

8

Sequential Cell Design

This chapter details the structure and behavior of latches. The RS Latch, the D Latch and the edge-sensitive register are presented. The counters are also introduced, with an application concerning a 24-hour clock.

1. The Elementary Latch

Combinational circuits are able to add, multiply, shift, etc.. However, combinational circuits cannot store and retain values. The basis for storing an elementary binary value is called a latch. The simplest CMOS circuit is made from 2 inverters.

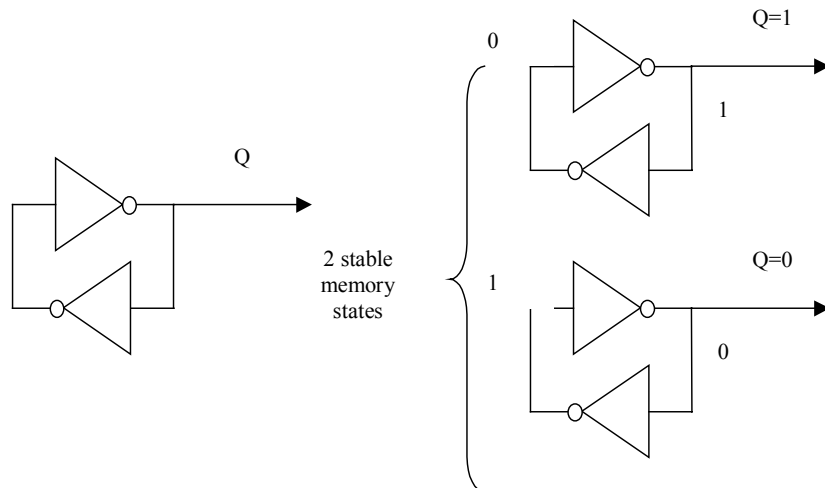


Figure 8-1: Elementary memory cell based on an inverter loop

Several techniques exist to modify the data inside the loop. One solution consists in adding a pass transistor, and make sure that the information coming from D overcomes the information retained by the feedback inverter. An other solution is based on NAND gate, which is extensively used in RS and D-latch structures. The *Set* and *Reset* signals are active low. A similar design, based on NOR gates is also shown in figure 8-2. The *Set* and *Reset* signals are active high in that circuit.

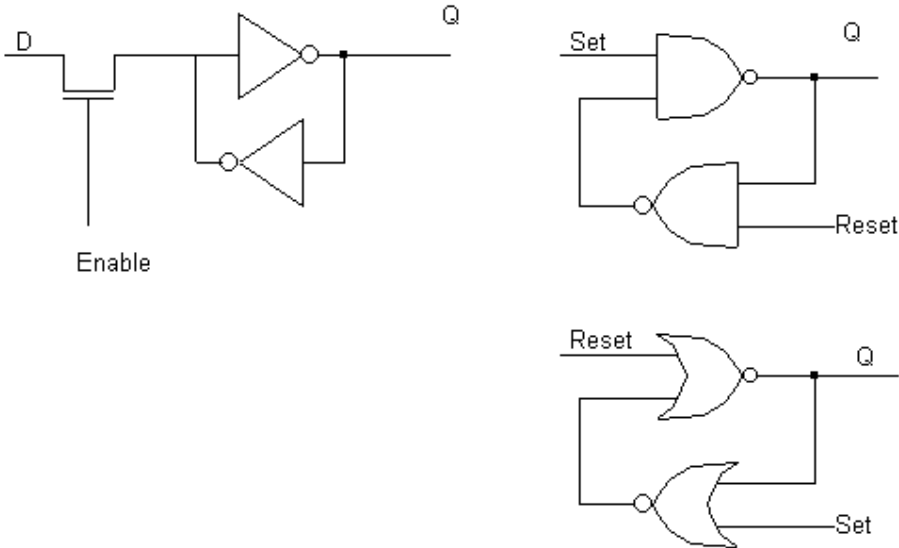


Figure 8-2: Methods to change the contents of the loop (latches.SCH)

2. RS Latch

The RS Latch, also called Reset-Set Flip Flop (RS FF), transforms a pulse into a continuous state. The RS latch can be made up of two interconnected NAND gates, as shown in figure 8-3. In the truth table, we see that the *Reset* and *Set* inputs are active low. The memory state corresponds to $Reset=Set=1$. The combination $Reset=Set=0$ should not be used, as it means that Q should be reset and set at the same time. In this case, $Q=nQ=1$.

RS Latch (NAND)

R	S	Q	nQ
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	nQ



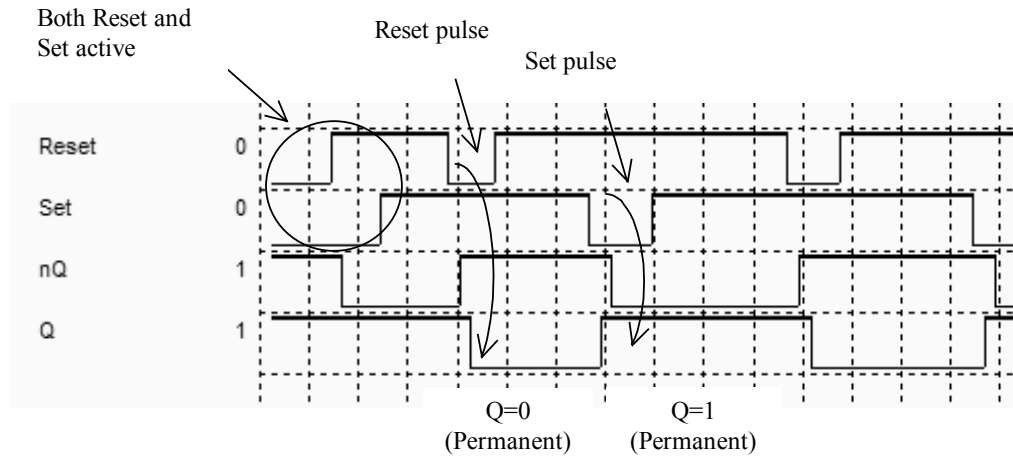


Figure 8-3: The RS-NAND latch and its typical simulation waveform (RSNand.SCH)

The simultaneous change from $Reset=Set=0$ to $Reset=Set=1$ (Figure 8.4) provokes what is called the meta-stable state, that corresponds to a high frequency oscillation. The meta-stability appears in the chronograms of figure 8-4 at the simultaneous rise of *Set* and *Reset*. At layout level, the parasitic oscillation does not exist, but a parasitic delay effect may be observed, up to 2 or 3 times the nominal gate delay.

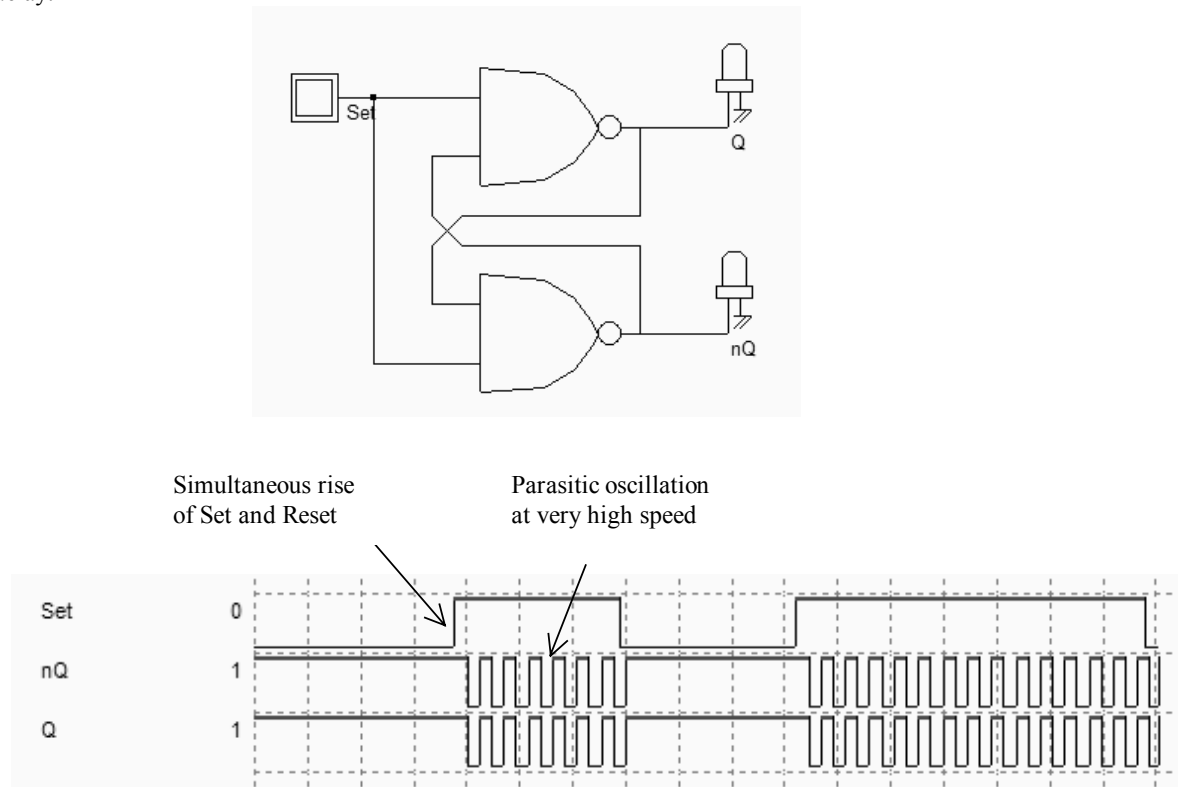


Figure 8-4: Illustration of the meta-stability at logical level (LatchMetaStable.SCH)

