

# 6

## Latches and Memories

This chapter details the structure and behavior of latches and memory circuits. The RS Latch, the D Latch and the edge-sensitive register are presented. Then, the concepts of ROM, static RAM and dynamic RAM memories are introduced, together with simulations.

### RS Latch

The RS Latch, also called Set-Reset Flip Flop (SR FF), transforms a pulse into a continuous state. The RS latch can be made up of two interconnected NAND gates. In that case, the Reset and Set inputs are active low. The memory state corresponds to  $\text{Reset}=\text{Set}=1$ . The combination  $\text{Reset}=\text{Set}=0$  should not be used, as  $Q=\text{not}Q=1$ . Furthermore, the simultaneous change from  $\text{Reset}=\text{Set}=0$  to  $\text{Reset}=\text{Set}=1$  provokes what is called the metastable state, that corresponds to a parasitic ring effect that may jeopardize the behavior of the whole circuit.

An alternative implementation of the RS latch is made from NOR gates. In that case, the Reset and Set inputs are active high. The cell transforms positive pulse into continuous states.

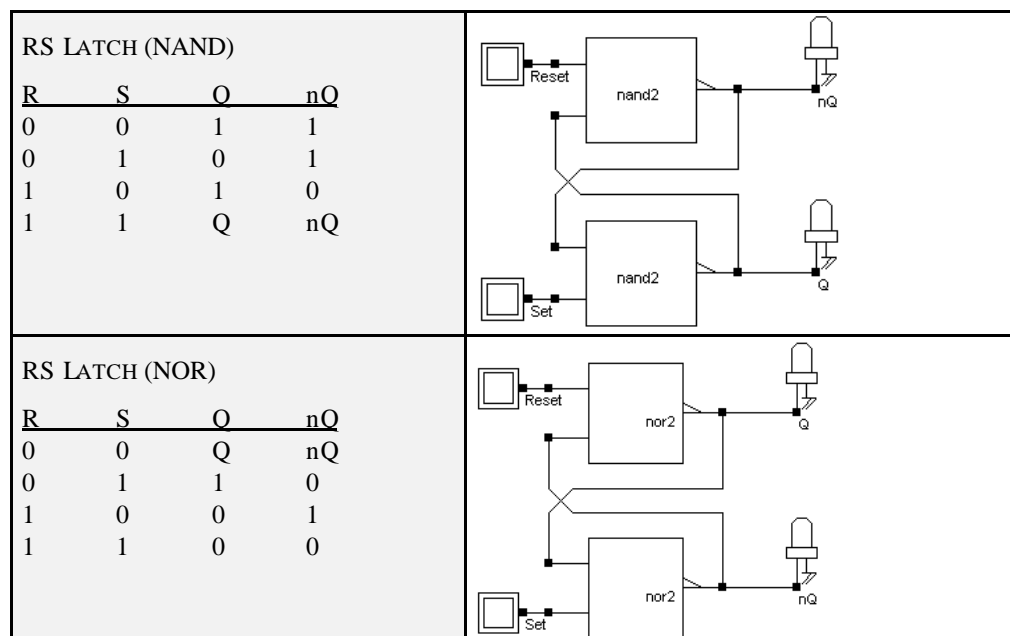
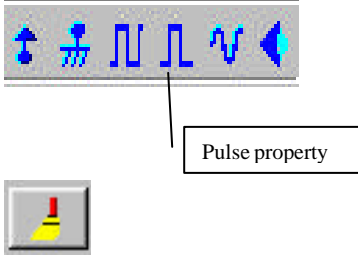


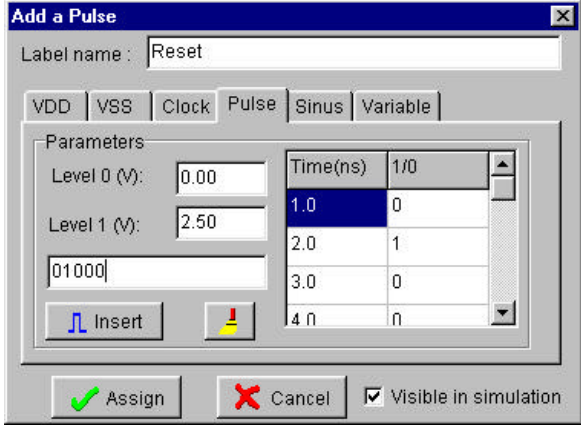
Fig. 6.1. The truth table and schematic diagram of a RS latch made (RSNor.SCH, RSNand.SCH)

FULL CUSTOM LAYOUT	You may create the layout of RS latch manually. The two NAND gates may share the VDD and VSS supply achieving continuous diffusions. The internal routing may also save routing area, leading to the layout shown in Figure 6.2.
LAYOUT LIBRARY	Load the layout design of the RS Latch through the <b>File -&gt; Open</b> and <b>RS.MSK</b> sequence.
VERILOG COMPILING	<ol style="list-style-type: none"> <li>1. Use DSCH2 to create the schematic diagram of the RS latch. Verify the circuit with buttons and lamps. Save the design under the name 'RS.sch' using the command <b>File -&gt; Save As</b>.</li> <li>2. Generate the Verilog text by using the command <b>File -&gt; Make Verilog File</b>.</li> <li>3. In Microwind2, click on the command <b>Compile -&gt; Compile Verilog File</b></li> <li>4. Select the text file 'RS.txt'. <pre> module RSNor( Reset,Set,Q,nQ);   input Reset,Set;   output Q,nQ;   nor nor1(Q,nQ,Reset);   nor nor2(nQ,Set,Q); endmodule </pre> </li> <li>5. Click on <b>Compile</b>. When the compiling is complete, the resulting layout appears as shown below. The NOR implementation of the RS gate is completed.</li> </ol>

#### Add a Pulse Property

With the Reset and Set signals behaving like clocks, the memory effect is not easy to illustrate. A much better approach consists in declaring pulse signals with an active pulse on RESET followed by an active pulse on SET. Consequently, you must change the "CLOCK" property into a "PULSE" property. For NOR implementation, the pulse is positive.

	<ol style="list-style-type: none"> <li>6. Select the "PULSE" icon. Click on the "RESET" node.</li> <li>7. Click the brush to clear the existing pulse properties of the pulse.</li> <li>8. Enter the desired sequence, for example 01000. An click "INSERT". A piece-wise-linear sequence is generated in the table, describing the 01000 waveform in an analog way.</li> </ol>
---	---



9. Repeat the same procedure to change the clock into a pulse for node “SET”. This time the sequence must be 000100 to delay the pulse.

10. Click on **Simulate ->Start Simulation**. The timing diagrams of figure 6.3 appear. Click on **Close** to return to the editor.

