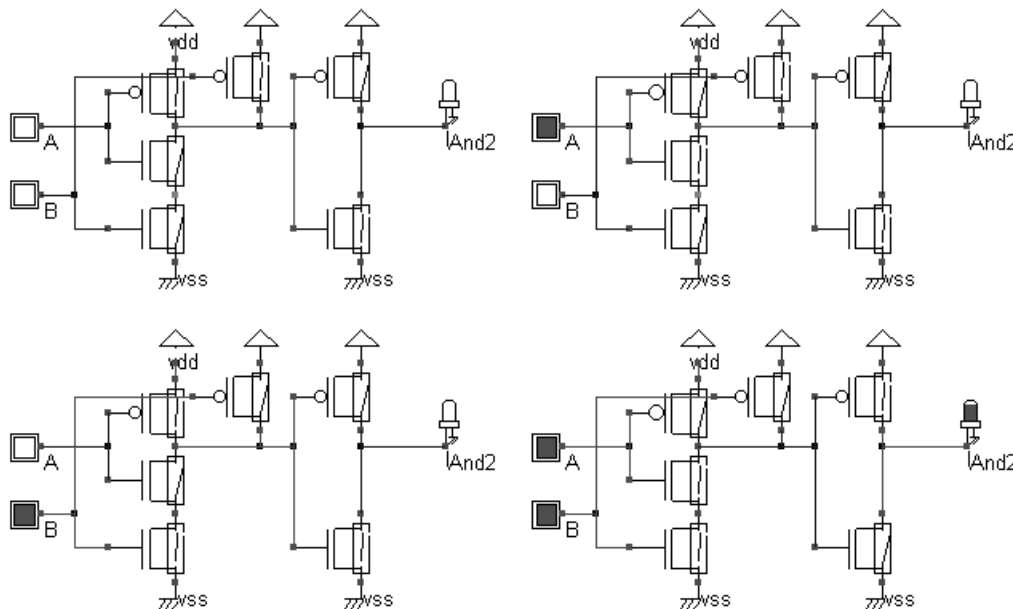*Fig. 6- 39 The switching details of the 2-input AND gate (And2Cmos.SCH)*

The layout of the *and* cell may be compiled using the command **Compile → Compile One Line.** Notice that a more compact layout of the AND cell may be found by joining the diffusions of the NAND and the Inverter cells. This leads to several design rule errors that must be corrected by moving contacts, decreasing polysilicon width, etc… The final arrangement (Figure 6-40 right) saves one horizontal routing pitch, that is 8 lambda.
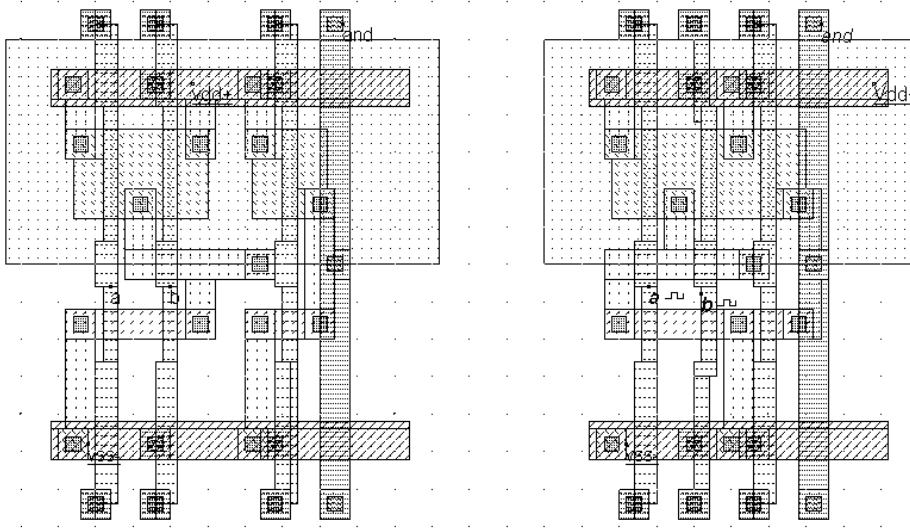

Fig. 6-40: Compiled layout of the AND gate (left) and manual arrangement (right) to achieve a more compact cell (And2.MSK)

The simulation described in figure 6-41 concerns the 2-input AND gate with a 10fF load on the output *and*. We also observe the internal node named *nand2*. We confirm that the *and2* output reacts with a certain delay, due to a cascade of two logic cells. However, the major cause of delay is the 10fF load on the output.
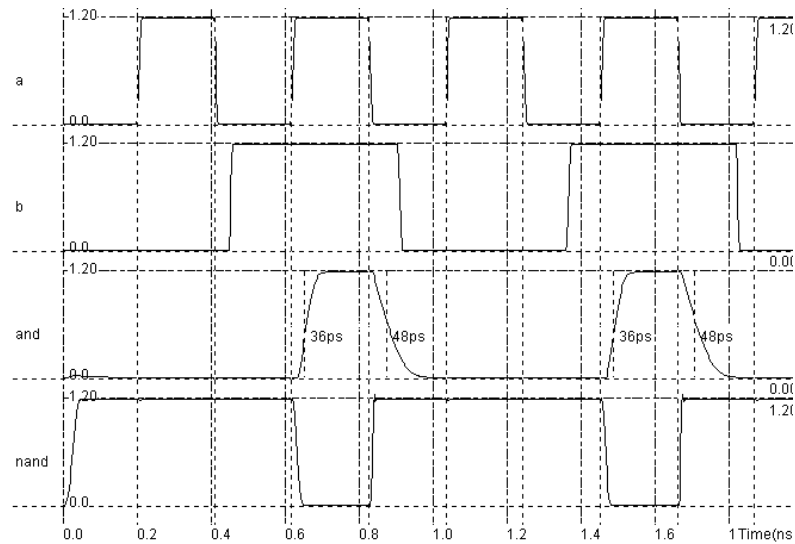

Fig. 6-41: Simulation of the AND gate with a 10fF load (And2.MSK)

6. The NOR Gate

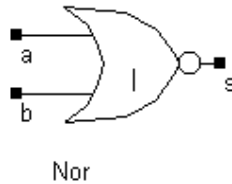| A B	Out
0 0	1
0 1	0
1 0	0
1 1	0
x 0	x
x 1	0
0 x	x
1 x	0

Figure 6-42. The truth table and symbol of the NOR gate

In CMOS design, the NOR gate consists of two nMOS in parallel connected to two pMOS in series. The schematic diagram of the CMOS NOR cell is reported below. The nMOS in parallel ties the output to the ground if either A or B are at 1. When both A and B are at 0, the nMOS path is cut, but the two pMOS devices in series tie the output to the supply VDD.

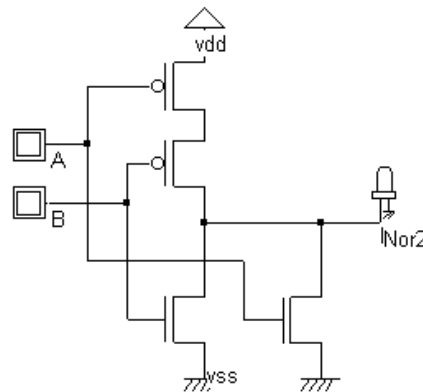| A B	nMOS	pMOS	Out
0 0	off	on	1
0 1	on	off	0
1 0	on	off	0
1 1	on	off	0

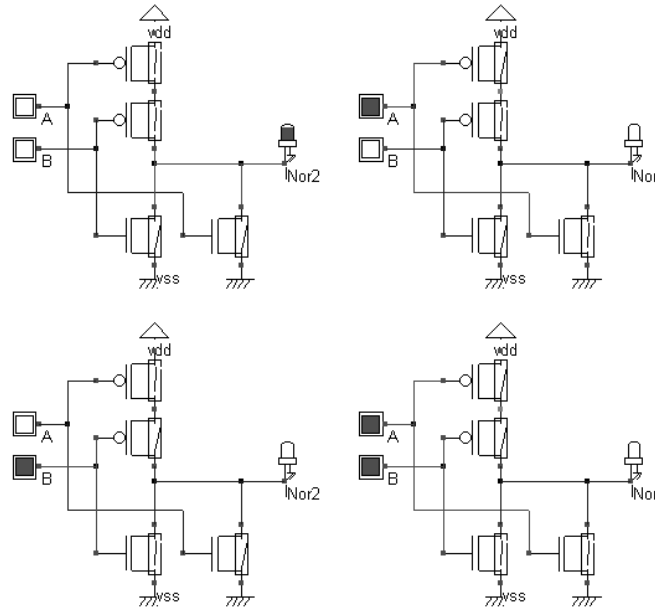*Figure 6-43. Schematic diagram of the CMOS NOR gate (NorCmos.SCH)*

Figure 6-44. The logic simulation of the NOR gate verifies the truth table (NorCmos.SCH)

Using the cell compiler we generate the NOR gate from its VERILOG description by entering the equation `Nor2=~(a|b)`. The '~' operator represents the NOT operator, the '|' symbol represents the OR operator. The compiled layout of the NOR gate appears in the screen, as drawn in figure 6-45. The compiler has fixed the position of the VDD power supply in the upper part of the layout, near the p-channel MOS devices. The compiler has also fixed the ground VSS in the lower part of the window, near the n-channel MOS devices. The texts *A*, *B*, and *out* have been fixed to the layout as for the NAND gate.
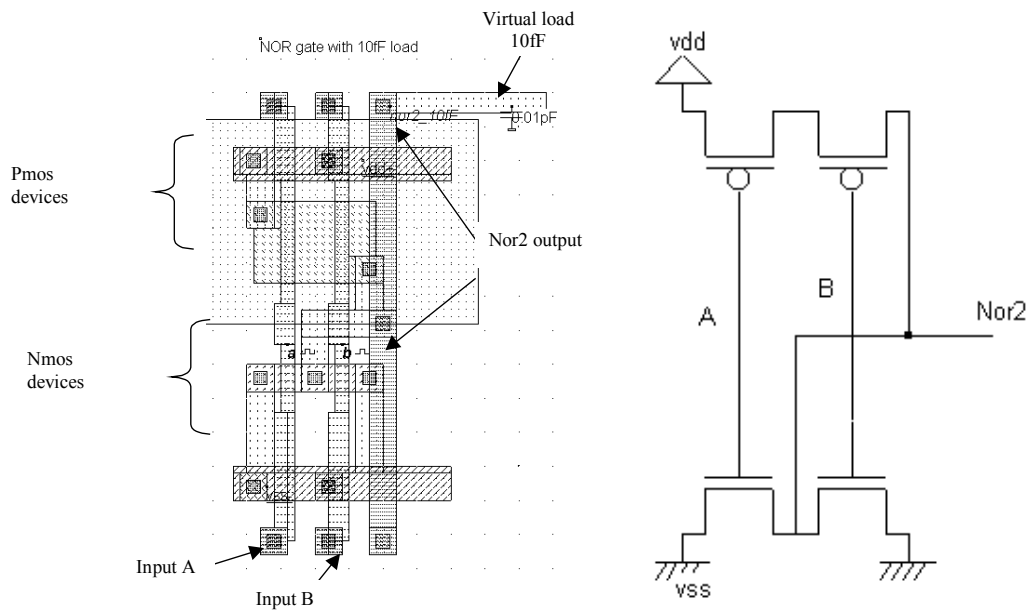

Figure 6-45. The layout of the NOR gate generated by the CMOS cell compiler (Nor2.MSK)

The simulation of figure 6-46 is obtained by the command **Simulate → Run simulation**. We verify that **nor2** output signal is equal to 0 when either *a*=1 or *b*=1, according to the truth-table. The rise time and fall time are shown for a nor2 gate without any load connected to its output or with a virtual 10fF load. We see that the 10fF slows down the switching speed considerably. Furthermore, the rise time is significantly larger than the fall time, due to p-channel MOS in series.
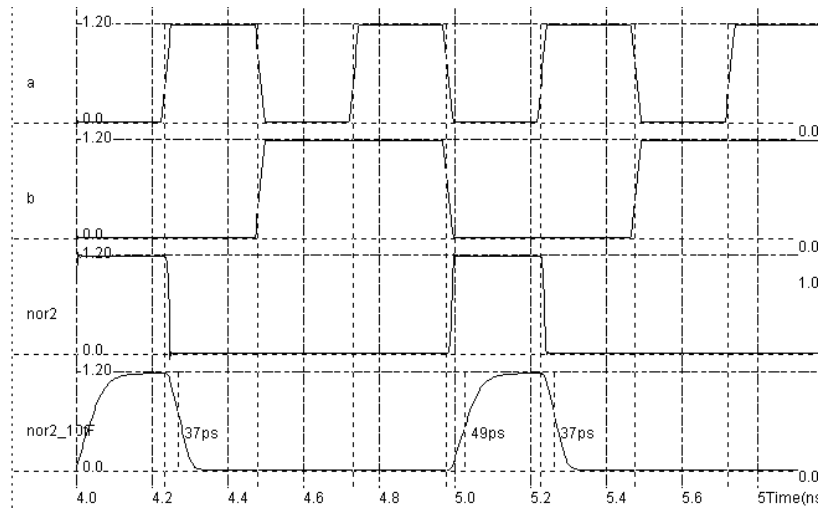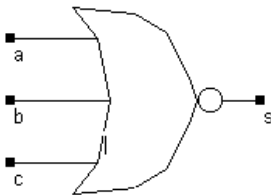