RS Latch (NOR)

R	S	Q	nQ
0	0	Q	nQ
0	1	1	0
1	0	0	1
1	1	0	0

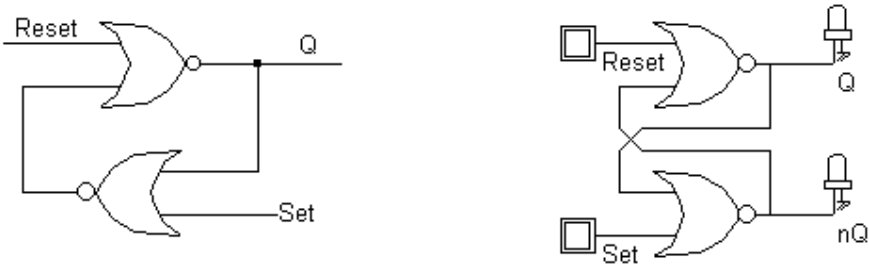


Figure 8-5. The truth table and schematic diagram of a RS-NOR latch (RsNor.SCH)

An alternative implementation of the RS latch is made from NOR gates (Figure 8-5). In that case, the *Reset* and *Set* inputs are active high. The cell transforms positive pulses into continuous states, as seen in the chronograms of figure 8-6.

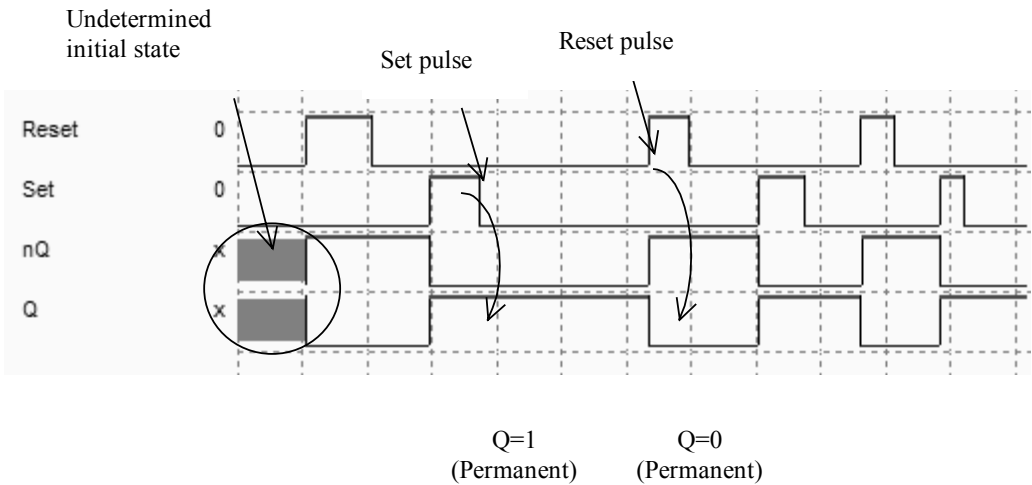


Figure. 8-6. The typical simulation of a RS-NOR latch (RsNor.SCH)

RS Latch 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 8-7 (RSNandManual.MSK).

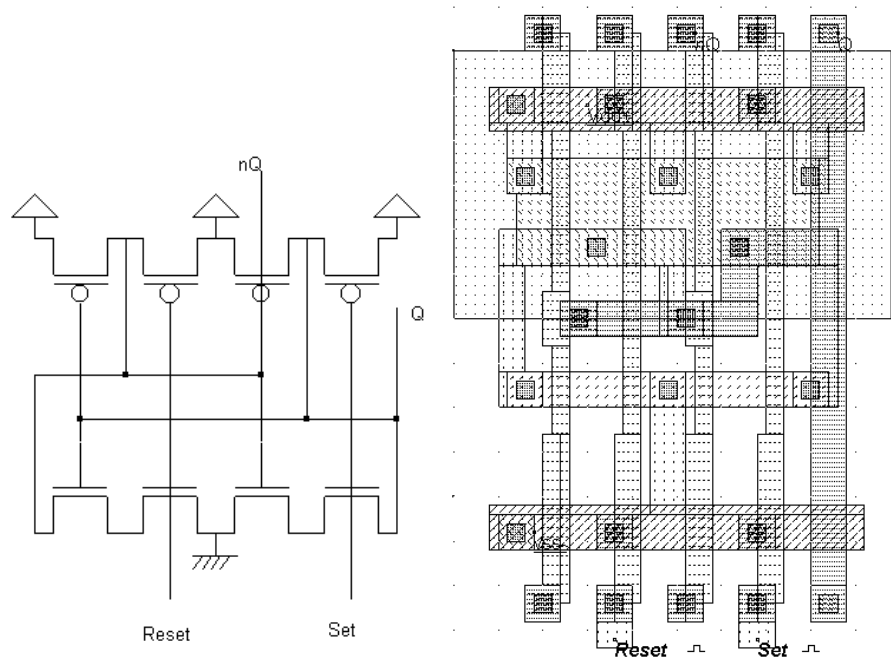


Fig. 8-7. Manual design of the RS-NAND (*RsNandManual.MSK*)

An alternative approach is to compile the layout directly from the schematic diagram. Using DSCH, create the RS Nand logic circuit, and generate the Verilog text by using the command **File → Make Verilog File**. In Microwind2, click on the command **Compile → Compile Verilog File**. Select the Verilog text file corresponding the RS-Nand gate.

```
module RSNand( Reset,Set,Q,nQ);
  input Reset,Set;
  output Q,nQ;
  nand #(23) nand2(Q,Set,nQ);
  nand #(23) nand2(nQ,Q,Reset);
endmodule

// Simulation parameters
always
#100 Set=~Set;
#200 Reset=~Set;
```

Fig. 8-8 The Verilog description of the RS Nand gate

Click on **Compile**. When the compiling is complete, the resulting layout appears as shown in figure 8-8. The NAND implementation of the RS gate is completed. Compared to the layout shown in figure 8-7, the compiled version is a little larger, due to the fact that the layout conversion tool did not merge the supply connections.

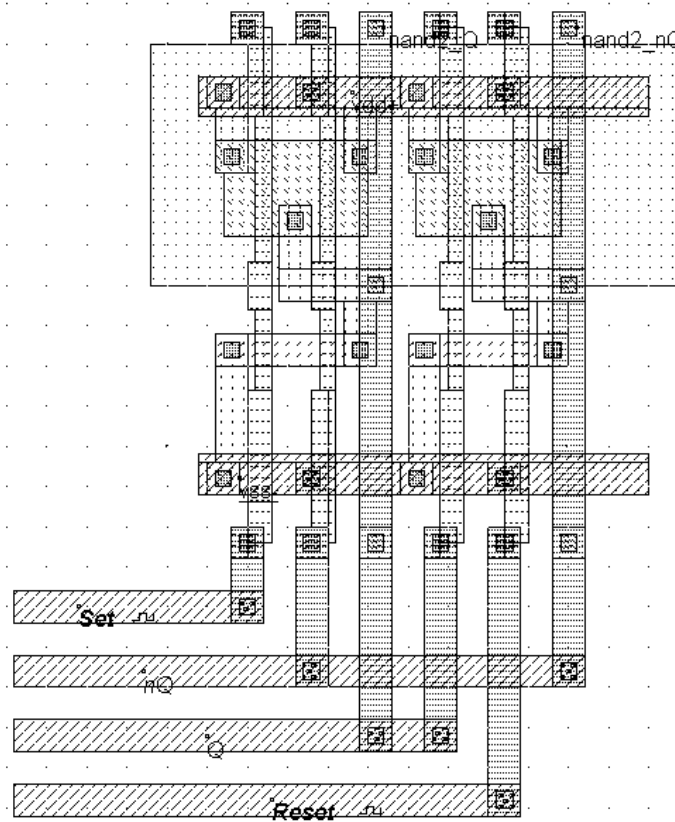



Figure 8-9. Automatic design of the RS-NAND (*RsNand.MSK*)

Control of the RS-Latch

If we keep the clock properties assigned by default to *Set* and *Reset*, we get chronograms which are inappropriate to validate the RS latch behavior, particularly the memory effect. A better approach consists in declaring pulse properties. For example, an active pulse is created on *Reset* followed by an active pulse on *Set*. The pulse parameters are shown in figure 8-10. To obtain a negative pulse, click the small icon placed in the middle of the window ().