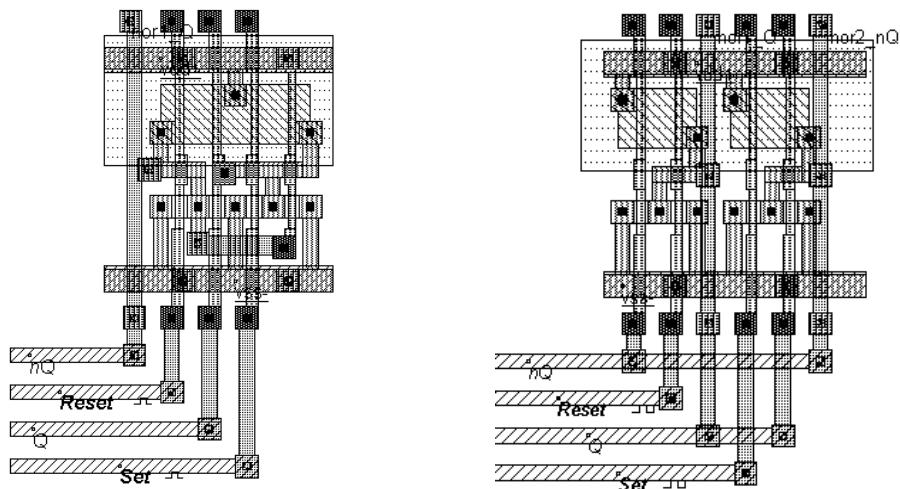


Fig. 6.2. Manual (Left) and compiled (Right) layout of the RS latch made (RSNor.MSK)

In the simulation of Figure 6.3, a positive pulse on “SET” turns Q to a stable high state. Notice that when SET goes to 0, Q remains at 1, which is called the ‘memory’ state. When a positive pulse occurs on “RESET”, Q goes low, nQ goes high. In this type of simulation, the combination Reset=Set=1 is not present.

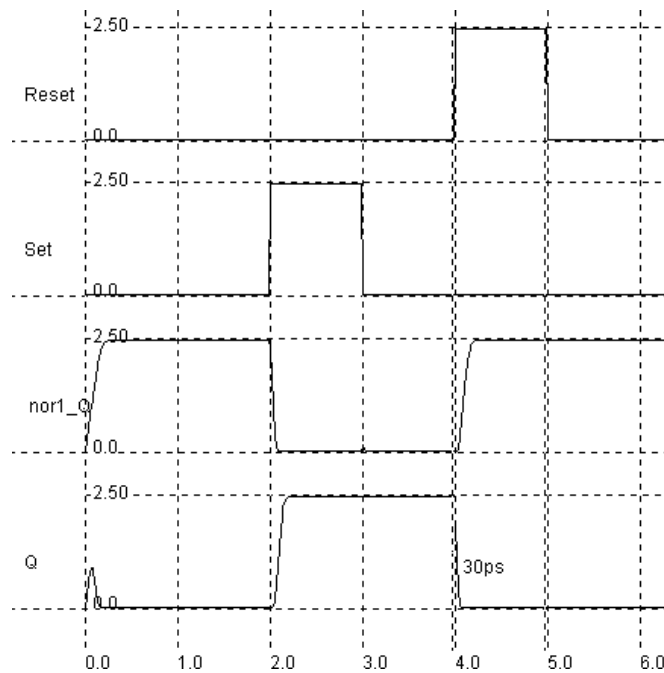


Fig. 6.3. Simulation of the RSNOR latch (RSNor.MSK)

#### Minimum Pulse Width

One technique to characterize the RSNor cell performances is to extract the minimum pulse width that provokes the “SET” or “RESET” effect. For this study, the latch outputs should be connected to a load, to conduct the simulations in a realistic environment. A simple charge consists in an inverter input, connected to Q. In figure 6.4, the pulse width reduction lead to a wrong behavior when the pulse width is below 100ps. This parameter is always given in the data sheet of the RS latch, and more generally in all latches information.

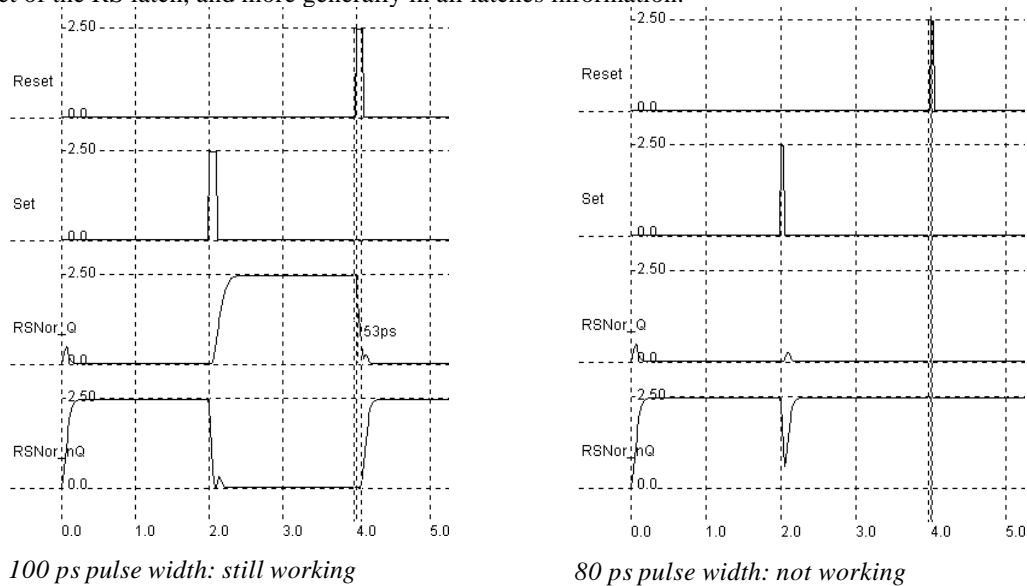


Fig. 6.4. Finding the minimum pulse width of the RSNOR latch (RSNor.MSK)

## D Latch

The truth table and schematic diagram of the static D latch, also called Static D-Flip-Flop, are shown in Figure 47. The data input D is transferred to the output if the clock input is at level 1. When the clock returns to level 0, the latch keeps its last value. When performing the logic simulation, instability appears in outputs Q and nQ (Figure 6.6). This very high frequency oscillation is issued from the simultaneous change of Q and nQ at initialization phase. In analog simulation the parasitic oscillation almost disappear, although a fluctuation may be observed, called metastability

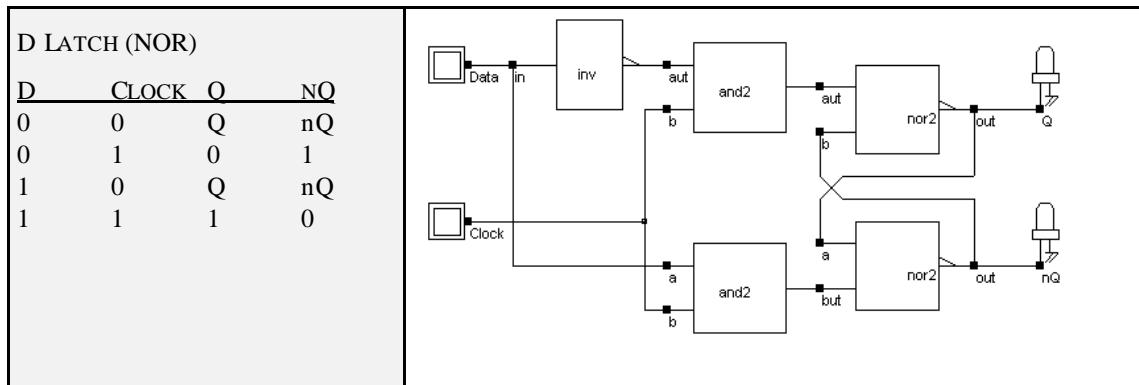


Fig. 6.5. The truth table and schematic diagram of a D Latch (File DLATCH.SCH).