Figure 6-46. Simulation of the NOR gate with and without loading capacitance (Nor2.MSK)

The NOR3 Gate

The 3-input NOR gate circuit may be deduced from the 2-input NOR gate design, but a problem rises. The pMOS devices in series lead to very poor output node charge to VDD, and consequently poor rise time performances, as shown below. Meanwhile, the nMOS devices in parallel provoke a very efficient path for discharge to ground, meaning a short fall time. This non-symmetrical behavior can be tolerated up to a certain limit. This is why the n-input NOR gates (with n=3,4..) are rarely used. NAND gate based designs are preferred because of the pMOS in parallel which naturally compensate the poor hole mobility of their channel, producing symmetrical switching characteristics.
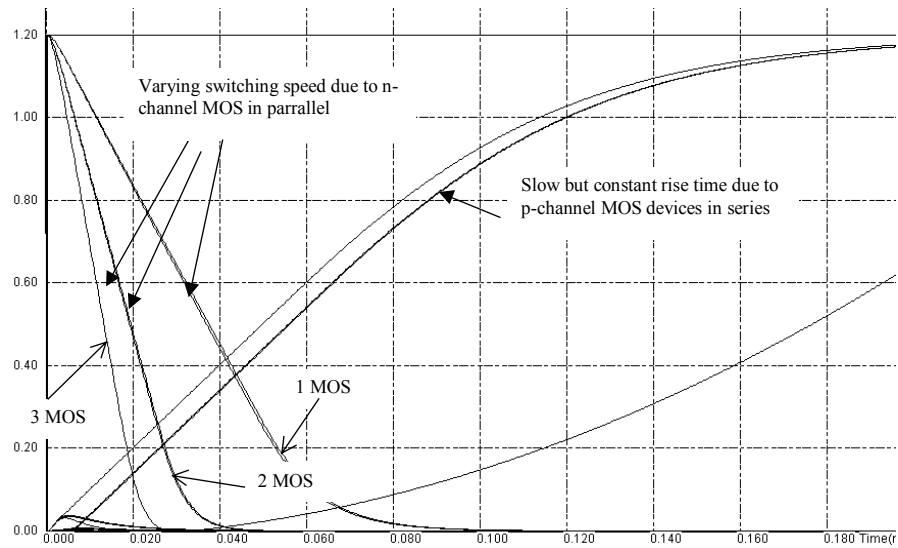
Fig. 6-47 : the non-symmetrical behavior of the NOR3 gate (Nor3.MSK)

7. The OR Gate

A B	Out
0 0	0
0 1	1
1 0	1
1 1	1
x 0	x
x 1	1
0 x	x
1 x	1

OR2 Gate

Figure 6-48. The truth table and symbol of the OR gate

Just like the AND gate, the OR gate is the sum of a NOR gate and an inverter. The implementation of the OR2 gate in CMOS layout requires 6 transistors. An arrangement may be found to obtain continuous diffusions on n-MOS regions and pMOS regions, as illustrated in figure 6-49.
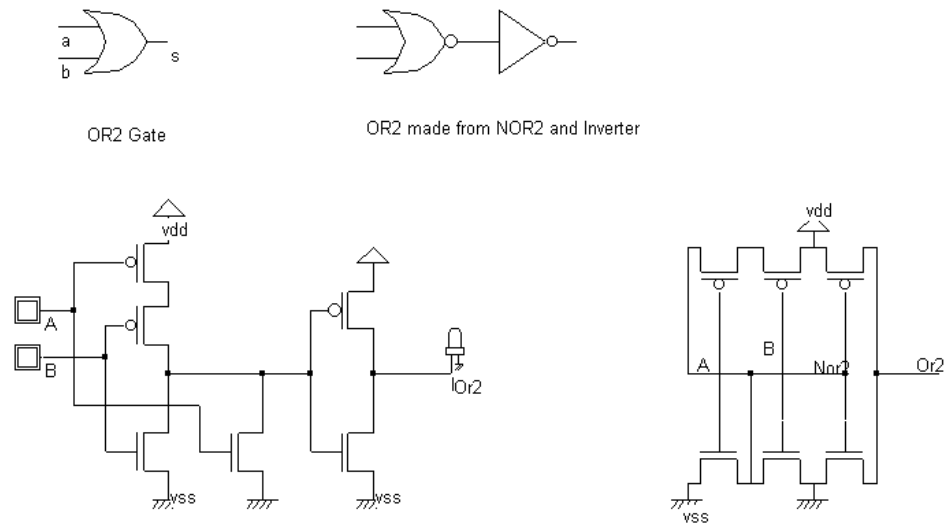
Fig. 6-49 : The schematic diagram of the OR gate and its CMOS structure (Or2.SCH)

The layout generated by the VERILOG compiler (or2=a|b) is shown in figure 6-50. A more compact version of the OR2 gate is also provided (Figure 6-51 right) where the supply pins VSS and VDD are shared by the NOR2 and Inverter cells.
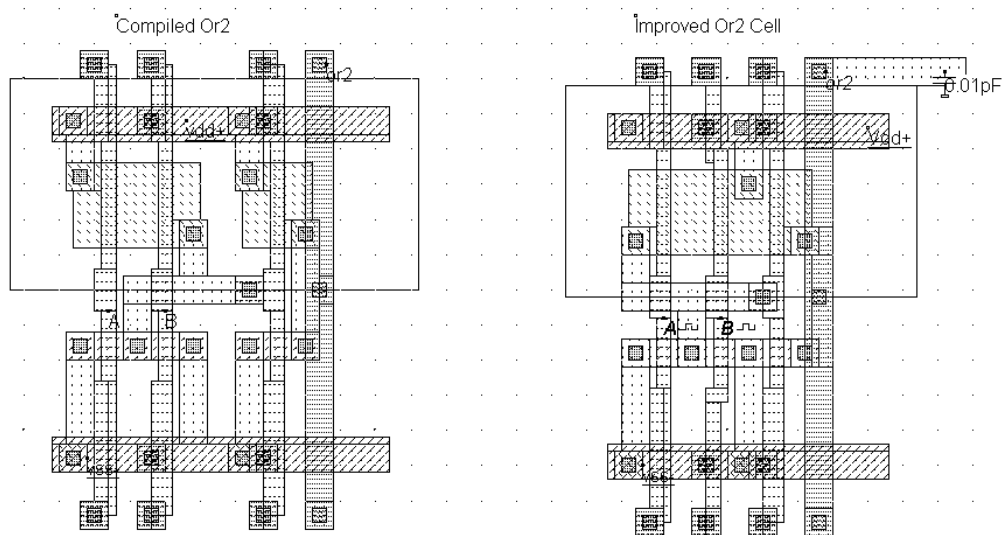

Fig. 6-50 : Layout of the compiled OR2 cell (left) and a more area-efficient manual arrangement (right) (OR2.MSK)

8. The XOR Gate

A	B	XOR2
0	0	0
0	1	1
1	0	1
1	1	0
X	0	X

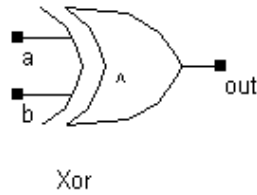| X	1	X
0	X	X
1	X	X

Xor

Fig. 6-51 : Truth table and symbol of the XOR gate

The truth-table and the usual symbol of the CMOS XOR gate are shown in Figure 6-51. There exist many possibilities to implement the XOR function into CMOS, which are presented in the following paragraphs.

A poor design

The least efficient design, but the most forward, consists in building the XOR logic circuit from its Boolean equation given by equation 6-4. The direct translation into primitives leads to a very complicated circuit shown in figure 6-52. The circuit requires two inverters, two AND gates and one OR gate, that is a total of 22 devices. Furthermore, the XOR gate has a critical path based on 6 stages of CMOS gates, which slows down the XOR switching response considerably.

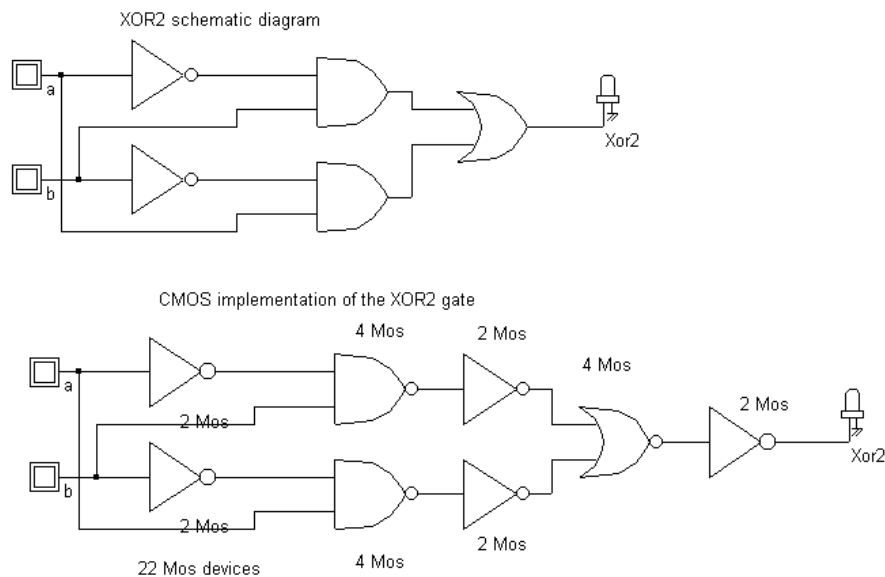$$xor = A\overline{B} + B\overline{A} \qquad\qquad (Equ. 6-4)$$

Fig. 6-52. The direct translation of the XOR function into CMOS primitives leads to a 22-transistor cell with 5 delay stages (Xor2.SCH)

Keeping in mind that negative logic is preferred in CMOS, we can re-arrange the expression of the **xor** cell in a less complex circuit, by replacing the AND/OR stage by NAND gates, following the equation 6-5. The total number of transistors is decreased to 16.

$$xor = \overline{\overline{\overline{AB}} + \overline{\overline{BA}}} = \overline{\overline{\overline{AB}}.\overline{\overline{BA}}}$$ (Equ. 6-5)
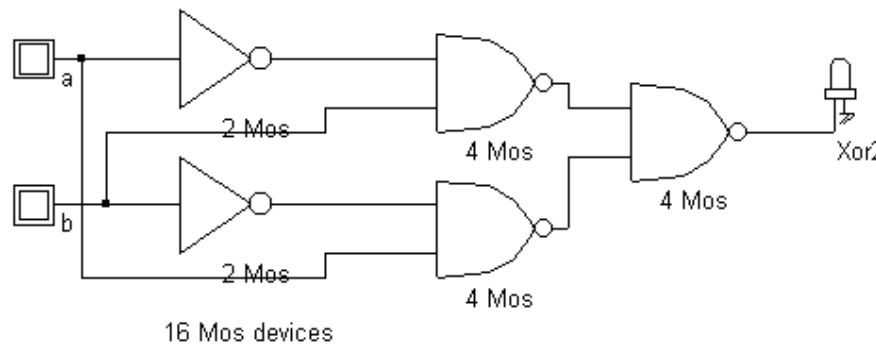

Fig. 6-53. An XOR function based on NAND gates leading to 16 transistors and 3 delay stages (XOR2_16.SCH)

Compiling a Schematic Diagram

An alternative to the manual design consists in describing the logic circuit of the XOR gate in DSCH, and then in compiling the schematic diagram into layout using MICROWIND. The design flow is detailed in figure 6-54. The XOR circuit is created according to the schematic diagram of figure 6-53. The circuit is saved under the name **Xor2_16.SCH**, for example. Next, we create the VERILOG description corresponding to the circuit, using the command **File →Make Verilog File**. The text file **Xor2_16.TXT**, which includes the VERILOG description, serves as the input of MICROWIND, to drive the automatic compilation of the circuit into layout. In MICROWIND, the command is **Compile →Make Verilog File.**
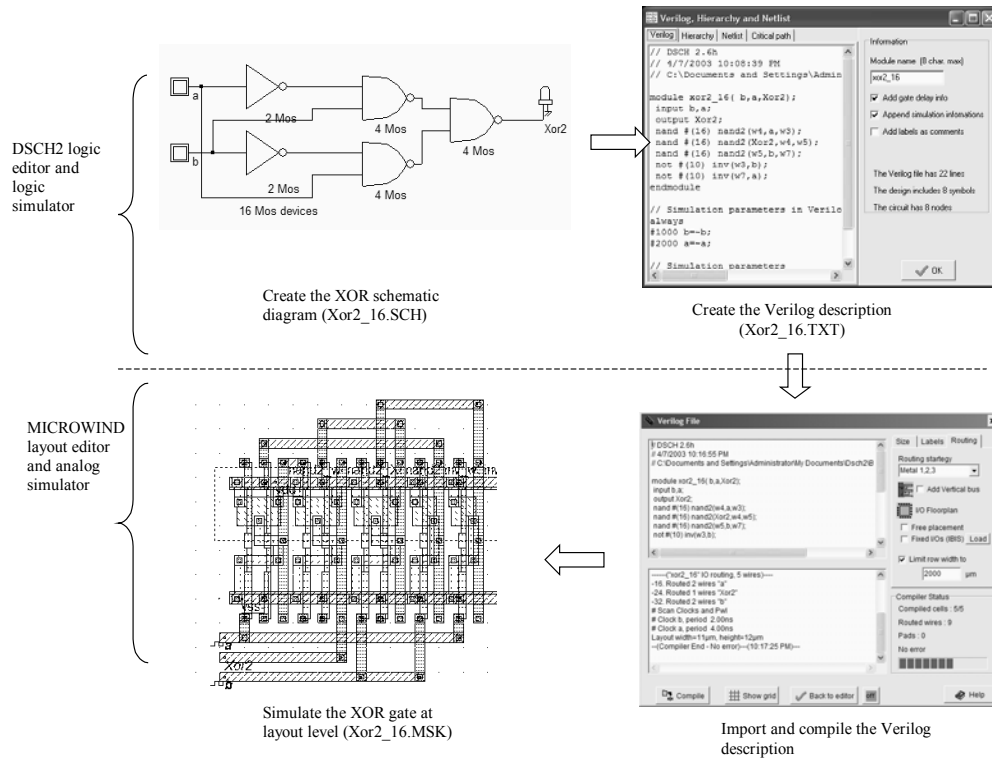
Figure 6-54: Compiling the XOR circuit from its logic description, through the VERILOG format (Xor2_16.MSK)