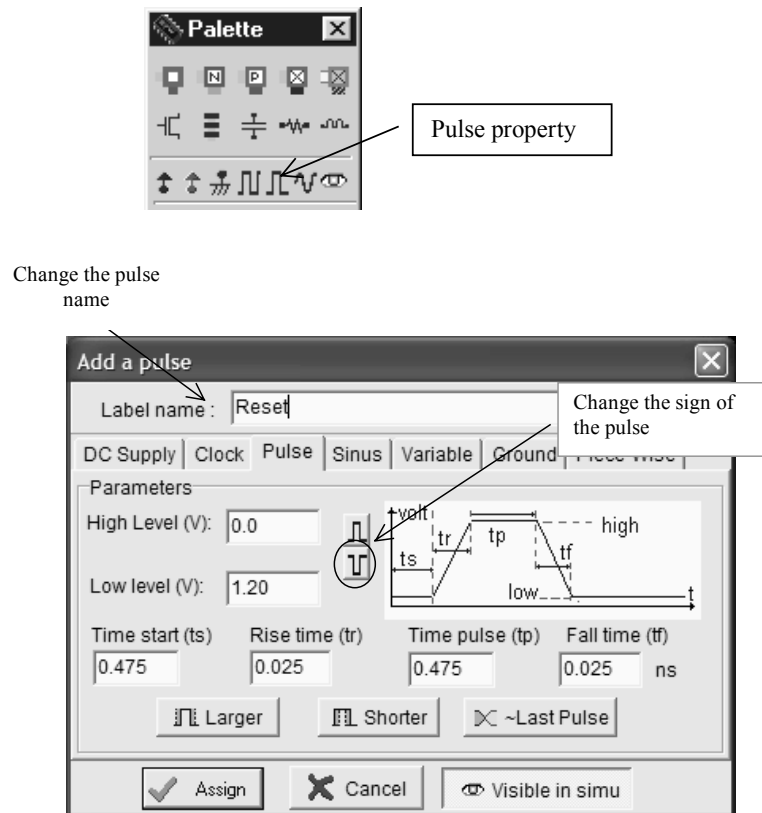


Figure 8-10. The pulse property in Microwind (RSNand.MSK)

The steps to add a pulse property are as follows. Select the Pulse icon in the palette. Click on the layout of the desired node. The screen shown in figure 8-10 appears. Change the label name, fix the sign of the pulse (positive by default), and click **Assign**.

Repeat the same procedure to assign a pulse property to node *Set*. The active level of the pulse is automatically delayed. Finally, click on **Simulate → Start Simulation**. The timing diagrams of figure 8-11 appear. A negative pulse on *Reset* turns *Q* to a stable low state. Notice that when *Reset* goes back to 1, *Q* remains at 0, which demonstrates the ability of the latch to transform a transient pulse into a permanent state. When a negative pulse occurs on *Set*, *Q* goes high, *nQ* goes low. The combination *Reset=Set=1* corresponds to the memory state.

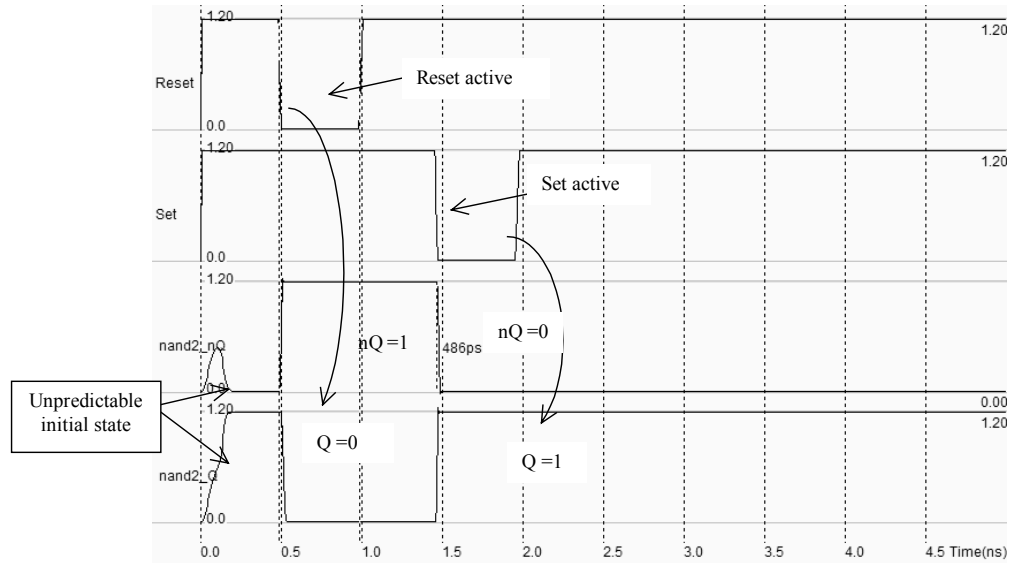


Figure 8-11. Simulation of the RSNAND latch (RSNand.MSK)

Minimum Pulse Width

One important characteristic of the RS cell is the minimum pulse width that provokes the *Set* or *Reset* effect. The pulse width is usually named t_w . One possible procedure consists in increasing the pulse width until the output Q is set to its proper value.

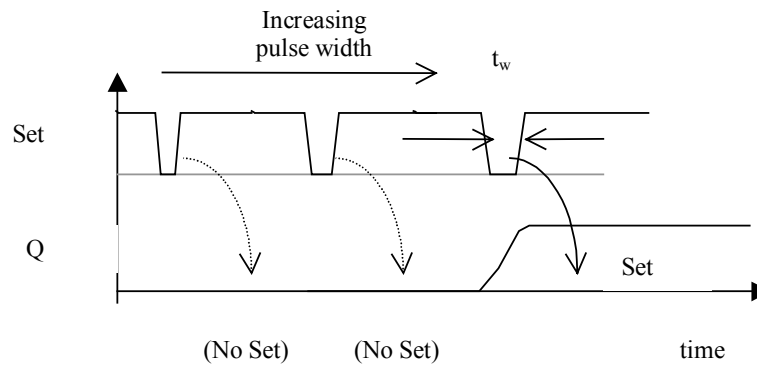


Figure 8-12. Definition of the minimum pulse width t_w that sets the latch correctly

To conduct this analysis, we program the input *Set* as a Piece-Wise-Linear signal (PWL <Gloss>). This property is derived from the pulse property, but not limited to one single shot. The signal is described using pairs of (time,voltage) information, listed in an array. To access to the PWL property, click the Pulse first, then change the selection in the property window.

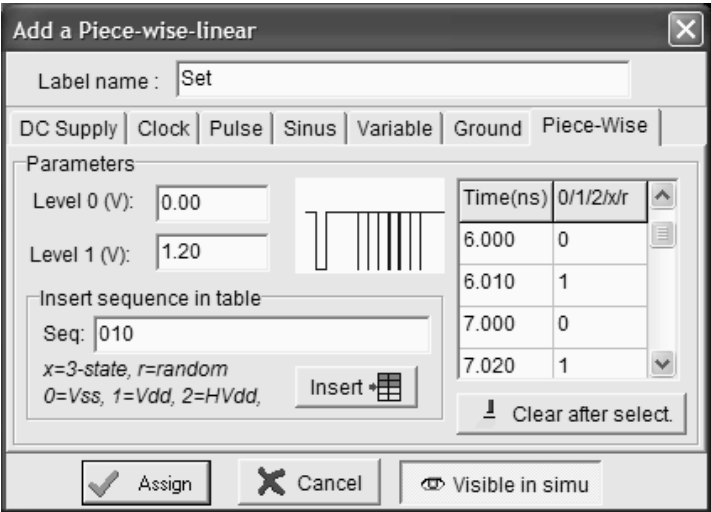


Figure 8-13. Setting a series of negative pulses with increased width, step 10ps on the Set signal using a Piece-Wise-Linear waveform (RSNandTw.MSK)

For this study, the latch outputs should be connected to a load in order to conduct the simulations in a realistic environment. A 10fF virtual capacitor is connected to Q and nQ . In figure 8-14, the pulse width which leads to a correct setting of the output Q is approximately 40ps. We used the Monte-Carlo simulation mode (**Simulation** → **Simulation Parameters**) to use a random set of technological parameters which accounts for the process variations. Depending on the process parameters, the result is slightly changed. The parameter t_w is usually specified in the data sheet of the RS latch, and more generally that of all latches. Notice that this value is strongly dependent on the loading conditions. We use the BSIM4 model for an accurate analog simulation. In that case, however, the results are similar with model LEVEL 3.