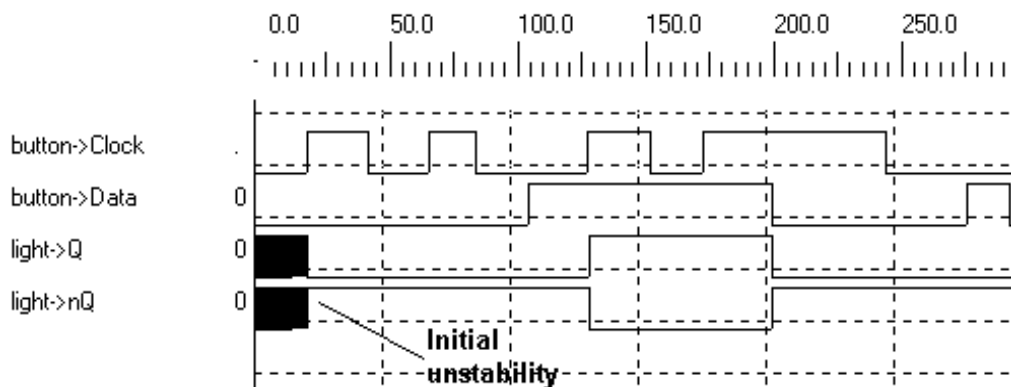


Fig. 6.6 Logic simulation of the D Latch (File DLATCH.SCH)

MANUAL DESIGN. Note that the NOR2-AND combination can be implemented in a complex-gate style. You may find useful to invoke the one line compiler to create successively one inverter  $\text{nd} = \text{d}$ , and two complex gates which include the AND/NOR cells using the syntax  $\text{Q} = /(\text{nQ} + (\text{nd} \cdot \text{h}))$  and  $\text{nQ} = /(\text{Q} + (\text{d} \cdot \text{h}))$ . Build the interconnections and run the Design Rule Checker. Assign a clock to CLK and a clock to DATA. An example of such an implementation can be found in the file "DLatchLevel.MSK". Its layout and corresponding simulation are illustrated in figure 6.7.

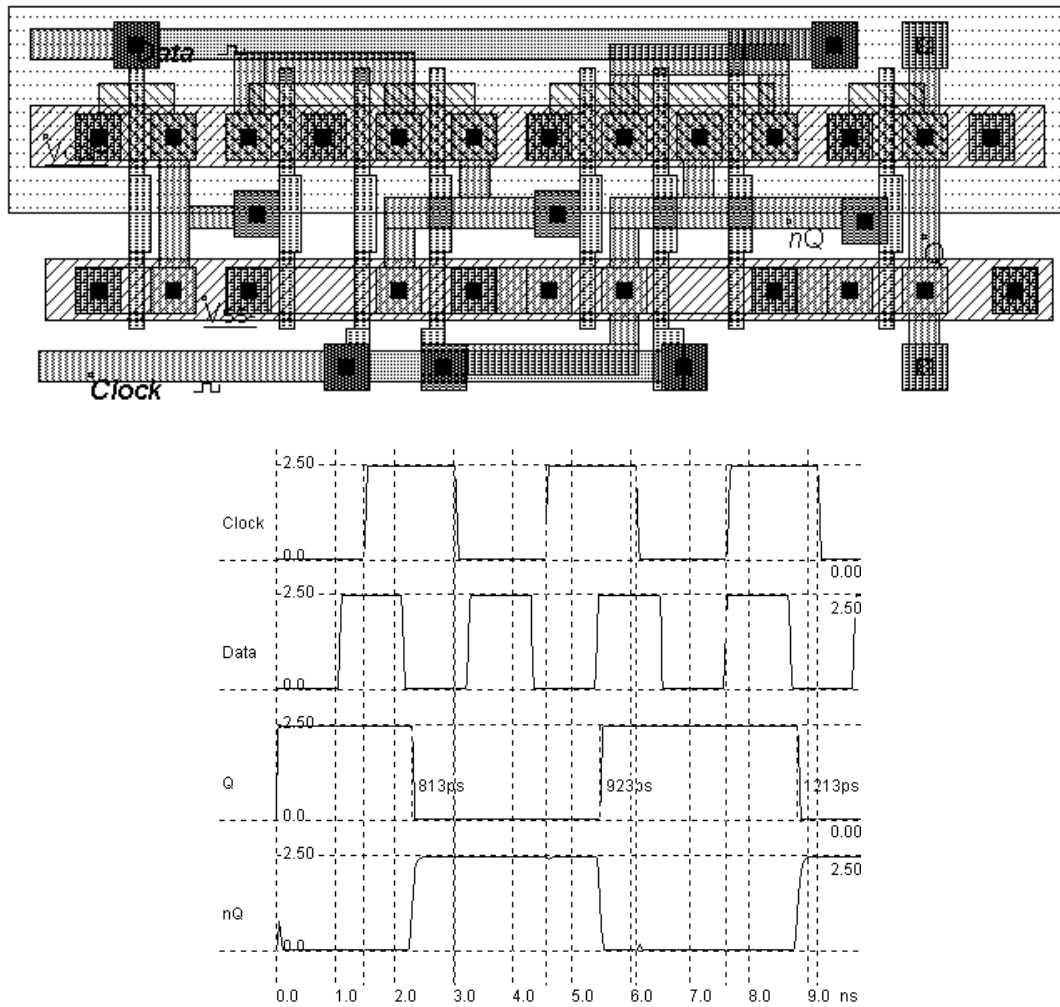


Fig. 6.7 Implementation and simulation of the D Latch (File DLatchLevel.MSK)

VERILOG COMPILING. Edit the file "DLATCH.SCH" using DSCH2. Generate the Verilog text by using the command **File -> Make Verilog File**. In Microwind2, click on the command **Compile -> Compile Verilog File**. Select the text file 'DLATCH.txt'. Click on **Compile**. When the compiling is complete, the resulting layout appears as shown in Figure 6.8.

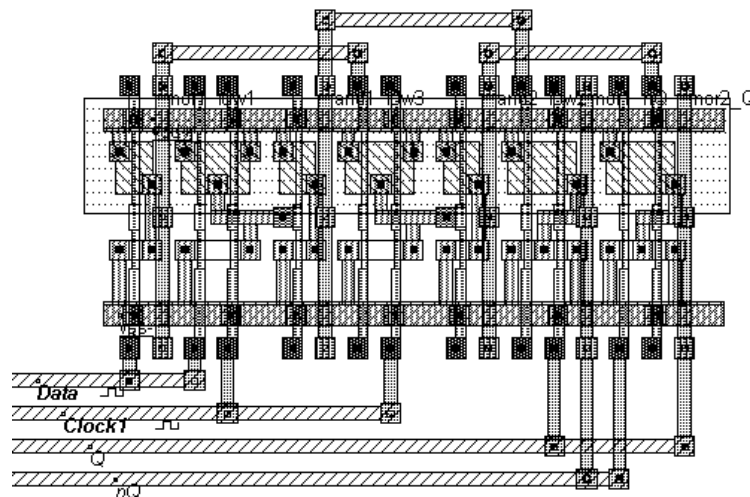


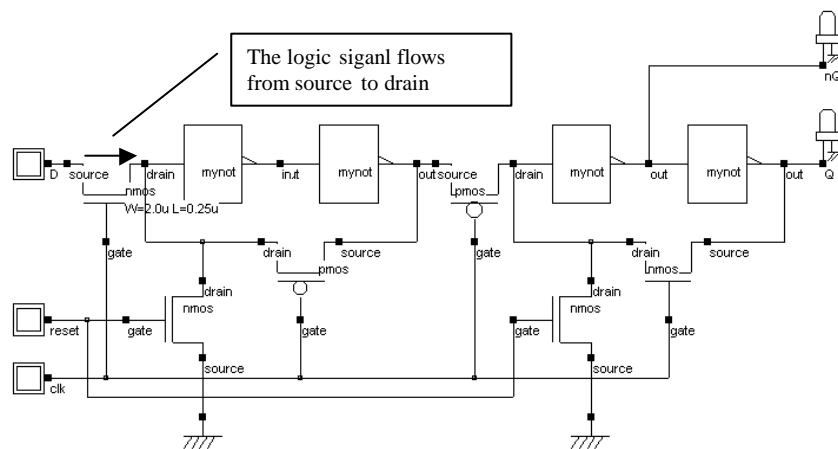
Fig. 6.7 Compiling of the DLatch (File DLatch.MSK)

The manual layout of figure 6.7 occupies a much smaller area than the automatic layout of figure 6.7, due to the use of complex-gate approach for the implementation of the AND/NOR combination, which saves space and makes the cell run faster.