The result is a layout dynamically created by MICROWIND which corresponds to the initial circuit defined at logic level. The simulation works fine, as shown in figure 6-55. Notice that no simulation property is required as it is inherited from the logic circuit simulation. The VERILOG text file not only includes the structural description of the circuit but also the list of inputs and the associated simulation parameters. Still, the layout is quite large, and the switching performances suffer from the three delay stages.
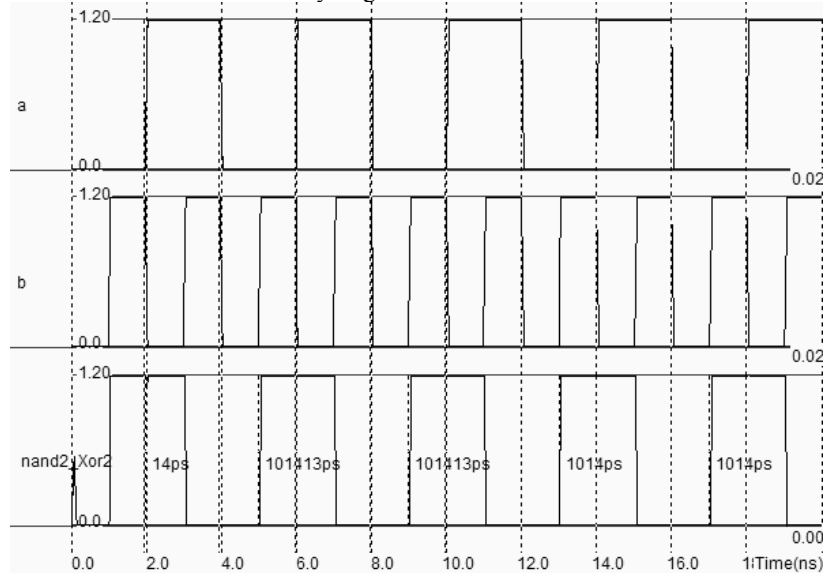

Figure 6-55: Simulation of the 3-stage XOR circuit (Xor2_16.MSK)

A better Design

An interesting design is shown in figure 6-55. The XOR function is built using AND/OR inverted logic (AOI logic <gloss>). The function created by the n-channel MOS network is equivalent to (A|~B)&(~A|B). The p-channel MOS network gives the function where all AND functions are transformed into OR, and vice-versa. In other words, the pMOS network realizes the function (A&~B)|(~A&B).
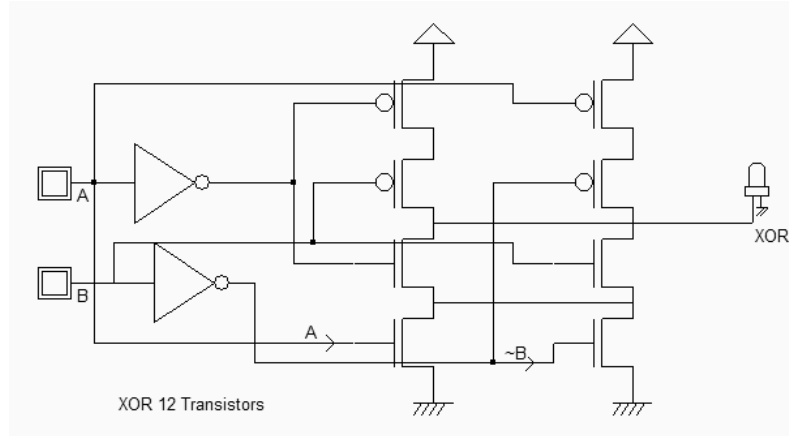

Figure 6-55: A 12-transistor XOR gate (XorAoi.SCH)

An efficient design

The most efficient solution consists of 2 inverters and 2 pass transistors, that is only 6 MOS. The cell is not a pure CMOS cell as two MOS devices are used as transmission-gates. The truth table of the XOR can be read as follow: IF B=0, OUT=A, IF B=1, OUT = ~A. The principle of the circuit presented below is to enable the *A* signal to flow to node *N1* if *B*=1 and to enable the *~A* signal to flow to node *N1* if *B*=0. The node *OUT* inverts *N1*, so that we cover the truth-table of the XOR operator. Notice that the nMOS and pMOS devices situated in the middle of the gate serve as pass transistors (Figure 6-56).
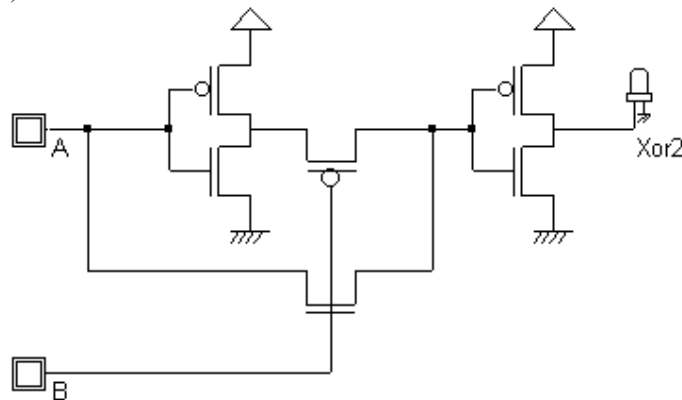

Fig. 6-56. The schematic diagram of the XOR gate with only 6 MOS devices (Xor2.SCH)
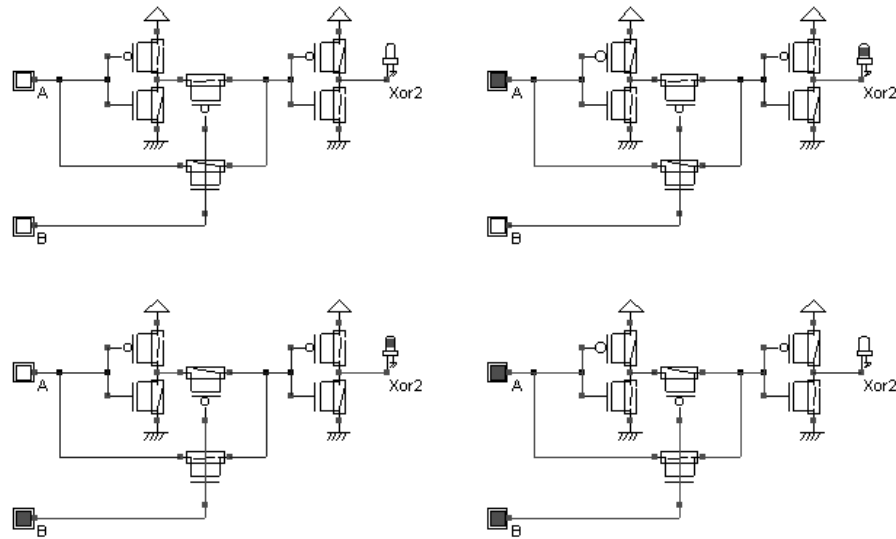
Fig. 6-57. Simulation of the XOR gate (Xor2Cmos.SCH)

The Verilog description of the XOR gate in Microwind is `xor=a^b`. The layout compiler produces an XOR cell layout as reported in Figure 6-58.
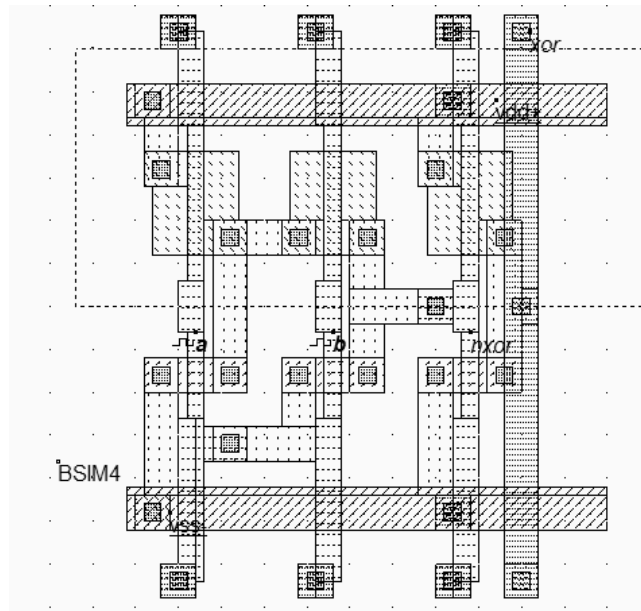

Fig. 6-58. Layout of the XOR gate. The BSIM4 label configures the simulator with BSIM4 model (XOR2.MSK).

When you add a visible property to the intermediate node **xnor** which serves as an input of the output stage inverter, see how the signal is altered by *Vtn* (when the nMOS is ON) and *Vtp* (when the pMOS is ON).
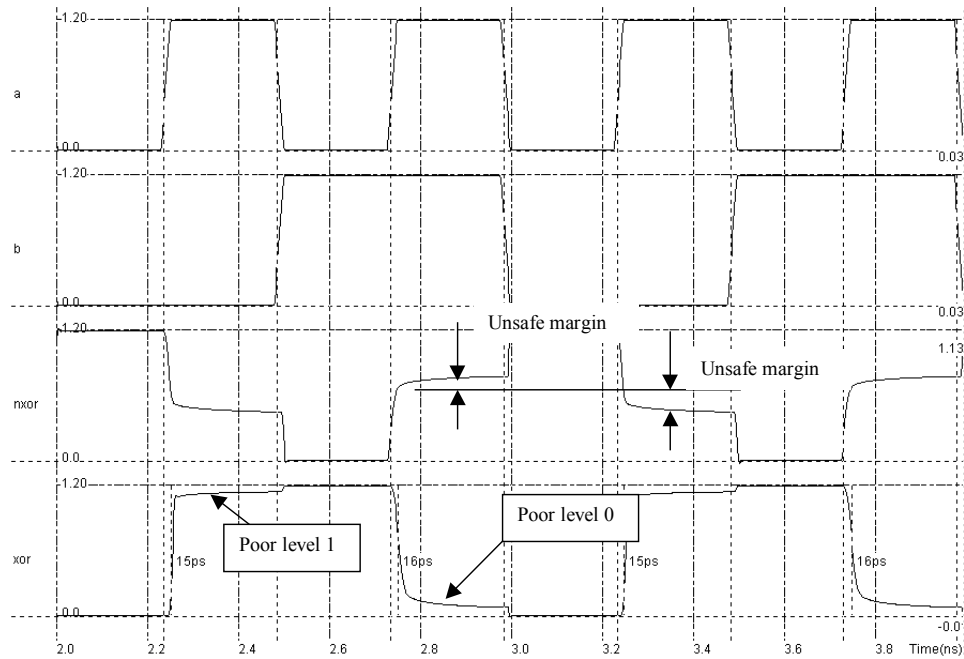
Fig. 6-59 Simulation of the XOR gate using BSIM4 model (XOR2.MSK).

The inverter regenerates the signal, but fails to produce clean 0 and 1 levels. This is because the MOS devices used as pass transistor reduces the voltage amplitude, resulting in dangerous voltage levels, close to the switching point of the logic. The alternative is to abandon the single pass-gate and use a complete pair of n-channel MOS and p-channel MOS devices, as shown in figure 6-60.
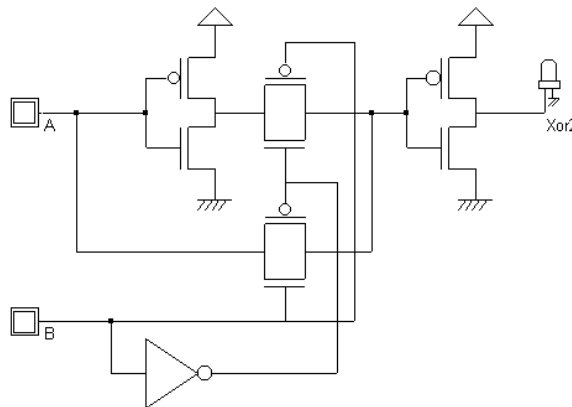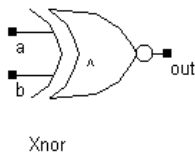

Fig. 6-60: Schematic diagram of the XOR gate with complete transmission gates (XOR2Full.MSK).