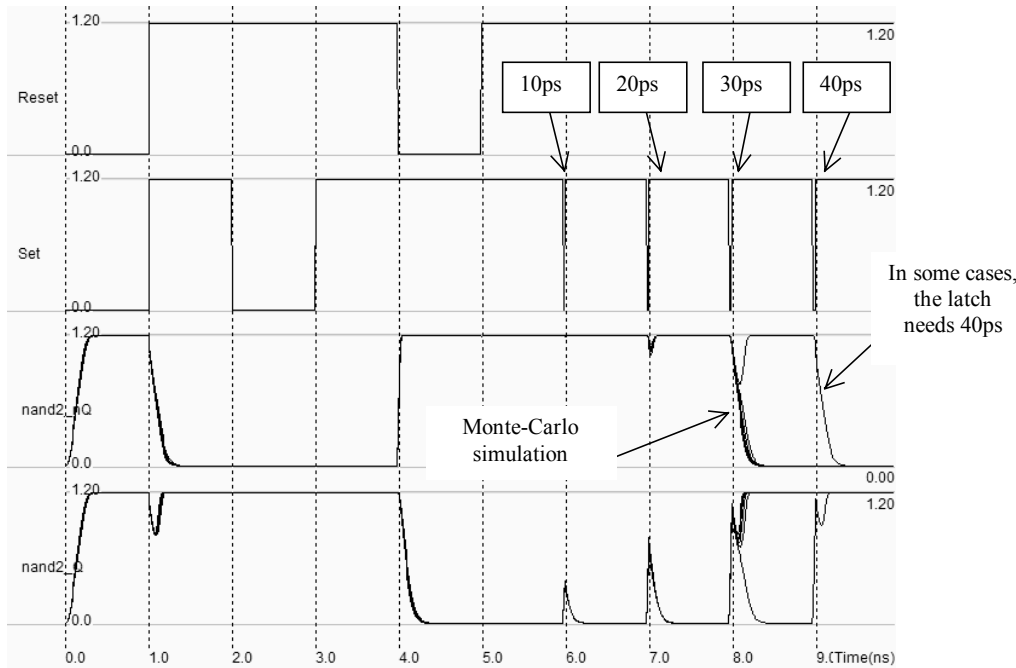


Figure 8-14 Finding the minimum pulse width of the RSNand latch (RSNandTw.MSK)

3. D Latch

The vast majority of CMOS integrated circuits use a single clock to synchronize the sequences of operations. This technique is called synchronous design, and is very popular as it allows automated and safe design. A well known D-latch controlled by a clock is shown in figure 8-15. When the clock input is high, the latch output Q follows the changes of the input D. The latch is transparent. Now, when the clock input goes low, the latch is in memory mode, meaning that it produces the value stored in the loop at the output Q. The latch holds the memory state as long as the two inverters are supplied.

D Latch

D	Clock	Q	nQ
0	0	Q	nQ
0	1	0	1
1	0	Q	nQ
1	1	1	0

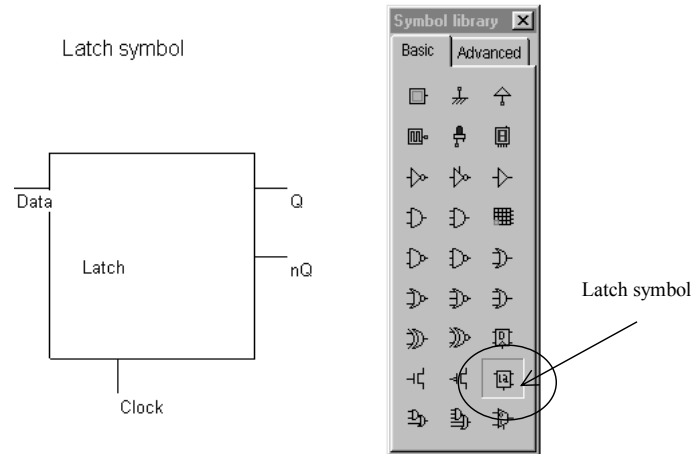


Figure 8-15 The truth-table and symbol of the D-latch

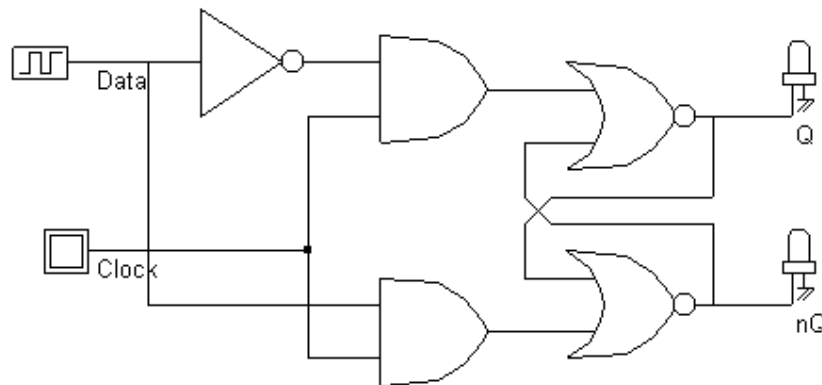


Figure 8-16 The schematic diagram of a D Latch (File DLATCH.SCH).

The truth table and the symbol of the static D-latch, also called Static D-Flip-Flop, are shown in figure 8-15. The schematic diagram of the D-latch is shown in figure 8-16. When performing the logic simulation, Q and nQ start with an undetermined state (Appearing in gray in Figure 8-17). Once the clock is active, Q and nQ turn to a deterministic state, as the data input D is transferred to Q , and its opposite to nQ . When the clock returns to level 0, the latch keeps its last value.

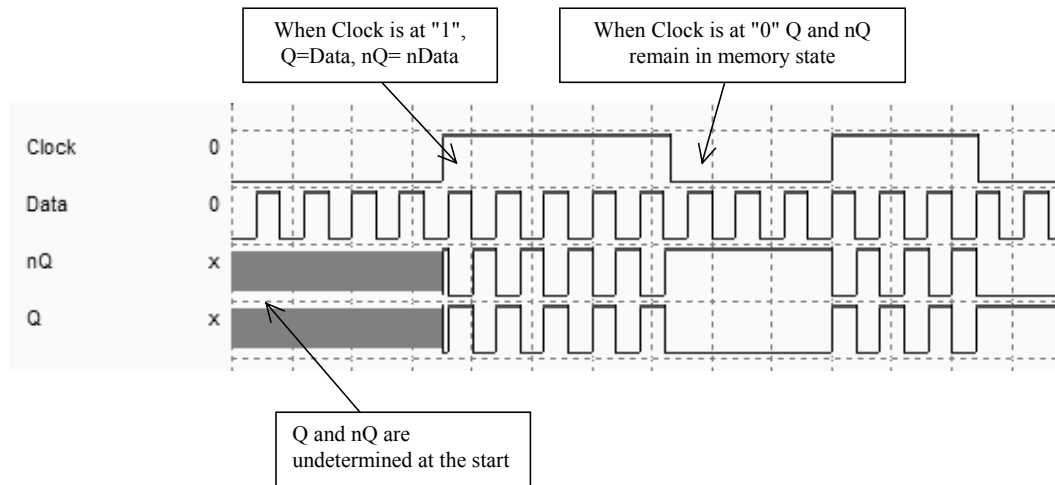


Figure 8-17 Logic simulation of the D Latch (File DLATCH.SCH)

A Complex Gate implementation of the D-Latch

In order to obtain a compact design, complex gates should be used rather than discrete AND cells which require a NAND gate with an Inverter. The two circuits are compared in figure 8-18: the direct implementation uses 22 MOS devices, the complex gate one only 14. The complex gate solution also leads to shorter propagation delay. Notice the description of the complex function using the expression:

$$s = \sim((a \& b) | c) \quad (\text{Equ 8-1})$$

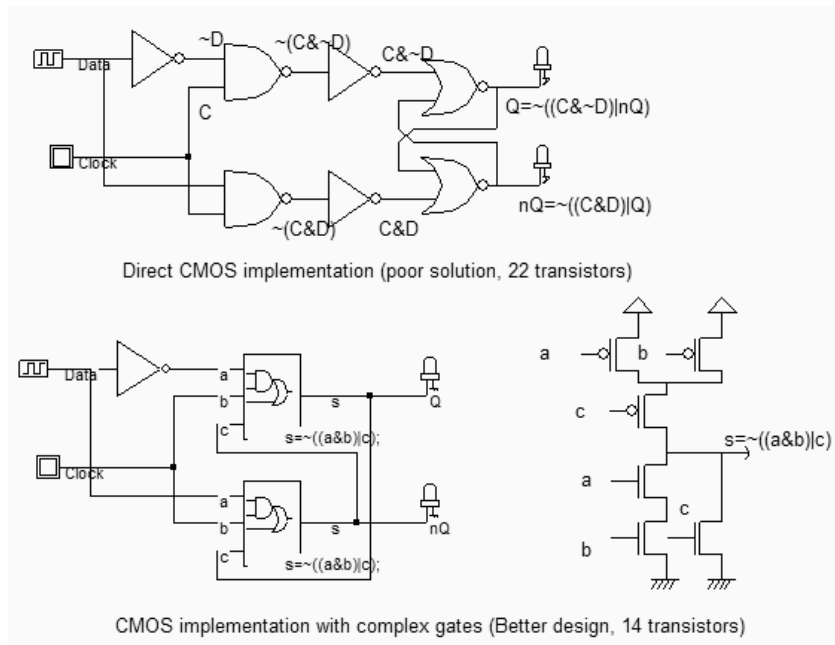


Figure 8-18 CMOS implementation of the D Latch (Dlatch.SCH)

Generate the Verilog text by using the command **File → Make Verilog File**. In Microwind2, click on the command **Compile → Compile Verilog File**. In the example given in figure 8-19, the file 'DLatchCompile.TXT' contains the D-Latch structural description, where we recognize the two complex gates and the Inverter. Also included in the Verilog text are the declarations of the control signals.

```
module DLatchCompile( Data,Clock,Q,nQ );
  input Data,Clock;
  output Q,nQ;
  assign      Q=~ (w1&Clock) |nQ);
  assign      nQ=~ ((Data&Clock) |Q);
  not inv(w1,Data);
endmodule

// Simulation parameters in Verilog Format
always
#1000 Data=~Data;
#1000 Clock=~Clock;
```

Figure 8-19 Generating a Verilog file corresponding to the D-latch (File DLatchCompile.TXT)

When the compiling is complete, the resulting layout appears as shown in Figure 8-20. The layout of the D-latch includes, from left to right, the two complex gates and the inverter. The internal wire is routed on the top of the cell, all I/Os being routed on the bottom.