## Edge Triggered Latch

The most common example of an edge-triggered flip flop is the JK latch. Anyhow, the JK is rarely used, a more simple version that features the same function with one single input D is preferred. This simple type of edge-triggered latch is one of the most widely used cells in microelectronics circuit design. The cell structure comprises two master-slave basic memory stages.

The most compact implementation of the edge-triggered latch is reported below. The schematic diagram is based on inverters and pass-transistors. On the left side, the two chained inverter are in memory state when the pMOS pass transistor P1 is on, that is when CLK=0. The two-chained inverters on the right side act in an opposite way. The reset function is obtained by a direct ground connection of the master and slave memories, using nMOS devices.



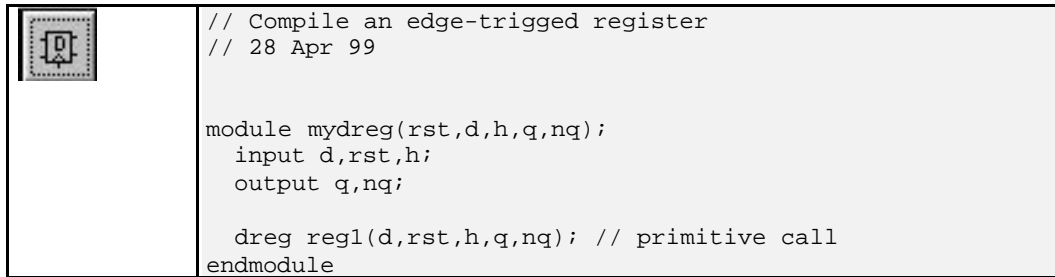
c

Figure 6.8 : The edge-triggered latch and its logic simulation (Dreg.MSK)

Notice that the logic model of the MOS device is not working the same way as for the real MOS switch. In the case of the logic implementation, the logic signal flows only from the source to the drain. This is not the case of the real switch where the signal can flow both ways.

### REGISTER COMPILING

Use the Verilog compiler to generate the edge-triggered latch, using the following text (dreg.txt), or by creating a schematic diagram including the “D” register symbol, in the symbol palette of DSCH2. As can be seen, the register is built up from one single call to the primitive “dreg”.



### SIMULATION PATTERNS

- RESET is active on a level 1. RESET is activated twice, at the beginning and later, using a piece-wise linear description included in the pulse property.
- CLK is a clock with 10ns at 0 and 10ns at 1.
- D is the data chosen here not synchronized with CLK, in order to observe various behaviors of the register.

```
// rst pwl 0 0 10 1 20 0 60 1 85 0
// h clk 10 10
// d clk 25 25
```

To compile the DREG file, use the command “Compile” → “Compile Verilog Text”. The corresponding layout is reported below. The piece-wise-linear data is transferred to the text “rst” automatically.

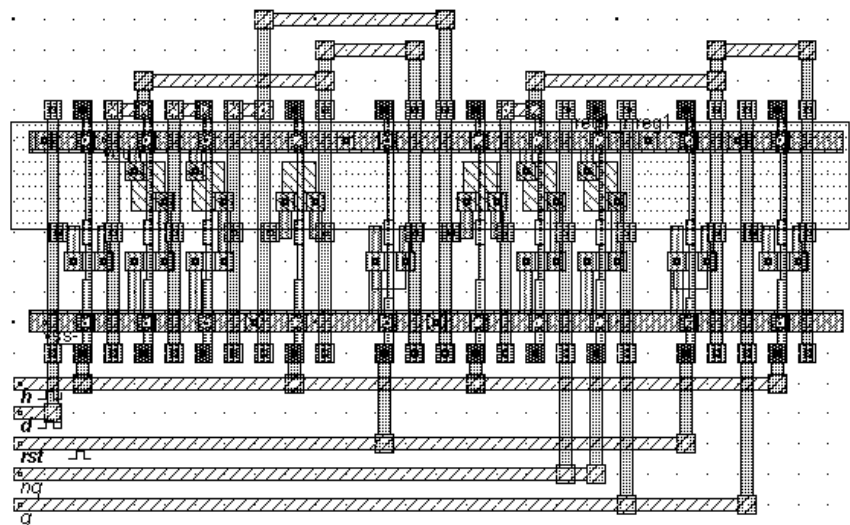


Fig. 6.9: Compiled version of the Edge-triggered D Flip Flop

The simulation of the edge-triggered latch is reported in figure 6.10. The signals Q and nQ always act in opposite. When RESET is asserted, the output Q is 0, nQ is 1. When RESET is not active, Q takes the value of D at a fall edge of the clock. For all other cases, Q and nQ remain in memory state. The latch is thus sensitive to the fall edge of the clock.



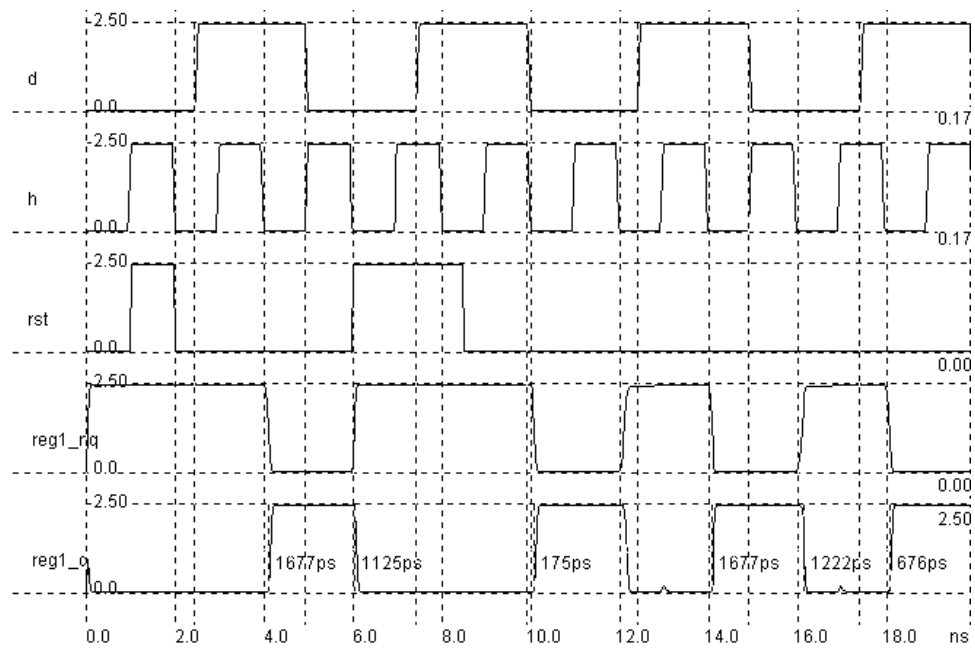


Fig. 6.10: Simulation of the DREG cell (DREG.MSK)

## Counter

The one-bit counter is able to produce a signal featuring half the frequency of a clock. The most simple implementation consists of a D flip-flop where the output  $nQ$  is connected to  $D$ , as shown in figure 6-11. In the logic simulation shown in figure 6-12, the clock "Clock1" changes the state of "Clock\_Div\_2" at each fall edge. The "RESET" is active high, and stuck the output to 0.