About the XNOR gate

Xnor

The XNOR gate symbol is shown above. The XNOR circuit is usually an exact copy of the XOR gate, except that the role of the B and ~B signals are opposite in the transmission-gate structures. Removing the last inverter is a poor alternative as the output signal is no more amplified. Adding a supplementary inverter would increase the propagation delay of one stage.

9. Complex Gates

Principles

The complex gate design technique applies for any combination of operators AND and OR. The AND operation is represented in logical equations by the symbol "&". The OR operation is represented by "|" (Table 6-3). The complex gate technique described below produces compact cells with higher performances, in terms of spacing and speed, than conventional logic circuits.

Bit operation	Verilog symbol used in complex gates
Not	~
And	&
Or		

Table 6-3: Logic operator used to compile complex gates

To illustrate the concept of complex gates, let us take the example of the following Boolean expression (Equation 6-6). Its truth-table is reported in table 6-4.

F= A&(B|C)   (Equ. 6-6)

A	B	C	A&(B|C)	F
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Table 6-4: Truth-table of the function F= A|(B&C))

A schematic diagram corresponding in a straightforward manner to its equation and truth-table is reported below. The circuit is built using a 2-input OR and a 2-input AND cell, that is 12 transistors and four delay stages.
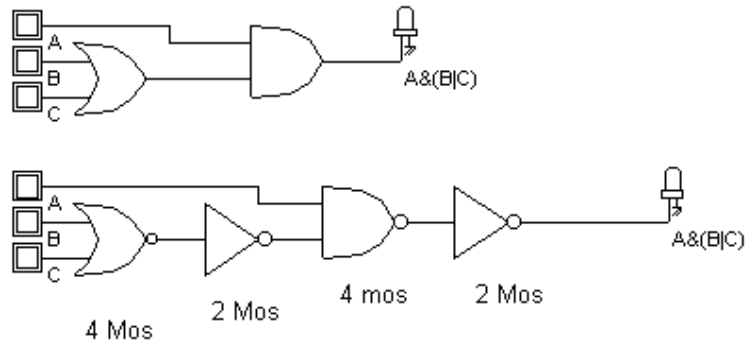
Fig. 6-61: The conventional schematic diagram of the function F (ComplexGate.SCH)

A much more compact design exists in this case (Figure 6-62), consisting in the following steps:

1. For the nMOS network, translate the AND operator '&' into nMOS in series , and the OR operator '|' into nMOS in parallel.

$$Fn= A \ series(B \ parallel \ C) \qquad (Equ. \ 6\text{-}7)$$

2. For the pMOS network, translate the AND operator '&' into pMOS in parallel, and the OR operator '|' into pMOS in series.

$$Fp= A \ parallel \ (B \ series \ C) \qquad (Equ. \ 6\text{-}8)$$

3. If the function is non-inverting , as for F, the inverter is mandatory.

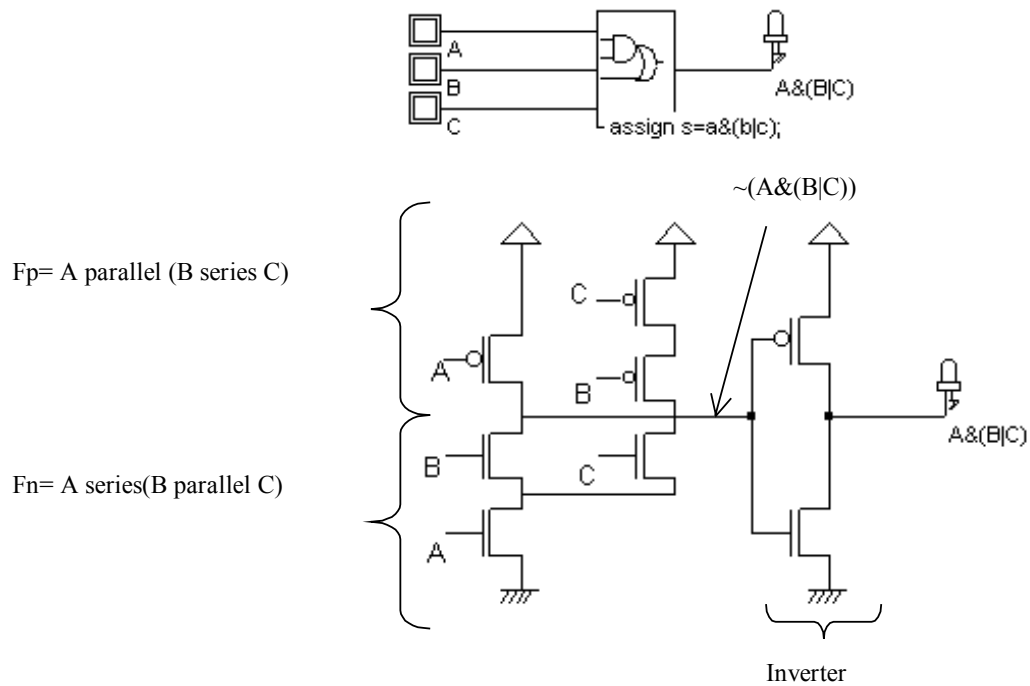*Fig.6- 62: The complex gate implementation of the function F=A&(B|C) (ComplexGate.SCH)*

Complex Gates in Dsch

Specific symbols exist to handle complex gate description in DSCH. The location of these symbols (3-input complex gate and 5-input complex gate) is shown in figure 6-63. At a double click inside the symbol, the menu shown in figure 6-64 appears. You must describe the logical function that links the output s to the inputs a,b,c. The syntax corresponds to the examples proposed in the previous paragraph (~for NOT, & for AND and | for OR). By using this behavioral description approach instead of building the function with basic cells, the switching performances of the gate are improved. Furthermore, the complex gate can be directly compiled into a compact layout using Microwind.
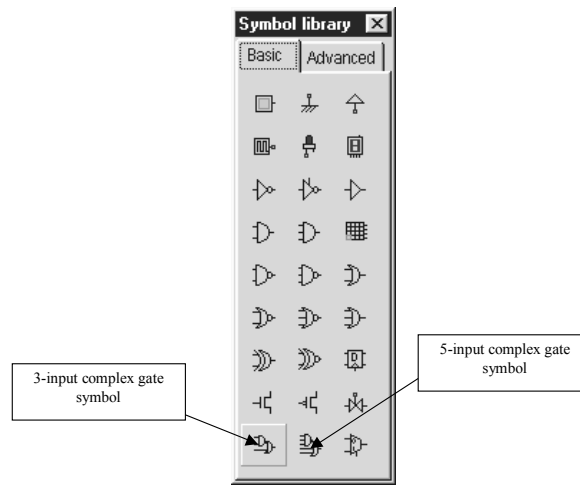

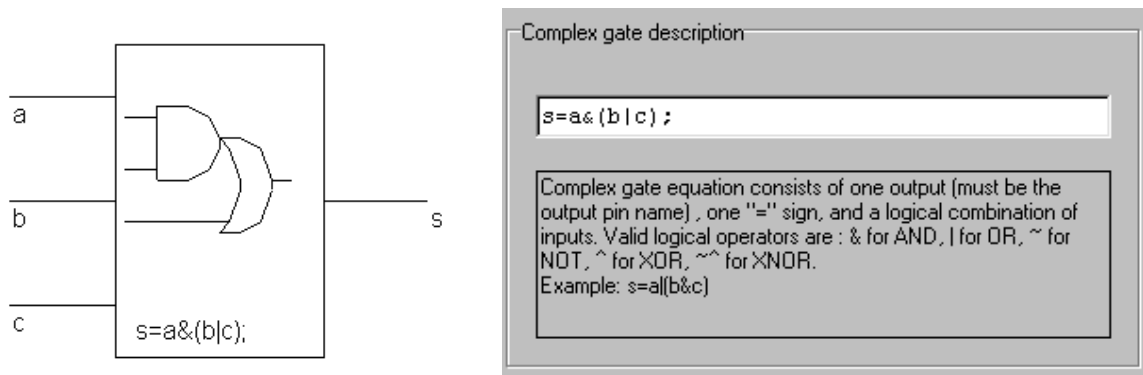*Fig. 6-63. Specific symbols proposed for complex gate description at logic level*

Fig. 6-64. The complex gate symbol and its logic description

Complex Gates in Microwind

Microwind2 is able to generate the CMOS layout corresponding to any description based on the operators **AND** and **OR**, using the command **Compile → Compile one line**. Using the keyboard, enter the cell equation, or modify the items proposed in the list of examples. In the equation 6-9, the first parameter *AndOr* is the output name. The sign '=' is obligatory, and follows the output name. The '~' sign corresponds to the operation NOT and can be *used only* right

after the '=' sign. The parenthesis '(' ')' are used to build the function, where '&' is the AND operator and '|' is the OR operator.

    Andor=A&(B|C)   (Equ. 6-9)
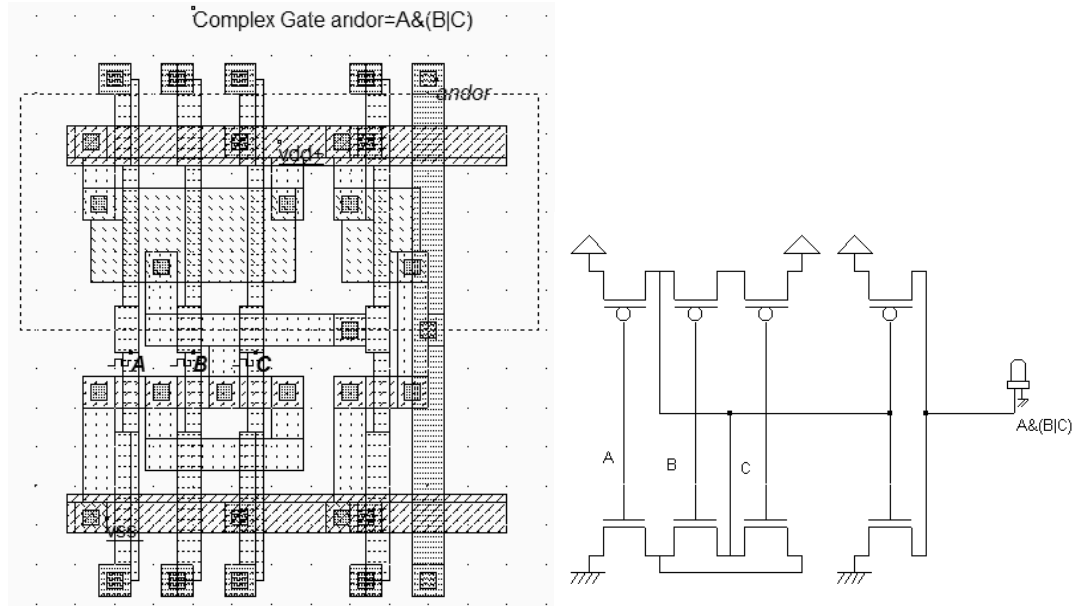

Fig. 6-65. Compiled complex gate andor=A&(B|C) (ComplexGate.MSK)

The MOS arrangement proposed by Microwind consists of a function ~A&(B|C) and one inverter. Notice that the cell could be rearranged to avoid the diffusion gap by a horizontal flip of the left structure and the sharing of VDD and VSS contacts.
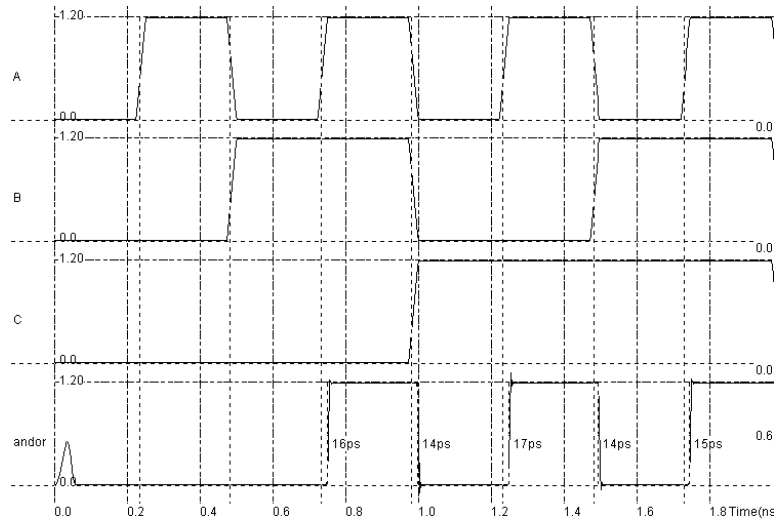

Fig. 6-66. Simulation of the complex gate andor=A&(B|C)

10. Multiplexor

Generally speaking, a multiplexor is used to transmit a large amount of information through a smaller number of connections. A digital multiplexor is a circuit that selects binary information from one of many input logic signals and directs it to a single input line. A behavioral description of the multiplexor is the case statement:

```
Case (sel)
  0 : f=in0;
  1 : f=in1
endcase
```