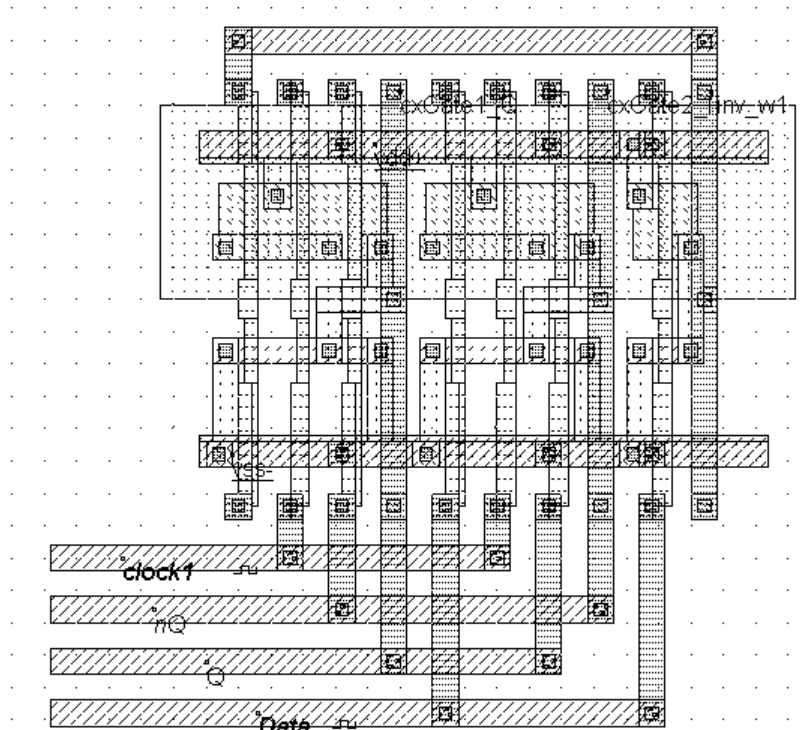


Figure 8-20 Compiling of the DLatch (File DLatchCompile.MSK)

The simulation is reported in figure 8-21. The default clocks assigned during compilation have been modified. The parameters of the clock *Data* have been changed to avoid the perfect synchronization with *clock1*, in order to watch several situations in one single simulation. When *clock1* is asserted, the logic information contained in *Data* is transferred to *Q*, its inverted value to *nQ*. When *clock1* falls down to 0, *Q* and *nQ* stay in memory state.

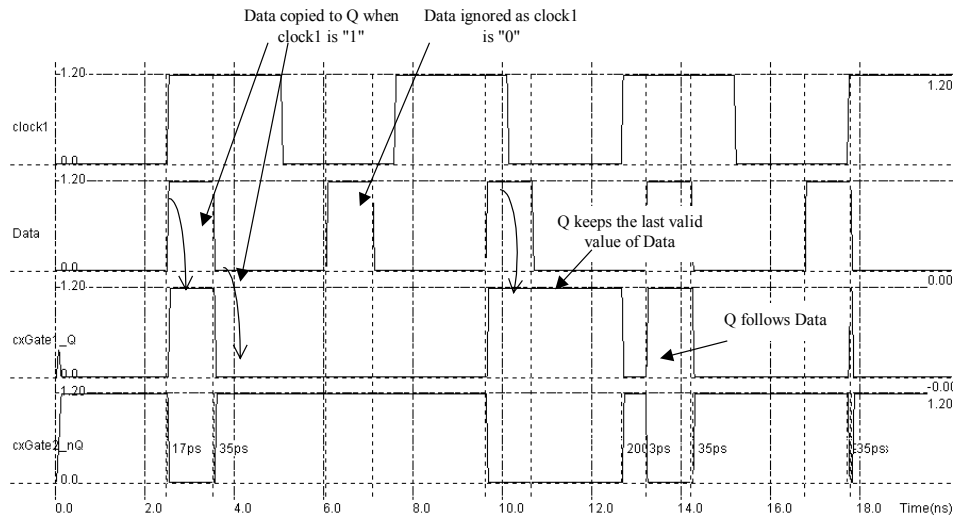


Figure 8-21: The compiled D latch at work (DlatchCompile.MSK).

Timing Analysis

When the latch is transparent, the delay between the change of *D* and the change of *Q* is named t_{PD} (Figure 8-22). The outputs *Q* and *nQ* are usually loaded with a 10fF capacitance to account for the connection to other logic gates through metal interconnects. The delay between the rise of *D* and the rise of *Q* may differ from the delay evaluated between the fall of *D* and the fall of *Q*.

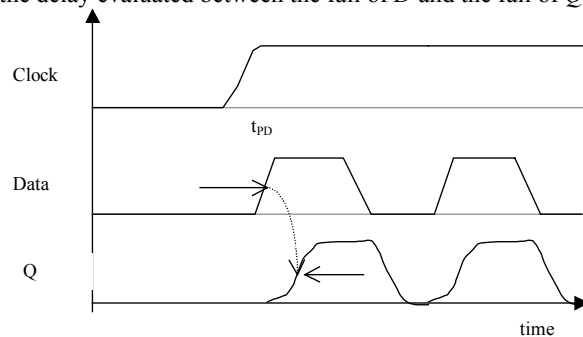


Figure 8-22 Definition of the propagation delay t_{PD} within D Latch

The minimum pulse width for the clock to pass the input *Data* to the output *Q* properly is labeled t_w . An illustration of the test setup to characterize t_w is given in figure 8-23. We change the clock property into a pulse property and increase the width of the clock pulse step by step, until the data passes to the output.

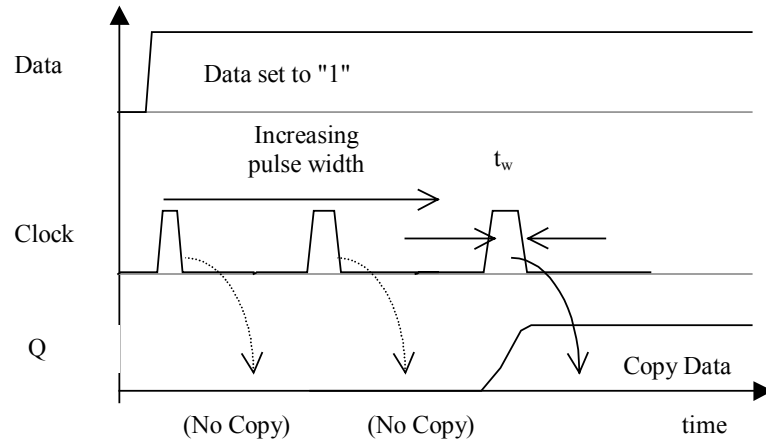


Figure 8-23 Simulation setup to characterize the minimum clock pulse of the D Latch

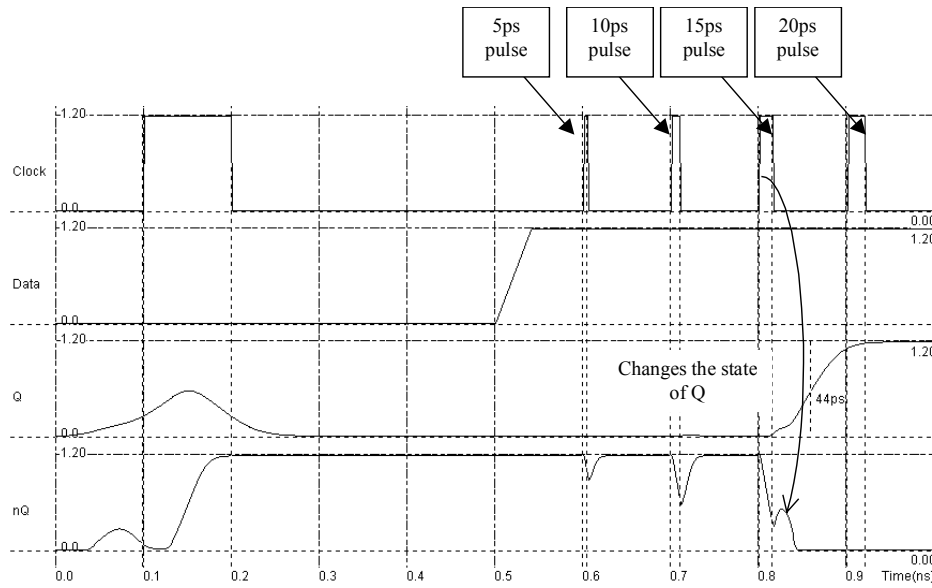


Figure 8-24 characterization of the minimum clock pulse of the D Latch (DlatchLevel.MSK)

We find that a 15ps pulse is required to enable the *Data* information to be transferred to *Q*. In this design, with a 10fF parasitic load, a safe value for t_w would be 20ps. However, the latch should be tested in extreme conditions (Worst case technological parameters, worst case temperature), and the largest value of t_w should be kept. In figure 8-24, the delay t_{PD} between the rise of *D* and the rise of *Q* is around 45ps.