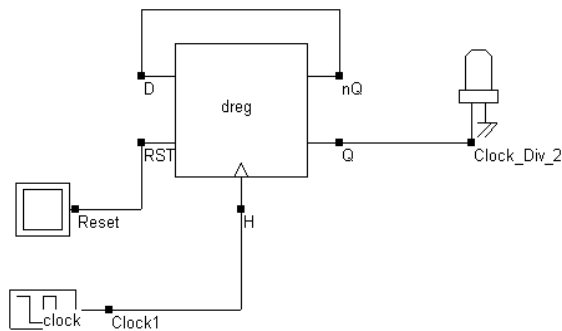


Fig. 6-11. Schematic diagram of the 2-bit counter (DivFreq.MSK).

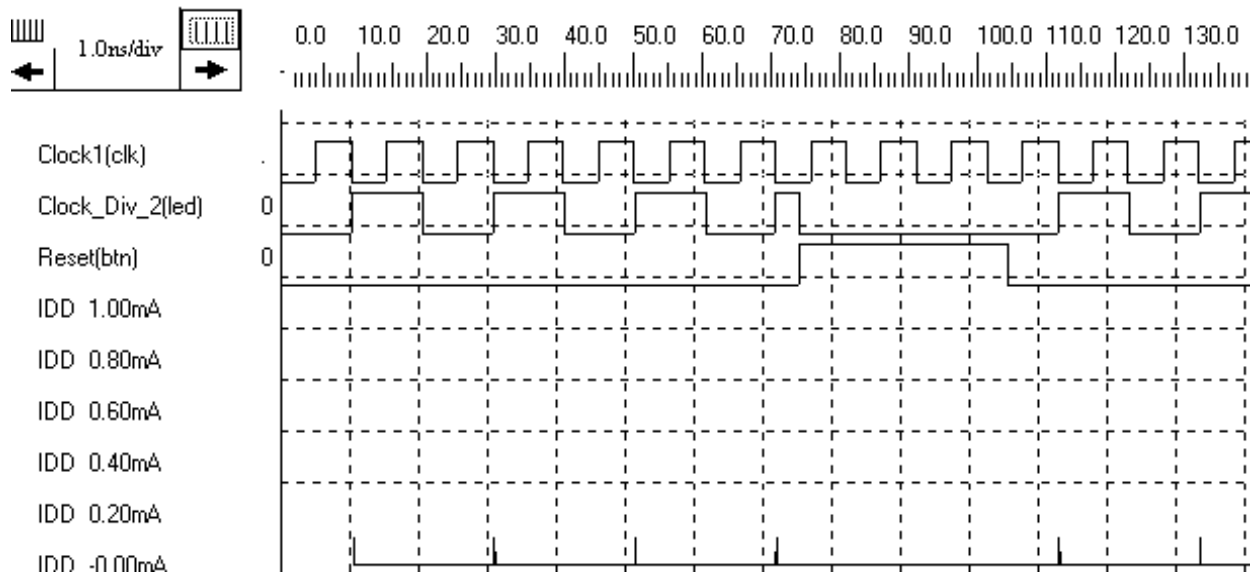


Fig. 6-12. Logic simulation of the divider-by-two (ClockDiv2.SCH)

### MAXIMUM OPERATING FREQUENCY

The most important parameter to be characterized in the clock divider is the maximum frequency  $f_{max}$  up to which the cell divides properly. Let us extract this frequency  $f_{max}$  using the compiled version of the clock divider.

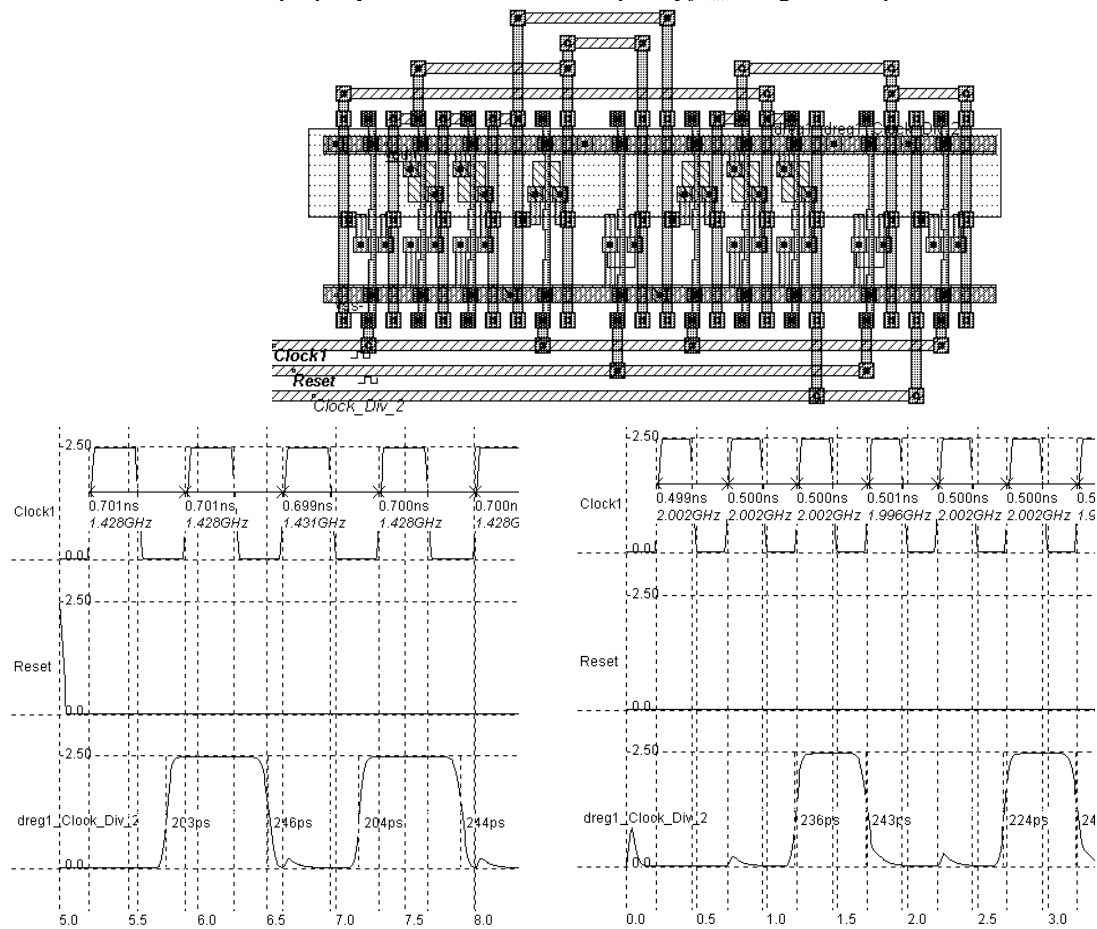


Fig. 6-13. Analog simulation of the divider-by-two and maximum operating frequency (ClockDiv2.MSK)

For that purpose, the schematic diagram of figure 6-11 is compiled and gives the layout reported in figure 6-13. Two simulations are performed. One operates at an input frequency around 1.4GHz, where the divider circuit works correctly. The frequency of “Clock1” is increased slightly until the division is becoming erratic, around 2GHz. A much better performance can be obtained when conducting the manual design of the clock divider.