Sel	In0	In1	f
0	x	0	0
0	x	1	1
1	0	x	0
1	1	x	1

Table 6-8: the multiplexor truth-table

The usual symbol for the multiplexor is given in figure 6-67. It consists of the two multiplexed inputs *in0* and *in1* on the left side, the command *sel* at the bottom of the symbol, and the output *f* on the right.
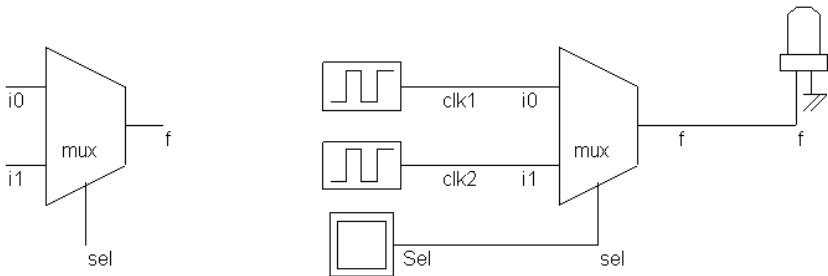

Fig. 6-67. The multiplexor circuit (MUX.SCH)

The simulation of the multiplexor proposed in figure 6-68 uses two different clocks *clk1* and *clk2*. When *sel*=0, *f* copies *clk1*. When *sel*=1, *f* copies *clk2*.
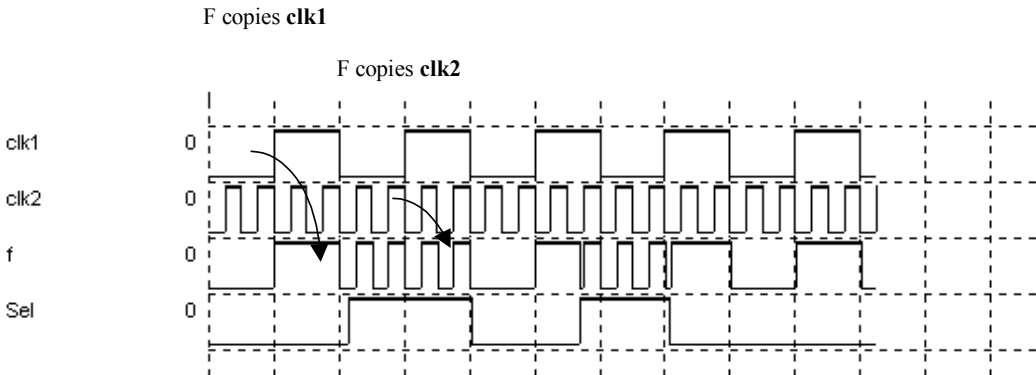

Fig. 6-68. Logic simulation of the multiplexor circuit (MUX.SCH)

Design of the Multiplexor

The most simple schematic diagram of the 2-input multiplexor is based on two MOS devices. The main problem of using single pass devices is the degradation of voltage levels. In 0.12μm, the default MOS devices ("low leakage") have a high threshold voltage, meaning that f does not reach clean high and low voltages in many cases. Furthermore, the output is not buffered, so f cannot drive significant loads. Moreover, inputs $i0$ and $i1$ are connected to the diffusions of n-channel and p-channel transistors. Depending on the value of sel, the load may vary, so the cell delay may vary accordingly. This circuit, appearing as structure (1) in figure 6-69 is rarely found in CMOS design as the signal f is weak and very sensitive to noise.
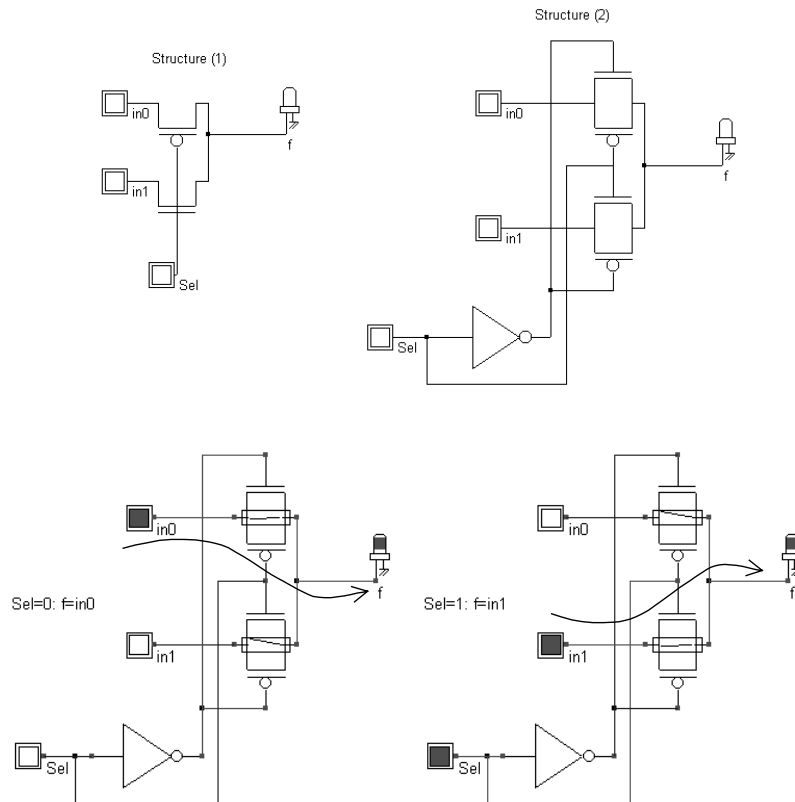

Fig. 6-69. Two-transistor and 6-transistor implementation of the multiplexor (MUX.SCH)

A better circuit, proposed in figure 6-69 as structure (2) is commonly used for low power operations. The threshold voltage degradation is eliminated by the use of transmission gates. However, the signal f is not amplified. Two implementations of the multiplexor are proposed in figure 6-70. The layout on the left has been generated automatically by placing one inverter and four single MOS devices one after the other. A manual arrangement of the MOS devices(Figure 6-70 right) leads to a much more compact circuit. When the output is loaded by 10fF, the circuit behaves quite well in terms of delay. As predicted, no parasitic threshold effect may be seen.
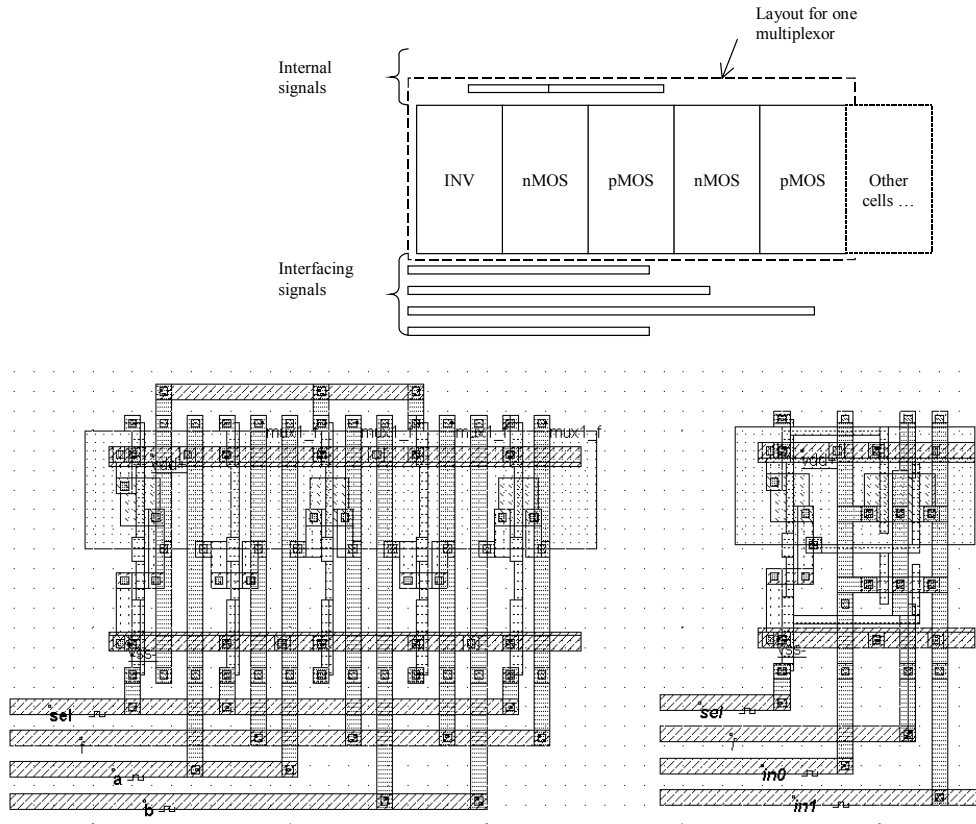
Fig. 6-70. compiled and manual implementation of the 6-transistor multiplexor (MUX.MSK)
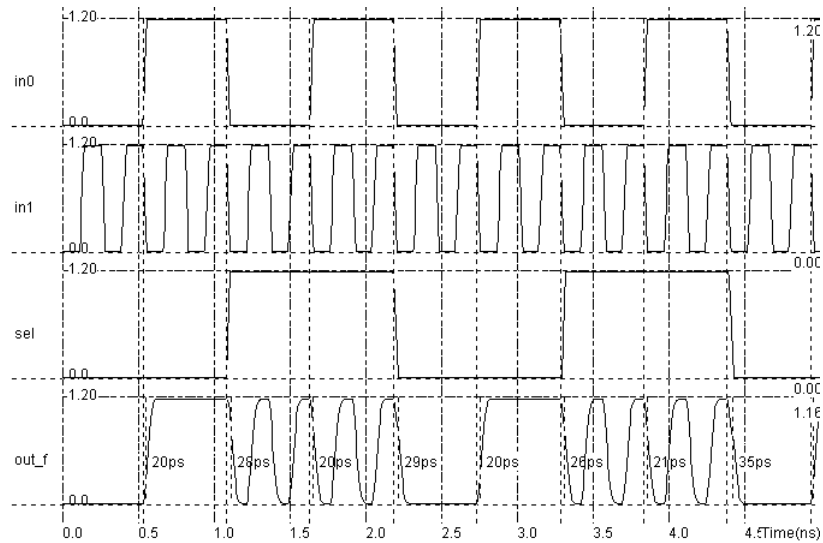

Fig. 6-71. Simulation of the 6-transistor multiplexor with a 10fF load (MUX.MSK)

Still, inputs i0 and i1 are connected to the diffusions of n-channel and p-channel pass transistors, which may lead to varying switching delays. Adding an output buffer and providing isolation from input signals to drain/sources lead to the multiplexor structure (3), presented in figure 6-72. This design is safe, easy to modelize at logic level, but requires many transistors and consumes a lot of power. When a multiplexor with large strength is required, the output inverter

can be modified by enlarging the output buffer width. The structure (4) also implements the multiplexor function, using complex gates for the combined OR/AND function, instead of transmission gates.
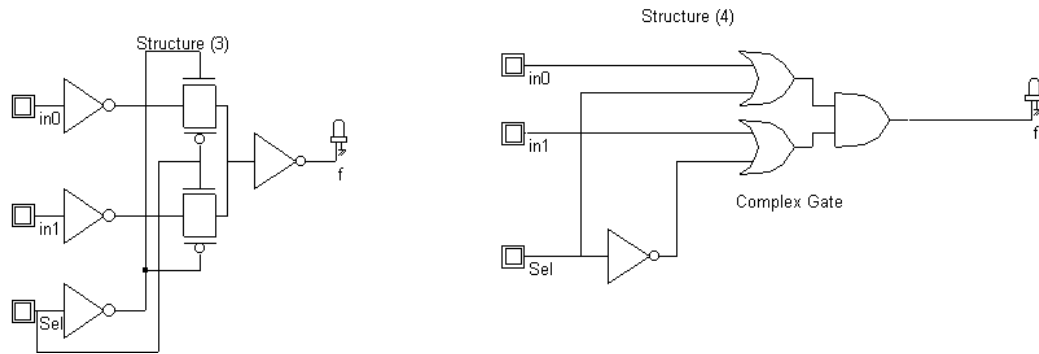

Fig. 6-72. Multiplexor implemented with transmission-gates or with complex gates (MUX.SCH)

n to 1 Multiplexer

The n-to-1 multiplexor performs the selection of a particular input line among n input lines. The selection is controlled by a set of lines which represent a number *sel*. Normally, there are 2^m input lines and m selection lines whose bit combinations determine which input is selected. Figure 6-73 shows one possible implementation of the 8-to-1 multiplexor, based on an elementary multiplexor as described in paragraph 6-69. A behavioral description of the n-to-1 multiplexor is given below:

```
Case (sel)
 0 : f=in0;
 1 : f=in1;
 2 : f=in2;
 3 : f=in3;
 4 : f=in4;
 5 : f=in5;
 6 : f=in6;
 7 : f=in7;
endcase
```
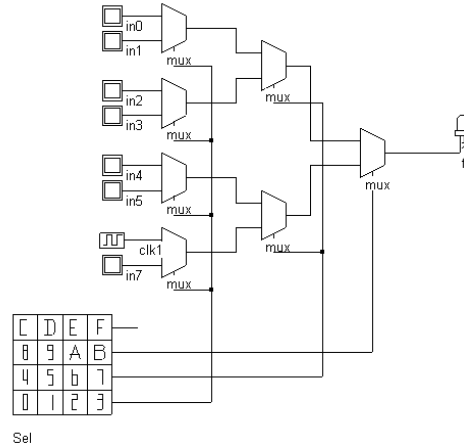
Fig. 6-73 8-to-1 multiplexing based on elementary multiplexor cells(Mux8to1.sch)