Compact D-Latch Layout

A very compact design of the D-latch is obtained by using the original two-inverter memory loop and by adding the minimum hardware to change its state : one n-channel pass transistor to inject the new *Data*, and one p-channel to perform the feedback, or to suppress the loop effect when writing a new data. The resulting schematic diagram is shown in figure 8-25. Only 6 MOS devices are required to construct the D-latch. Recall that the complex gate implementation was based on 14 devices.

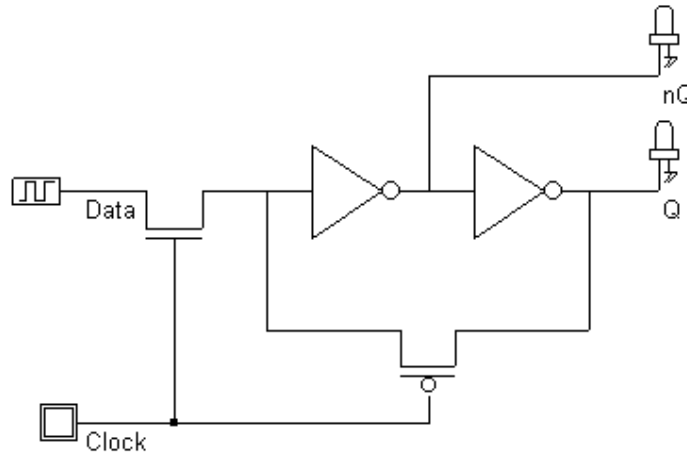


Figure 8-25: The very compact D-latch including 2 inverters and 2 pass transistors (Dlatch.SCH).

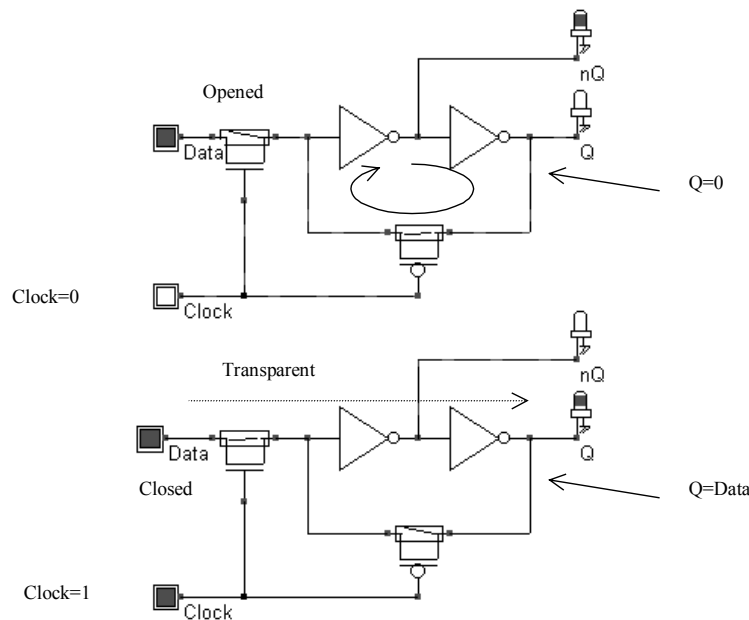


Figure 8-26: Simulation of the very compact D-latch (Dlatch.SCH).

An illustration of the latch at work is given in figure 8-26. When clock is at 0 (Upper configuration), the loop is active thanks to the pMOS, and the Data information is isolated from the loop. When clock is at 1 (Lower configuration), the latch is transparent, and the loop is broken by the pMOS.

Usually, latches proposed in cell libraries propose designs based on transmission gates rather than single pass transistors. The single pass transistor approach leads to more compact designs. However, the voltage amplitude is degraded, so the noise margin is reduced, and the switching speed is slower. Consequently, low power designs would prefer pass transistor approach, while high speed circuits would use pass transistors.

D-Latch Limitations

The main limitation of the D-latch is its inadequacy to build shift registers or counters. On a positive level of the clock, the whole series of D-latches is transparent to the input data.

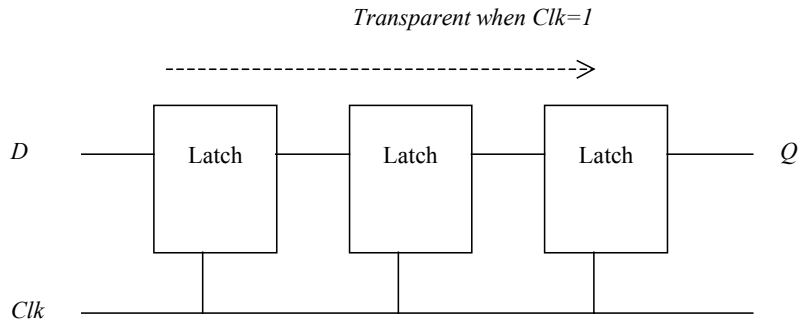


Figure 8-27: Incorrect shift register design using level sensitive D latch.

4. EDGE-TRIGGERED D-Register

To overcome the limitation of latches when trying to build registers, we use edge triggered latches, where the information flows from the input D to the output Q only at a rise edge of the clock (Figure 8-28). The latch is commonly known as D-Flip Flop (DFF) or D-register (Dreg).

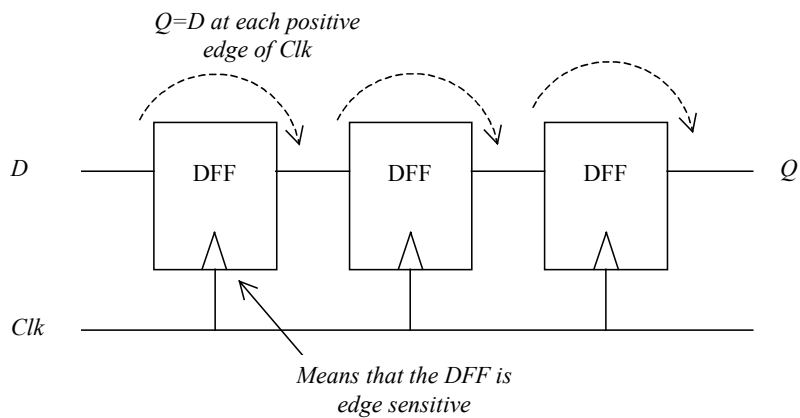


Figure 8-28: Correct shift register design using edge triggered flip-flop.

A well known edge-triggered flip flop is the JK latch. However, the JK is rarely used due to its complexity. In practice, 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.

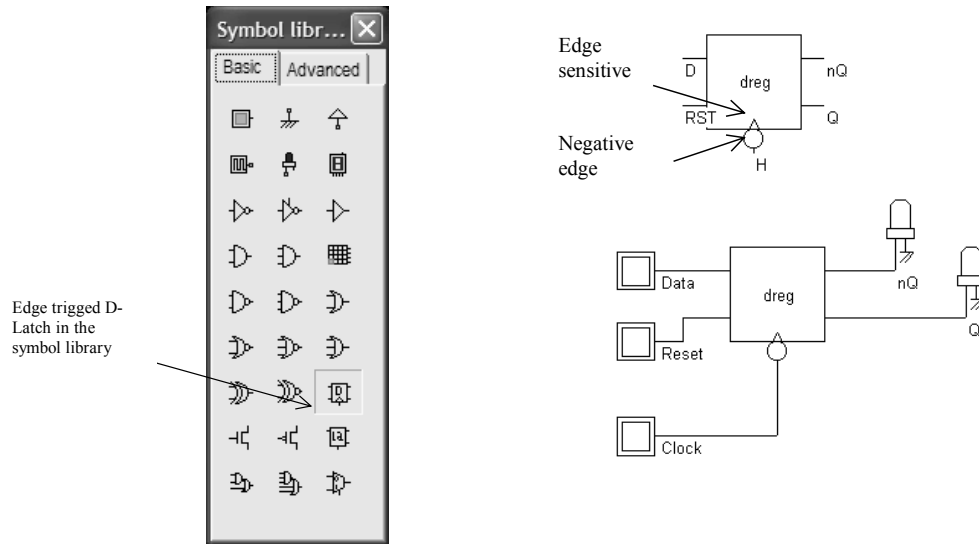


Figure 8-29: The edge triggered D-Flip Flop symbol

The symbol of the Edge-triggered D-flip flop may be found in the symbol palette at the location shown in figure 8-29. The symbol and its typical connection are also given in that figure. Notice the triangle at the clock input that recalls the sensitivity of outputs Q and nQ to the edge of the clock. The circle indicates that this Dreg cell is sensitive to a fall edge of the clock.