## RAM Memory

The schematic diagram of the static memory cell used in High Capacity Static RAMs is given in Figure 6-14. The circuit consists of 2 cross-coupled inverters and two nMOS pass transistors. The cell has been designed to be duplicated in X and Y in order to create a large array of cells. Usual sizes for Megabit SRAM memories are 256 x 256 cells or higher. An arrangement of 4x4 RAM cells is also shown in figure 6-14. The selection line **Sel** concerns all the cells of one row. The lines **Data** and **nData** concern all the cells of one column.

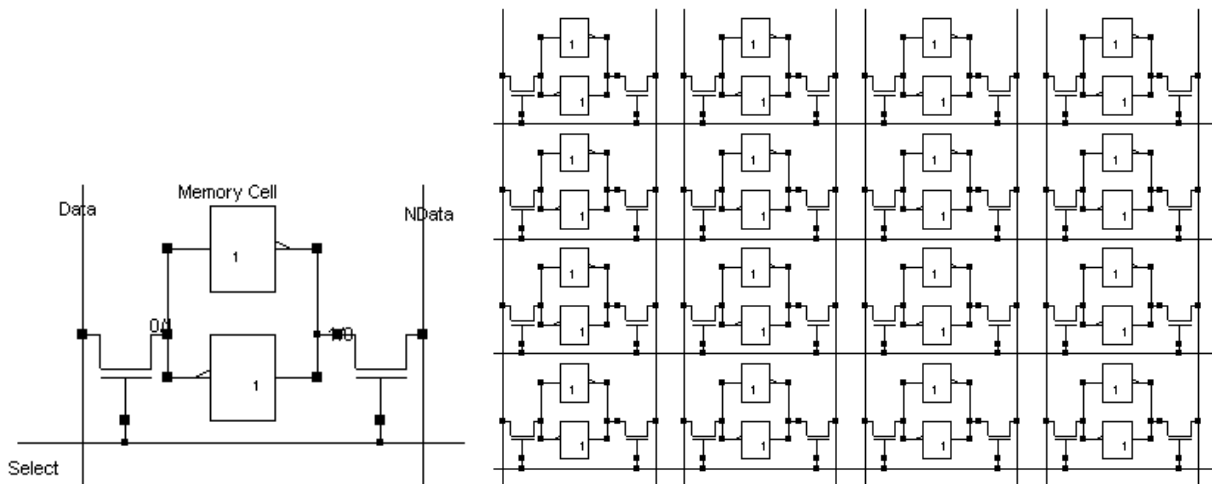


Fig. 6-14. The schematic diagram of the static RAM cell (RAM1.SCH).

The RAM layout is given in Figure 6-15. Click on **File → Open → RAM.MSK** to read it. The **Data** and **nData** signals are made with metal2 and cross the cell from top to bottom. The supply lines are horizontal, made with metal3. This allows easy matrix-style duplication of the RAM cell. The cross-section shows the nMOS devices and the connection to VSS using metal3, situated on the middle of the cell. The Data and nData lines, in metal2 are on both sides.

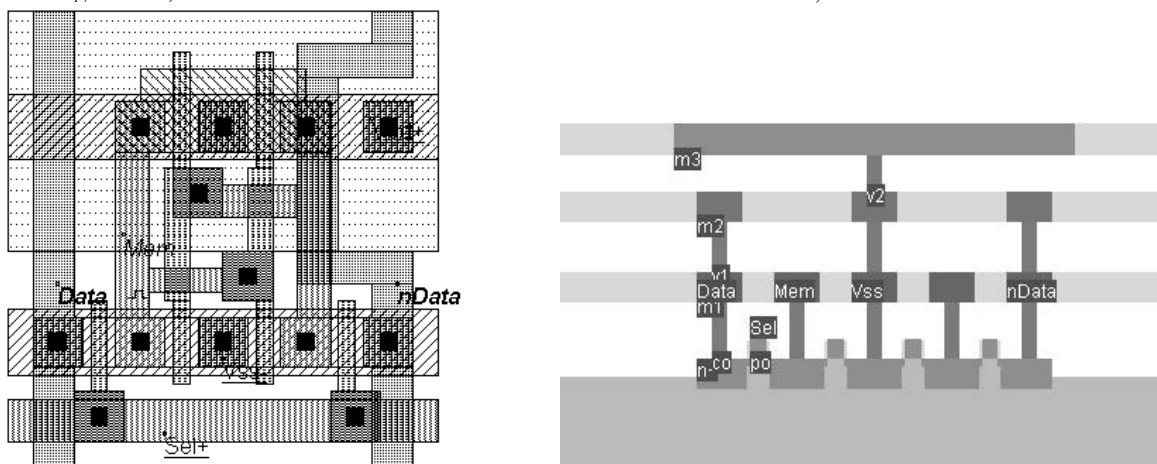


Fig. 6-15. The layout of the static RAM cell (RAM1.MSK).

**WRITE CYCLE.** Values 1 or 0 must be placed on **Data**, and the data inverted value on **nData**. Then the line **Sel** goes to 1. The two-inverter latch takes the Data value. When the line **Sel** returns to 0, the RAM is in a memory state.

**READ CYCLE.** In order to read the cell, the line **Sel** must be asserted. The RAM value propagates to **Data**, and its inverted value propagates to **nData**.

**SIMULATION.** The simulation parameters correspond to the write cycle in the RAM. The simulation steps describe in figure 6-16 are as follows:

- ❶ **Mem** reaches 1, after an unstable period (unpredictable value).
- ❷ **Data** gets to value 0 and **nData** to value 1.
- ❸ **Sel** is asserted. The memory cell **Mem** goes down to 0.
- ❹ **Data** gets to a value of 1 and **nData** gets to a value of 0.
- ❺ **Sel** is still asserted. The memory cell fights against Data=1 and surrenders (Mem=1).
- ❻ **Sel** is inactive. The RAM is in a memory state.

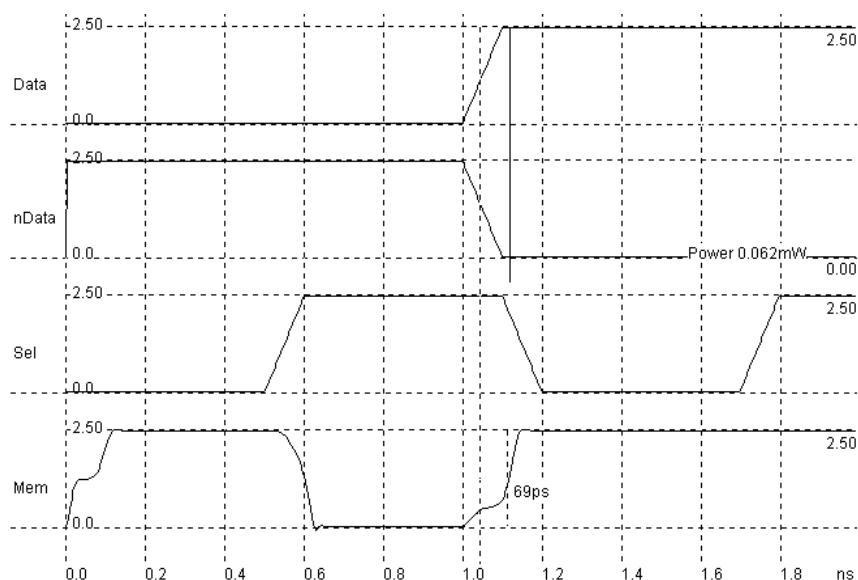


Fig. 6-16. Write cycle for the static RAM cell (RAM1.MSK).