Timing Analysis

One important characteristic of the multiplexor cell is the propagation delay between the change of the input and the corresponding change of the output. The propagation delay is usually named t_{PD}. Another important delay, named t_{SD} is measured between the change of the selection and the effective setup of the corresponding channel. These delays are illustrated in figure 6-74.
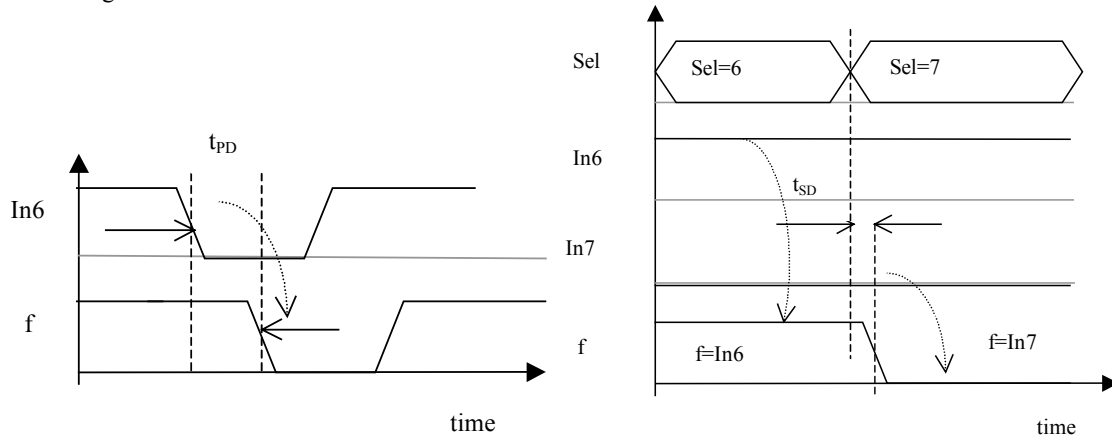

Fig. 6-74. Definition of the propagation delay t_{PD} and setup delay t_{SD}

The main drawback of the multiplexor design proposed in figure 6-73 is the use of local inverters at each elementary multiplexor gate, that lead to important power consumption and set-up delay. Figure 6-75 shows the direct transmission gate implementation of the 8-to-1 multiplexor. The result is simpler than for the multiplexor implementation, it works faster and requires fewer devices.
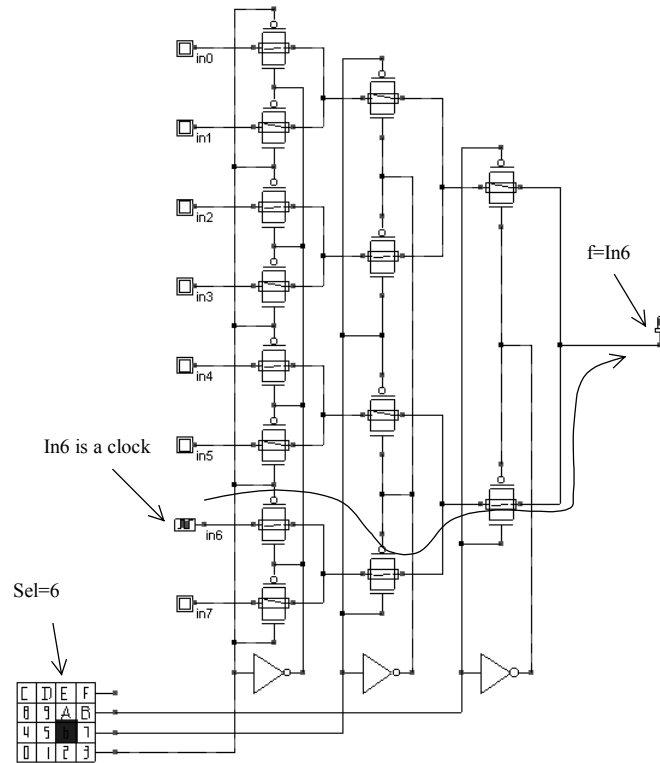
Fig. 6-75. 8-to-1 Multiplexor based on transmission gates (Mux8to1Tgate.SCH)

We have implemented the 8-to-1-transmission gate multiplexor by compiling its Verilog description, as shown in figure 6-77. The layout properties have been changed to consider only the selection of *In0* and *In6* alternatively. In0 is assigned a fast clock while *In6* is assigned a slow clock. The simulation scenario consists first in multiplexing first the input *In0* to the output *f* (Named *pmos_f* in the layout), with Sel1=0, Sel2=0, and then in multiplexing the input *In1* with *Sel1*=1, *Sel2*=1, at time 2.0ns in figure 6-78. The output copies successively *In0* and *In1*, as expected. The eye diagram shows an homogeneous switching delay performance.
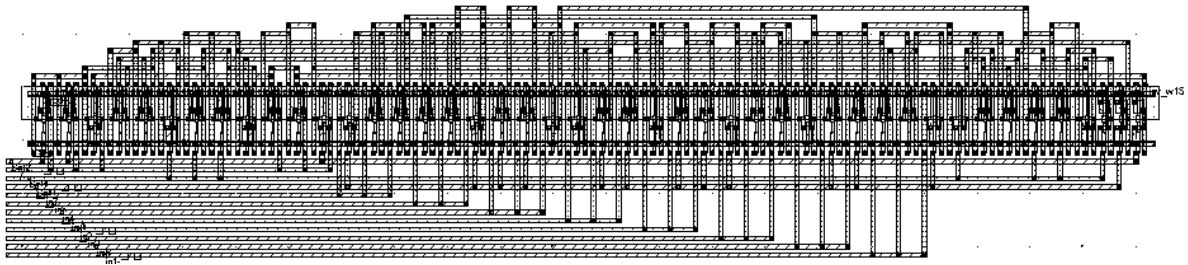

Fig. 6-77. Automatic generation of the 8-to-1 Multiplexor based on transmission gates (Mux8to1Tgate.MSK)
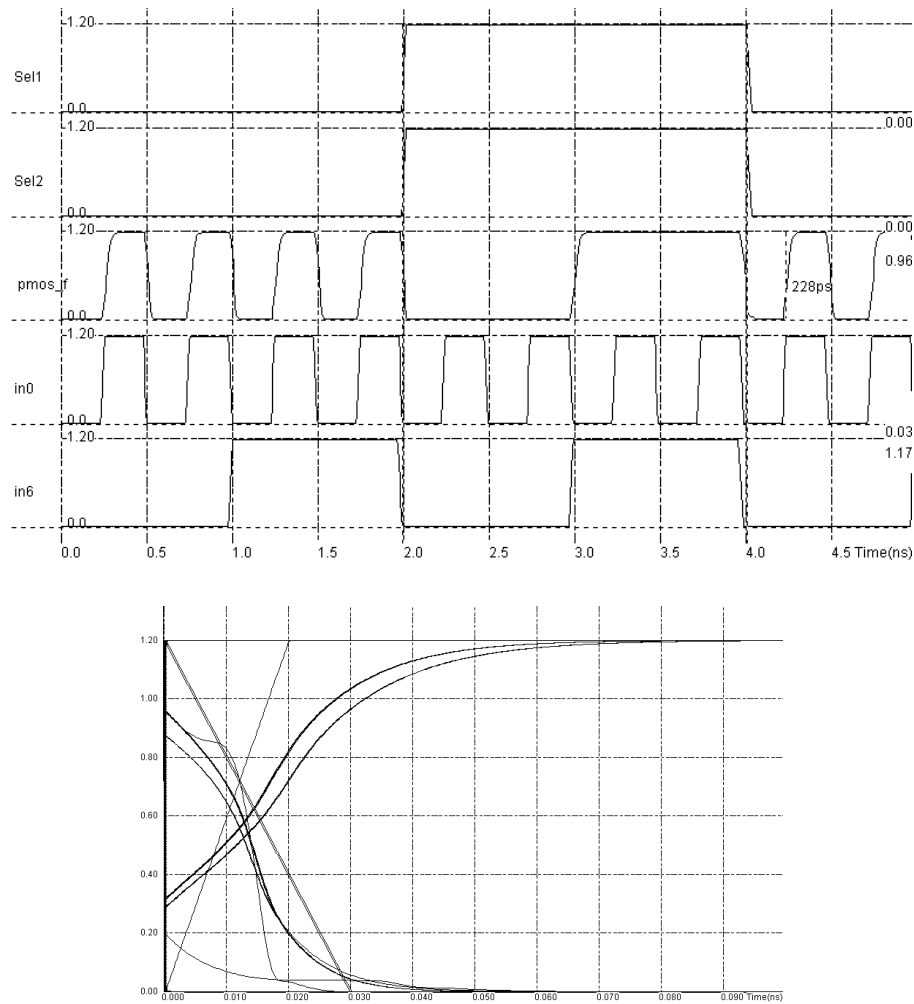
Fig. 6-78 Analog simulation and eye diagram of the 8-to-1 multiplexor (Mux8to1Tgate.MSK)
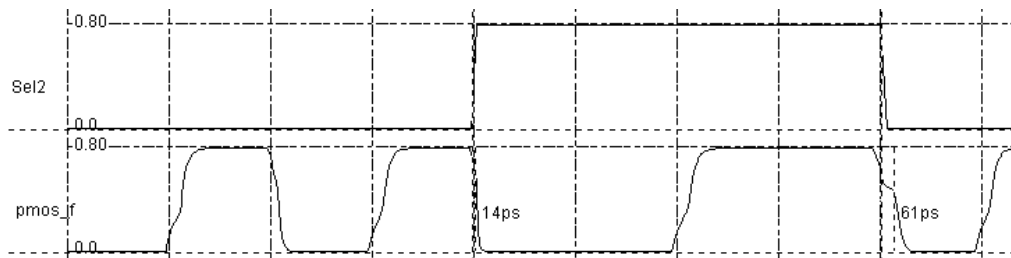

Fig. 6-79 Low voltage operation of the 8-to-1 multiplexor (Mux8to1Tgate.MSK)

If we try to perform the same operation with a voltage supply significantly lower than VDD (Here 0.8V, that is 66% of VDD), the circuit based on transmission gates still operates in a satisfactory way, although significant delays are observed (Figure 6-79).

All n-MOS multiplexor

Some authors have proposed multiplexor designs only based on n-channel MOS devices, as reported in figure 6-80. Although this architecture is easy to implement in a regular way, the waveform is degraded by the parasitic threshold effect of n-channel MOS devices when passing high signals. Therefore, the output should later be refreshed thanks to a buffer. Although the gain in silicon area is evident, no improvement in switching speed should be expected, as the cascaded threshold loss at each pass transistor leads to very slow rise times.
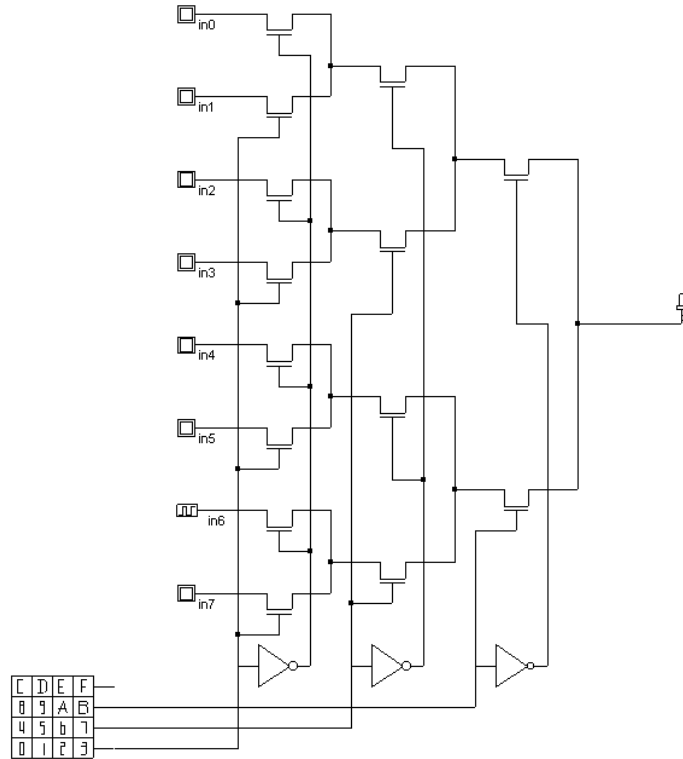

Fig. 6-80. 8-to-1 Multiplexor based on n-channel MOS (Mux8to1Nmos.SCH)