Behavioral Model of the DReg

In DSCH, the Dreg symbol is declared by a behavioral model shown in figure 8-30. Notice the difference between previous Verilog descriptions, where the structure of the cell was explicit (AND, Inv, complex gates, XOR, etc..), and this type of declaration which is purely a software description. At that stage, we do not know how the cell is constructed. Another example of behavioral model is the 8051 micro-controller described at the end of chapter 7. The behavioral models are very attractive for fast simulations, but do not accurately account for the physical effects such as delay or consumption.

In the behavioral description of table 8-1, the q and nq output signals are declared as registers. This specific variable is equivalent to a memory cell. In DSCH, a register *reg* is assigned an elementary memory, which may contain three possible values "1", "0" or "x". The line "always @(negedge clk)" is equivalent to a test on a fall edge of the signal *clk* (Figure 8-30). In that case, the instructions between the begin and the end keywords are executed. The constant "default_delay" has a value which is updated depending on the loading conditions and the technology. Notice the asynchronous reset of the Dreg when $rst=1$.

```
MODULE DReg(d,clk,rst,q,nq);
input d,clk,rst;
output q,nq;
reg q,nq;

always @(negedge clk)
```

```

begin
  #(default_delay) q=d;
  #(default_delay) nq=~d;
end
if (rst)
begin
  q=0;
  nq=1;
end
end
endmodule

```

Table 8-1: Behavioral model of the edge-sensitive D-flip flop used in DSCH

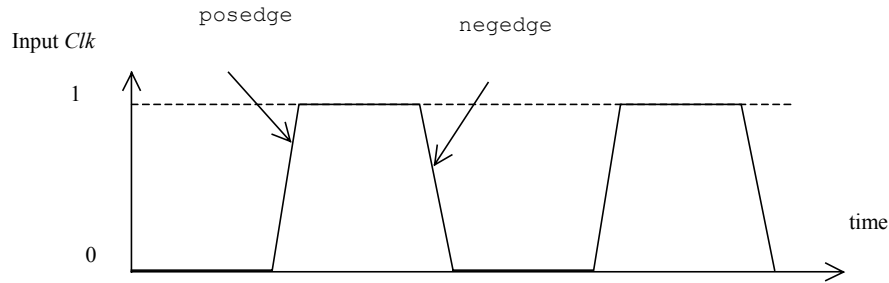


Figure 8-30: Definition of clock edges in Verilog

Schematic Diagram of the DReg

One very compact implementation of the edge-triggered Dreg is reported below. For clarity, the *Reset* circuit has been removed. The schematic diagram is based on inverters and pass-transistors. It is constructed from two memory loop circuits in series. The cell structure includes a master memory cell (left) and slave memory cell (right).

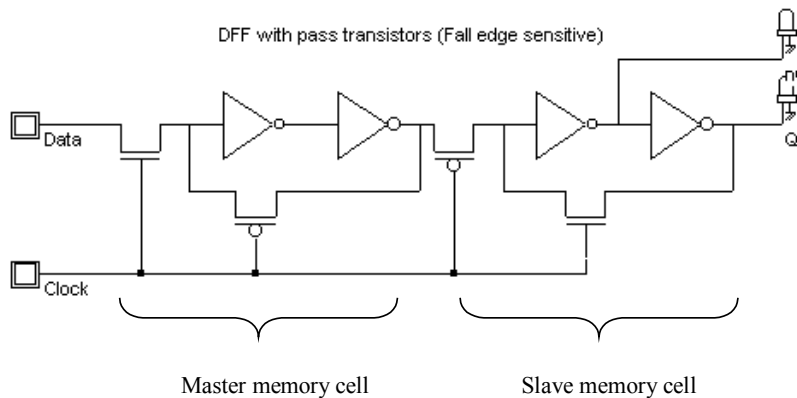


Figure 8-31: A compact implementation of the edge-sensitive Dreg (Dreg.SCH)

In figure 8-32, *clock* is high, the master latch is updated to a new value of the input *D*. The slave latch produces the previous value of *D* on the output *Q*. When *clock* goes down, the master latch turns to

memory state. The slave circuit is updated. The change of the clock from 1 to 0 is the active edge of the clock. This type of latch is a negative edge flip flop.

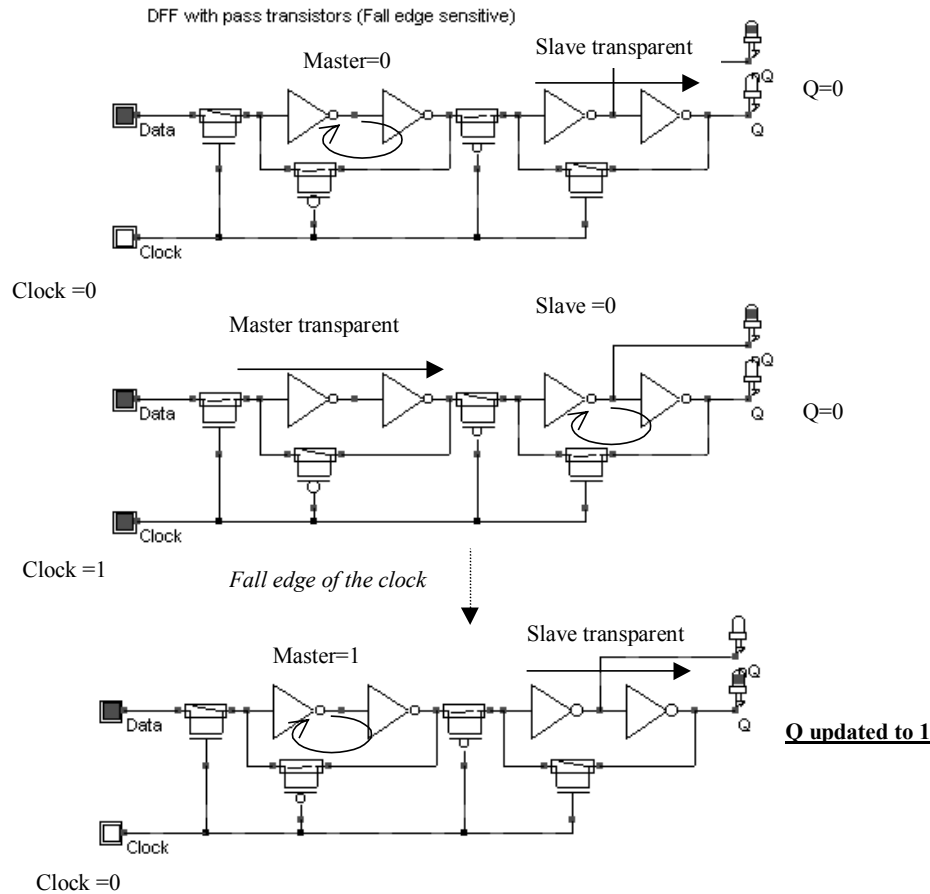


Figure 8-32 The edge-triggered latch and its logic simulation (Dreg.MSK)

Adding a Reset function to the DReg

The reset function is obtained by a direct ground connection of the master and slave memories, using nMOS devices. This added circuit is equivalent to an asynchronous Reset, which means that Q will be reset to 0 when *Reset* is set to 1, without waiting for an active edge of the clock.

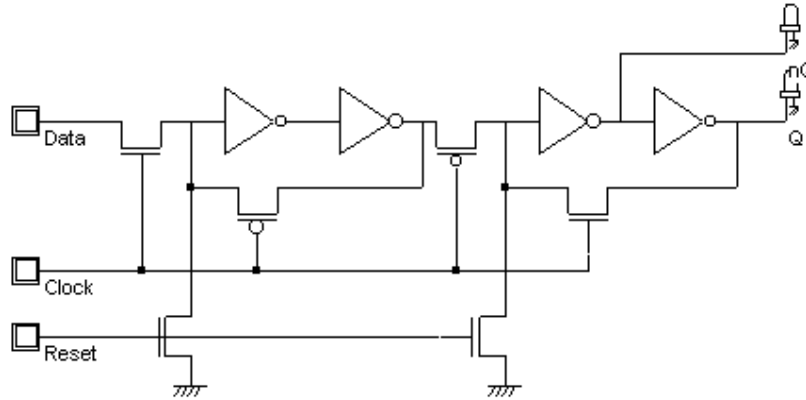


Figure 8-33 Reset function added to the DReg (Dreg.MSK)

Timing Analysis

Three delay parameters are important in the case of Dreg cells (Figure 8-34): the setup time t_{SU} , between a valid change of D and the active edge of the clock, the hold time t_H , between the active edge of Clock and a change of $Data$, and the propagation delay t_{PD} , between the active edge of $Clock$ and an updated setup of Q .