## Complete RAM 4x4 Bit

You can duplicate the RAM cell into a 4x4 bit array using the command **Edit -> Duplicate XY**. Select the whole RAM cell and a new window appears. Enter the value « 4 » for X and « 4 » for Y into the menu. Click on « **Generate** ». The result is shown below.

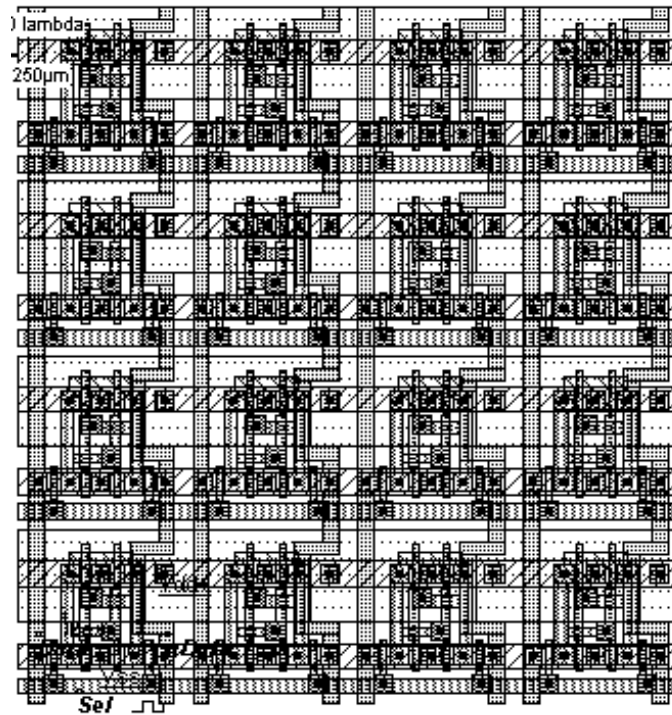


Fig. 6-17. Duplicating the RAM Cell in X and Y

The line decoder is based on the following schematic diagram. One line is asserted while all the other lines are at zero. In this circuit one line was picked out from a choice of four lines. Using AND gates would be an easy solution, but in order to save the inverter, we choose NOR gates with inverted inputs.

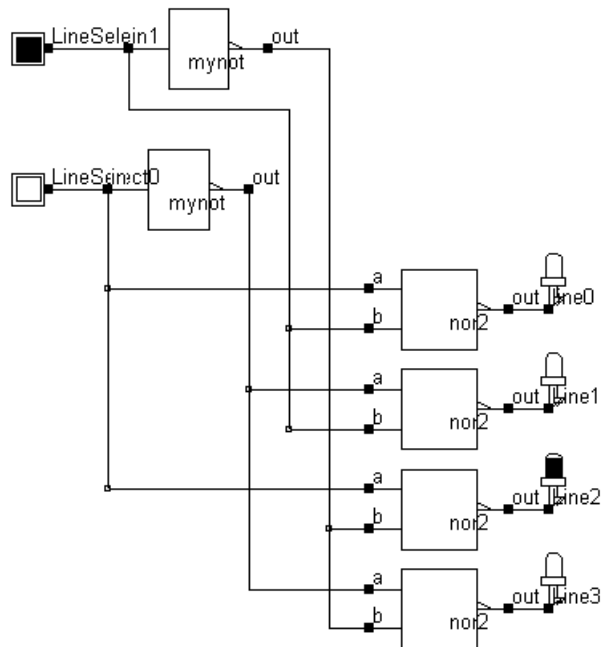


Fig. 6-18 A line selection circuit

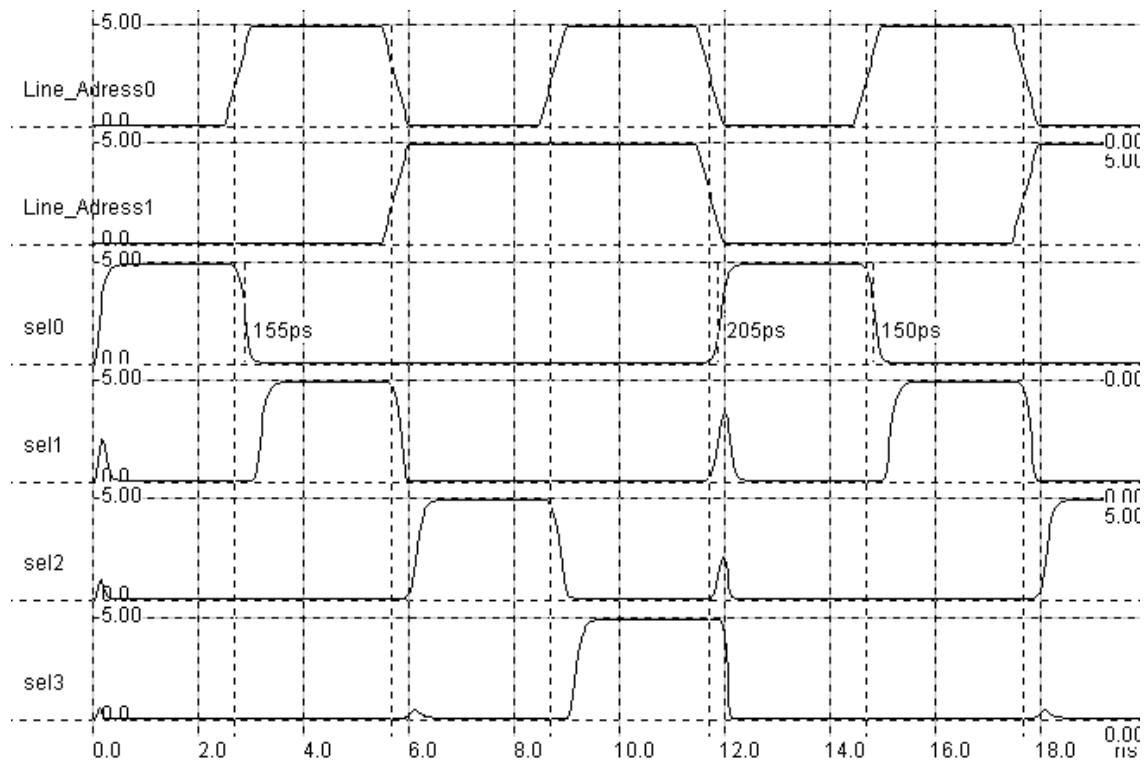


Fig. 6-19 . A line selection layout and its corresponding simulation (*RamLineSelect.MSK*)

The NOR gate height should be adjusted to that of the RAM cell height. When making the final assembly between blocks, the command **Edit -> Move Area** is very important. This command helps to move a selected block with a lambda step.

The row decoder is based on the same principles as those of the line decoder. The major modification is that the data flows both ways, that is firstly from the cell to the read circuit (Read cycle) and secondly from the write circuit to the cell (Write cycle). Fig. 55 proposes an architecture for this.

The n-channel MOS device is used as a switch controlled by the column selection. When the n-channel MOS is on and **Write** is asserted, the data issued from DataIn is amplified by the buffer, flows from the bottom to the top and reaches the memory. If **Write** is off, the 3-state inverter is in high impedance, which allows one to read the information.