We have implemented the 8-to-1 n-channel MOS multiplexor by compiling its Verilog description, as shown in figure 6-81. The layout properties have been changed in a similar way as for transmission gate design. The layout is much more compact as expected. Unfortunately, the switching performances are very poor, which is especially visible in the eye-diagram shown in figure 6-82. When trying to pass the "one", the series of n-channel MOS device degrades the levels considerably. The signal reaches VDD/2 only after 100ps, that is 3 times slower than the transmission gate design.
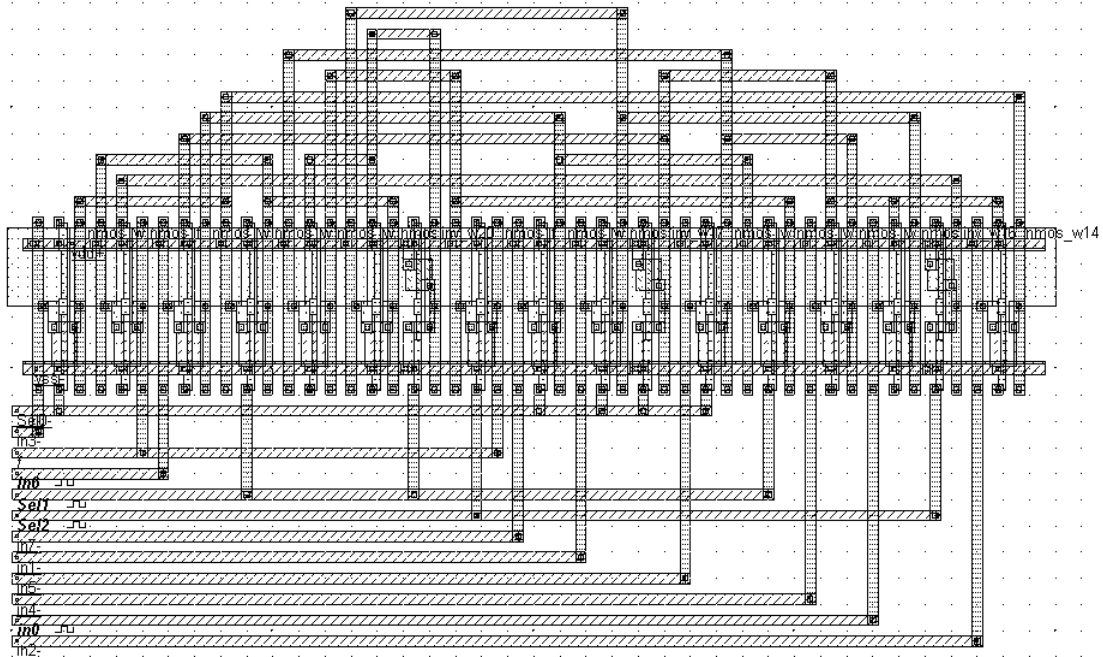
Fig. 6-81. Automatic generation of the 8-to-1 Multiplexor based on n-channel MOS (Mux8to1Nmos.MSK)
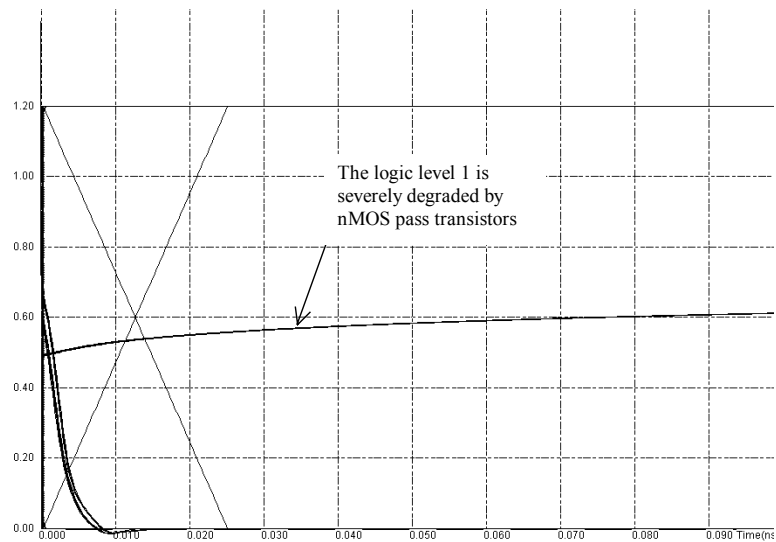

Fig. 6-82. Eye-diagram simulation of the 8-to-1 Multiplexor based on n-channel MOS (Mux8to1Nmos.MSK)

When we try to operate the circuit at low supply voltage (0.8V), the circuit does not work anymore as the final voltage achieved by the output f is as low as 0.3V (Figure 6-83), still below the switching point of most logic gates.
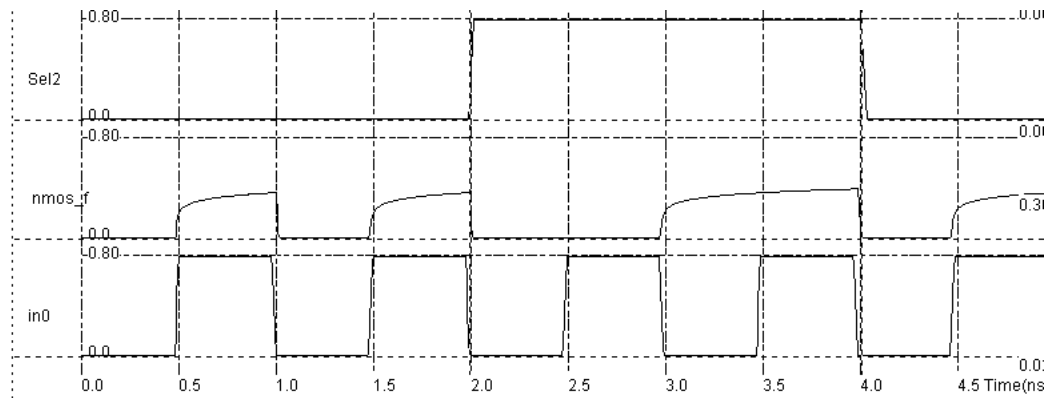
Fig. 6-83. The n-channel multiplexor does not operate at low voltage (Here 0.8V) (Mux8to1Nmos.MSK)

Demultiplexor

Probably one of the most well-known demultiplexor is the 74LS138 circuit, which is presented in figure 6-84. If the enable circuit is active (G1 must be at 1), all outputs O0..O7 are at 1 except the one issued from the binary decoding of the input A,B and C, according to the statements below.

```
Case (adress)
 0 : O1=0;
 1 : O2=0;
 2 : O3=0;
 3 : O4=0;
 4 : O5=0;
 5 : O5=0;
 6 : O6=0;
 7 : O7=0;
endcase
```

Note that the multiplexor can be implemented using NAND4 cells, to avoid the use of AND logic circuits which require NAND and Inverter circuits. Details on the demultiplexor layout are given in chapter 10 dealing with memory design.
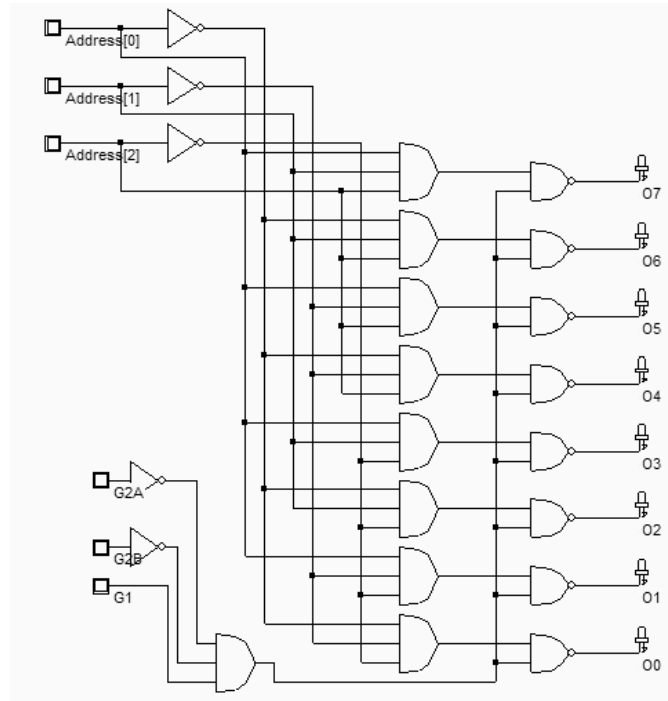
Fig. 6-84. The demultiplexor (74ls138.SCH)

11.   Shifters

Shifters are very important circuits found in virtually all processor cores. Shifters are able to manipulate data and shift the bits to the right or to the left. Taking the example of an 8-bit input data A, initially set to 0xB3 in hexadecimal (10110011 in binary), the result of shifting A two bits to the right is 0x3c (00101100 binary), as illustrated in figure 6-85 The corresponding symbol is >>. Now, the result of shifting A three bits to the left would be 0x98 (10011000 binary). The corresponding symbol is <<.

(a) Shift Right 2 bits (>>2)   *(b) Shift left 3 bits (<<3)*
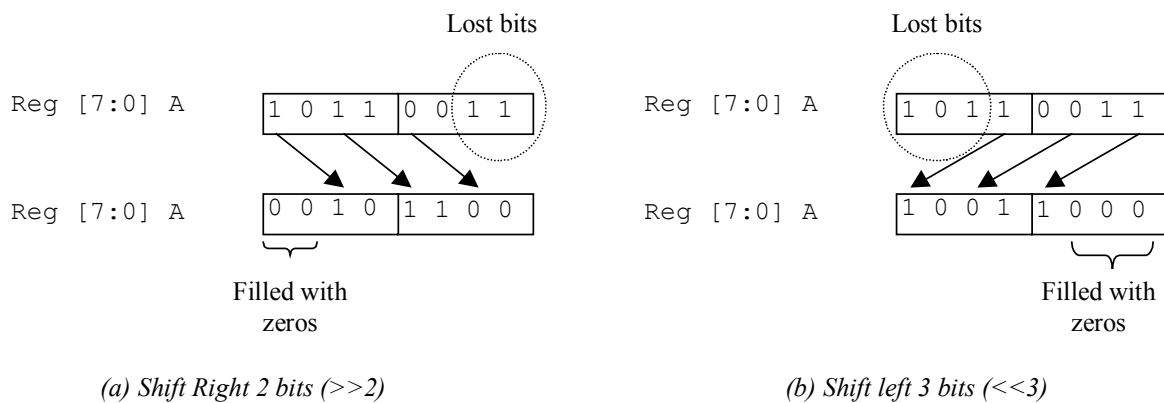
Figure 6-85: Principles for shifting data to the right and to the left

The rotate circuit is based on the shift mechanism, but has the property to re-inject the lost bits in the place left for zeros. An illustration of the rotate structure is given in figure 6-86.
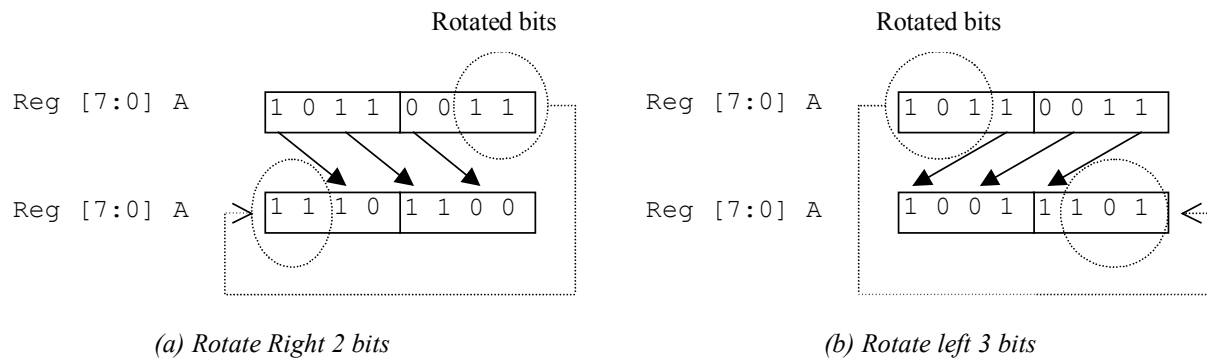
Rotated bits   Rotated bits

Reg [7:0] A   1 0 1 1 0 0 1 1   Reg [7:0] A   1 0 1 1 0 0 1 1

Reg [7:0] A   1 1 1 0 1 1 0 0   Reg [7:0] A   1 0 0 1 1 1 0 1

(a) Rotate Right 2 bits   *(b) Rotate left 3 bits*

Figure 6-86: Principles for rotating data to the right and to the left

There are mainly two ways to implement the shift circuits. One [Weste] is based on multiplexor cells, the other [Uyemura] is based on pass transistors. Pass transistors yield simpler and more regular layout structures. The main drawback is a slower switching and less predictable timing characteristic. The 4-bit shift right circuit is shown in figure 6-87, while the rotate left circuit is reported in figure 6-88. The shifter based on multiplexor has the same structure, except that all single pass transistors are replaced by a pair of nMOS and pMOS in parallel. Each gate control must be inverted, which makes the final circuit significantly more complex than the single MOS shifter.
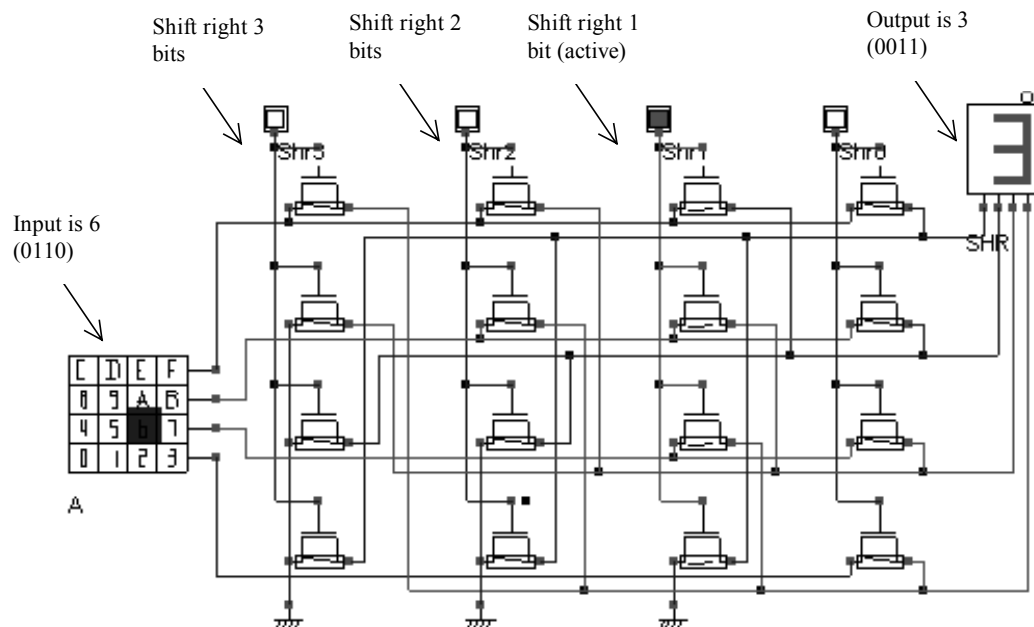