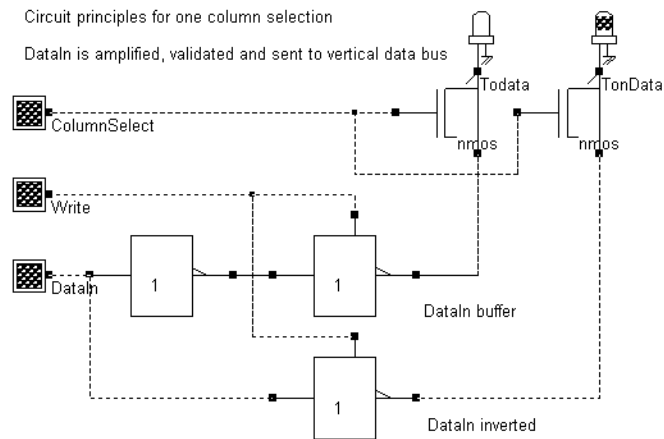


Fig. 6-20. Row selection and Read/Write circuit (RamColumn.SCH)

The final layout of the RAM 4x4 is proposed in Fig. 6-21. The simulation proposes the **read** and **write** cycles at a specific RAM cell address.

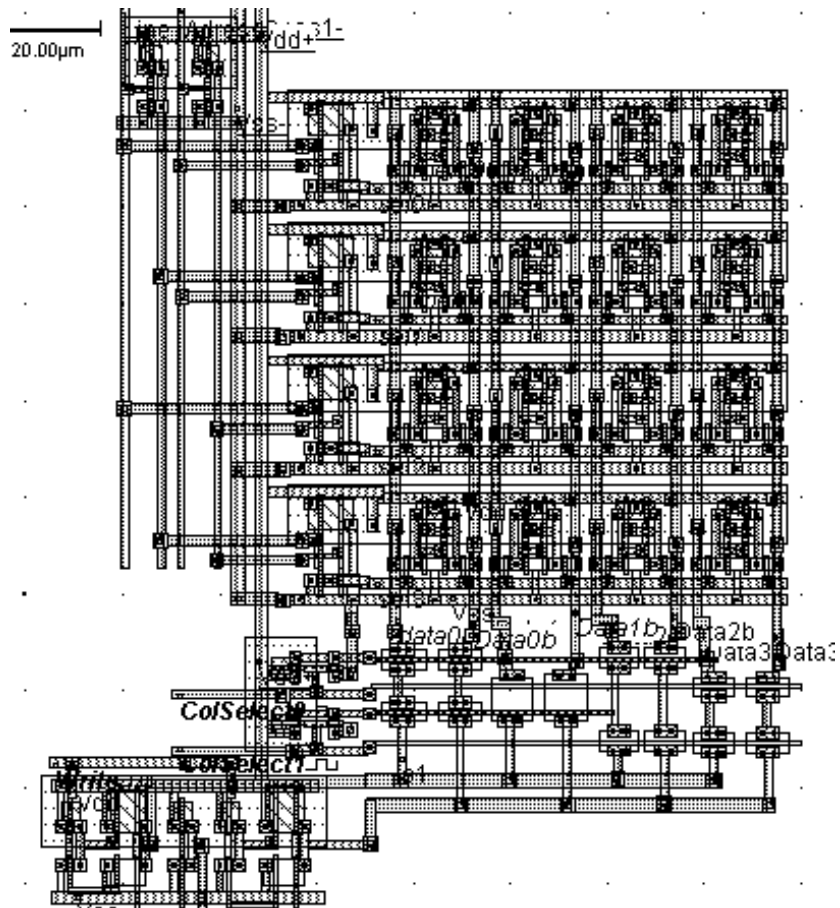


Fig. 6-21. RAM layout (RAM44.MSK)

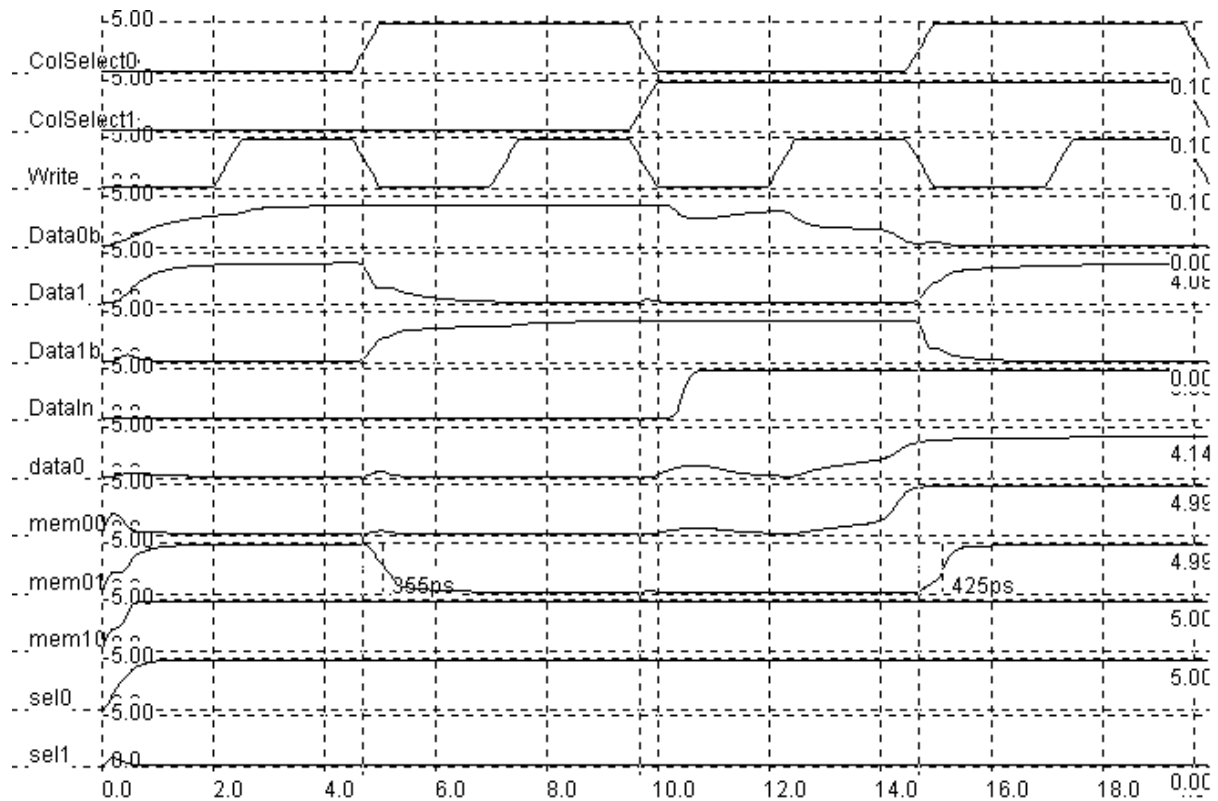


Fig. 6-22. RAM 4x4 layout and simulation (Ram44.MSK)

The simulation of Fig. 6-22 can be described as follows. A [00] fixed line selection selects the upper line, that way « sel0 » is asserted while all others are at 0. The memory cells mem00 and mem01 do not reach the same initial state : mem00 gets to 0 and mem01 is at 1. When DataIn is at zero, writing a zero has no effect on Mem00. But when the column selection changes, DataIn=0 is copied to Mem01.

When DataIn rises to 1 (t=10ns), and when write is 1, the memory cells change from 0 to 1. It is interesting to point out that the memory cell fights against the logic value before surrendering and changing its internal state.