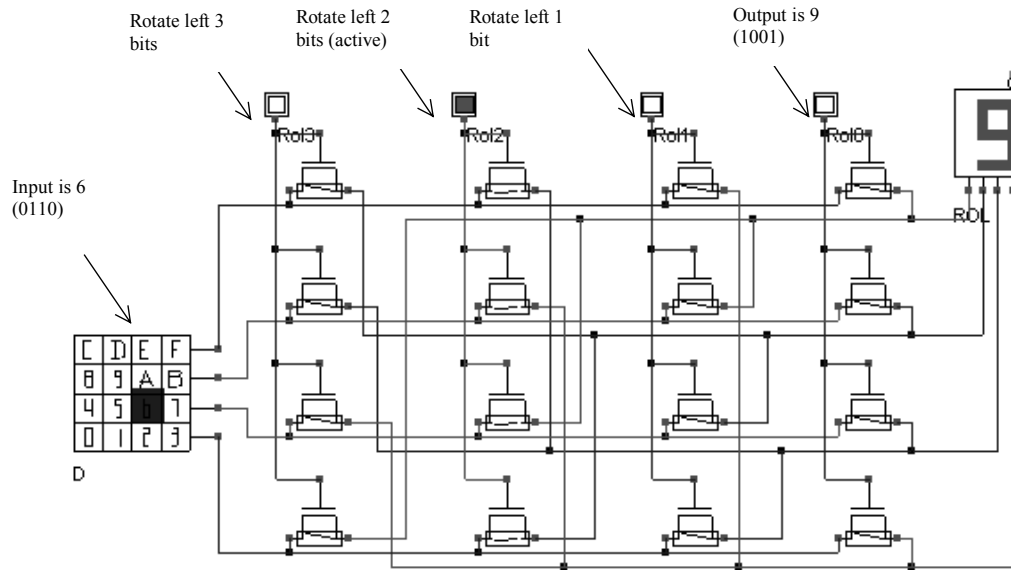*Figure 6-87: Simulation of the shift right circuit (ShiftRotate4b.SCH)*

Figure 6-88: Simulation of the rotate left circuit (ShiftRotate4b.SCH)

12. Description of basic gates in Verilog

We give in this paragraph a short introduction to Verilog hardware description language (HDL). This language is used by *Microwind* and *Dsch* tools to describe digital logic networks. The description is a text file, which includes a header, a list of primitive keywords for each elementary gate, and node names specifying how the gates are connected together. Let us study how the logic network of Figure 6-89 is translated into a Verilog text.

```
// DSCH 2.5d                                                      1
// 14/04/02 16:52:45                                              2
// C:\Dsch2\Book on CMOS\Base.sch                                 3
                                                                  4
module Base( Enable,A,B,s_BUF,s_NOT,s_NOTIF1,s_AND,s_NAND,        5
s_NOR,s_OR,s_XOR,s_XNOR,s_PMOS,s_NMOS);                           6
input Enable,A,B;                                                 7
output s_BUF,s_NOT,s_NOTIF1,s_AND,s_NAND,s_NOR,s_OR,s_XOR;        8
output s_XNOR,s_PMOS,s_NMOS;                                      9
and #(16) and2(s_AND,A,B);                                        10
notif1 #(13) notif1(s_NOTIF1,A,Enable);                          11
not #(10) not(s_NOT,A);                                           12
buf #(13) buf1(s_BUF,A);                                          13
nand #(10) nand2(s_NAND,A,B);                                     14
nor #(10) nor2(s_NOR,B,A);                                        15
or #(16) or2(s_OR,B,A);                                           16
xor #(16) xor2(s_XOR,B,A);                                        17
xnor #(16) xnor2(s_XNOR,B,A);                                     18
nmos #(10) nmos(s_NMOS,A,Enable); // 1.0u 0.12u                   19
pmos #(10) pmos(s_PMOS,A,Enable); // 2.0u 0.12u                   20
endmodule                                                         21
                                                                  22
// Simulation parameters                                          23
// Enable CLK 10 10                                               24
// A CLK 20 20                                                    25
// B CLK 30 30                                                    26
```

Fig. 6-89. The Verilog description of the circuit BASE.SCH (Base.TXT)

The Verilog description is a text file. Double slash characters are used for comments. The rest of the line is ignored. Consequently, lines 1 to 3 are comments. In line 5, the keyword **module** defines the start of the description, which will end after the keyword **endmodule** (line 21). The name of the module is **base**. Its parameters are the list of input and output signals. From line 7 to line 9, signals are declared as **input**, **output**, or internal **wire**. No internal wire exists in this file. Three input and 11 output signals are listed. Then, from line 10 to 20, the schematic diagram is described as a list of primitives.

Name	Logic symbol	Verilog primitive
INVERTER		Not <name>(out,in);
BUFFER		Buf <name>(out,in);
TRI-STATE INVERTER		Notif1 <name>(out,in,enable);
TRI-STATE BUFFER		Bufif1 <name>(out,in,enable);
AND		And <name>(out,in1,in2,..);
NAND		Nand <name>(out,in1,in2,..);
OR		Or <name>(out,in1,in2,..);
NOR		Nor <name>(out,in1,in2,..);
XOR		Xor <name>(out,in1,in2,..);
XNOR		Xnor <name>(out,in1,in2,..);
NMOS		Nmos <name>(out,source,gate);
PMOS		Pmos <name>(out,source,gate);

Table 6-8. The VERILOG primitives supported by DSCH and MICROWIND

The gate delay is described for the accurate simulation of time delays in large structures. The logic delay through each gate is specified in integer units. One unit corresponds to a single logic simulation cycle. The correspondence between the elementary cycle and the real time varies depending on the technology. In 0.12μm, which is the default directory used by DSCH, the elementary unit is 0.01ns, that is 10ps. Consequently, the AND gate described in line 10 has a switching delay 16x0.01ns=0.160ns, or 160ps. This delay is observed between each active transition of A,B and s_AND.
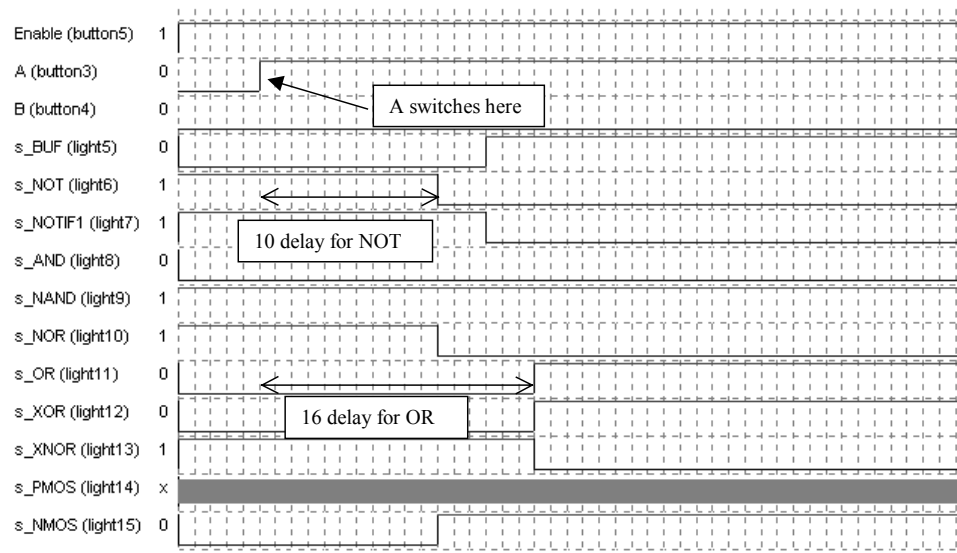
Fig. 6-90. Illustration of the elementary logic delay in the simulation chronograms (Base.SCH)

In the simulation chronograms, when we zoom strongly on the time scale in order to see each delay step (0.01ns), we observe the discrete aspect of the delay estimation, as shown in figure 6-90. A series of 10 elementary delay is used for the NOT cell, and 16 for the OR cell, according to the Verilog description made in figure 6-89.

The gate delay description is optional. If no delay information has been specified, the logic simulator assigns a default gate delay, which is a basic parameter of the technology. In 0.12μm, the default gate delay is 0.03ns, and each wire connected to the gate adds a supplementary 0.07ns delay. A more detailed description of the Verilog language may be found in [Uyemura].

13.   Conclusion

In this chapter, the design of basic cells has been reviewed. We have described in details the NAND, AND, OR and NOR gates, and the asymmetrical switching problems of multiple-input NOR circuits. The specific design techniques to design a compact XOR gate has then been reviewed. The techniques for translating AND/OR combinations into an optimized have also been studied. Finally, we have described the structure of multiplexors, shifters, and given some information about the VERILOG format used by DSCH and MICROWIND to exchange structural descriptions of logic circuits.

REFERENCES

[Weste] Neil Weste, K. Eshraghian "Principles of CMOS VLSI design", Addison Wesley, ISBN 0-201-53376-6, 1993

[Baker] R.J. Baker, H. W. Li, D.E. Boyce "CMOS circuit design, layout and simulation", IEEE Press, ISBN 0-7803-3416-7, 1998

[Uyemura] John.P. Uyemura "Introduction to VLSI Circuits and Systems", Wiley, 2002, ISBN 0-471-12704-3

Exercises

Exercise 4.1. Give the switching point voltage of the gate shown in figure 6-91. Find its logic function.

<Sonia: Sizing ? NMOS bizarre?>
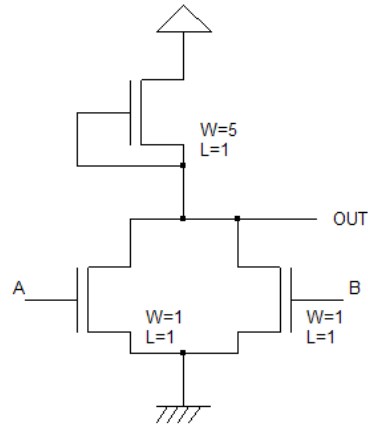
Figure 6-91: A logic gate case study

Answer: <Sonia>

Exercise 6.2: Design a CMOS 3-input XOR using CMOS And-OR-Invert logic.

Answer: See appendix F

Exercise 6.3: Design the following function using CMOS And-OR-Invert logic and try to find continuous diffusions.

F= ~(A&B|c&(A|B))

Answer: See appendix F

Exercise 6.4: Using Microwind , compare the switching point voltage of a 3-input NOR gate (minimum size MOS) to the switching point voltage of a 3 inputs NAND gate (minimum size MOS). Which one is closer to the ideal and why?

Answer: The Nand switching point voltage is closer to VDD/2 because the electron mobility is higher than that of the hole.

Exercise 6.5: What is the functionality of the circuit shown in figure 6-93 ? Justify with a chronogram using Dsch2.
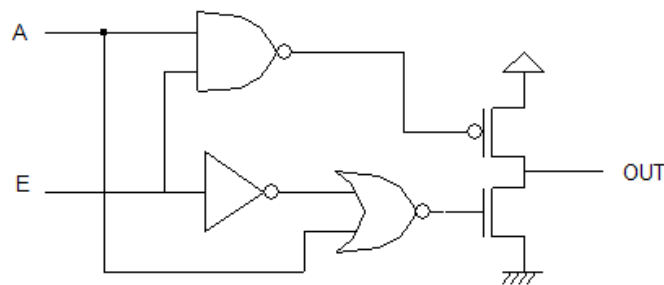

Figure 6-93: logic circuit case study

Exercise 6.6: What is the functionality of the circuit shown in figure 6-94 ? Justify with a chronogram using DSCH2. Implement this circuit into layout. What are the conditions to match the logic behavior?
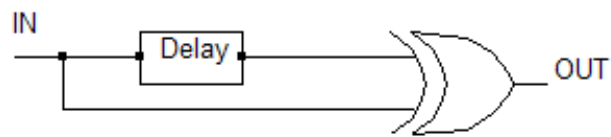

Figure 6-94: A logic gate case study