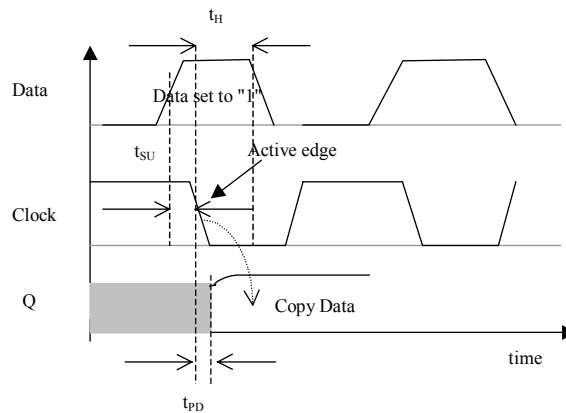
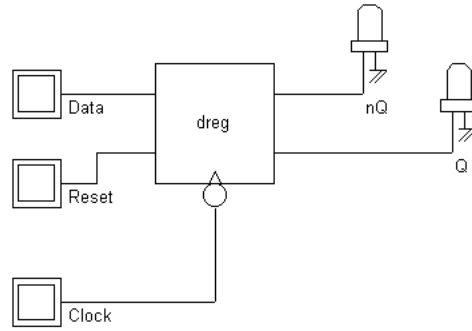


Figure 8-34 Important timing parameters in the DReg cell

Implementation of the DReg

We create a schematic diagram including the register symbol proposed in the symbol palette of DSCH. Three buttons are added to the circuit to control the inputs *Data*, *Reset* and *Clock*, as shown in figure 8-35. Using the command **Make Verilog File**, we create the corresponding Verilog description. As can be seen, the register is built up from one single call to the primitive “dreg”.



```

module dregCompile( Clock,Reset,Data,Q,nQ);
  input Clock,Reset,Data;
  output Q,nQ;
  dreg #(19) dreg1(Q,nQ,Data,Reset,Clock);
endmodule

// Simulation parameters in Verilog Format
always
#1000 Clock=~Clock;
#2000 Reset=~Reset;
#3000 Data=~Data;

```

Figure 8-35 The Verilog text corresponding to the Dreg circuit (DregCompile.SCH)

In Microwind, the Verilog text is converted into layout as shown in figure 8-36. The *dreg* primitive is converted into a complex structure including the master and slave memories, each with two inverters and two pass-transistors, as well as one pass transistor for the Reset function.

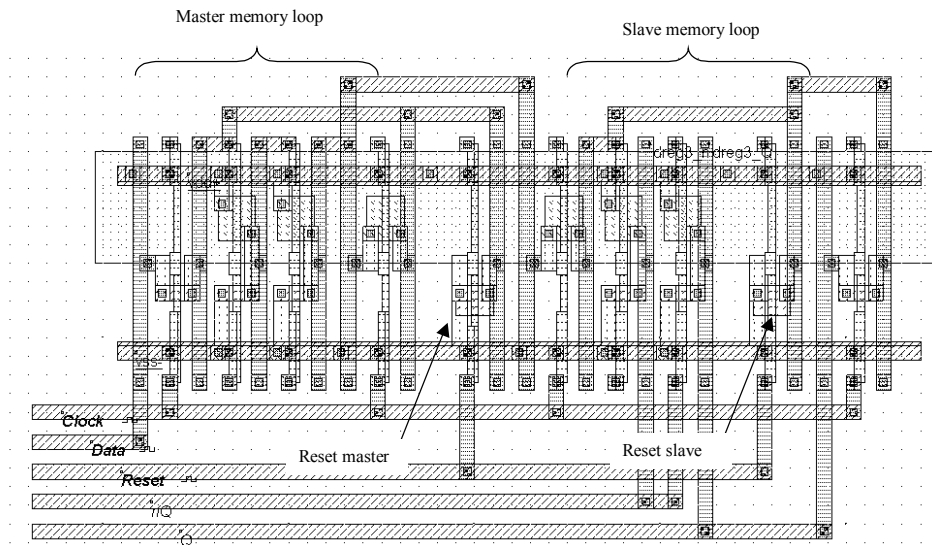


Figure 8-36 Compiling the Dreg with Microwind (DregCompile.MSK)

For testing the Dreg, the *Reset* signal is activated twice, at the beginning and later, using a piece-wise linear property. The *Clock* signal has a 2ns period. The data *D* is not synchronized with *Clock*, in order to observe various behaviors of the register.

The simulation of the edge-triggered D-register is reported in figure 8-37. 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.

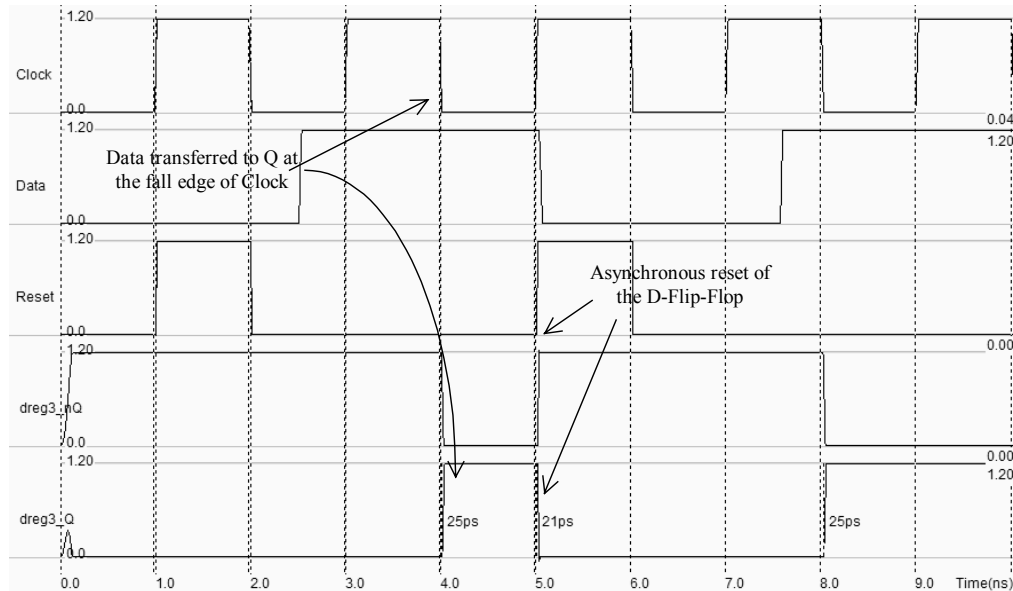


Figure 8-37 Simulation of the DREG cell (DregCompile.MSK)

Improved Dreg Design

In cell libraries, most latch designs are based on transmission gates rather than single pass transistors. The single pass transistor approach leads to more compact designs, dissipating less power. However, the voltage amplitude is degraded, so the noise margin is reduced, and the switching speed is slower. The *Dreg* cell implemented with transmission gates requires one inverter to generate the *nClock* signal (Figure 8-38). Notice the NAND gate for the *Reset* function which is active on a low level of *nReset*.

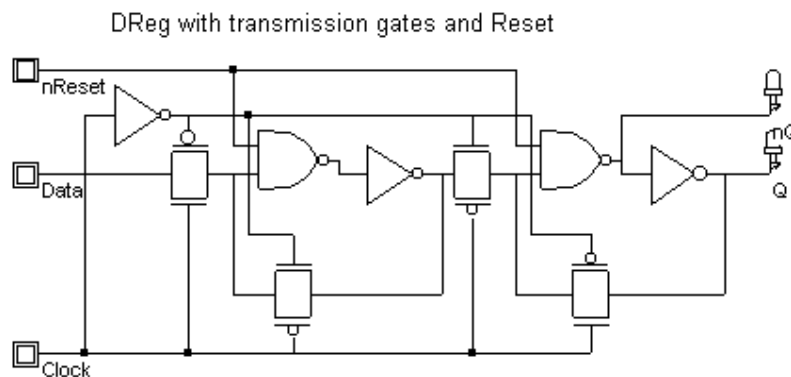


Figure 8-38 Dreg with transmission gates (DregTgate.SCH)

In the case of long distance routing between the Dreg output and its cell destination, the output node Q is rarely linked to the storage node directly (Figure 8-39). This is crucial for reliability in case of coupling noise, as introduced in chapter 5. Around one millimeter of coupled interconnects is sufficient to provoke crosstalk noise approaching the commutation point of the inverters. With a latch design with direct feedback from Q to the storage loop (Figure 8-39-a), a strong noise injected by capacitance coupling in an adjacent line may change the state of the register. If the signal Q is isolated from the storage loop (Figure 8-39-b), the noise vanishes and the memory state is not altered. A modified circuit with isolated outputs is proposed in figure 8-40. Notice the supplementary inverters that increase the propagation delay of the circuit and increase the power consumption.

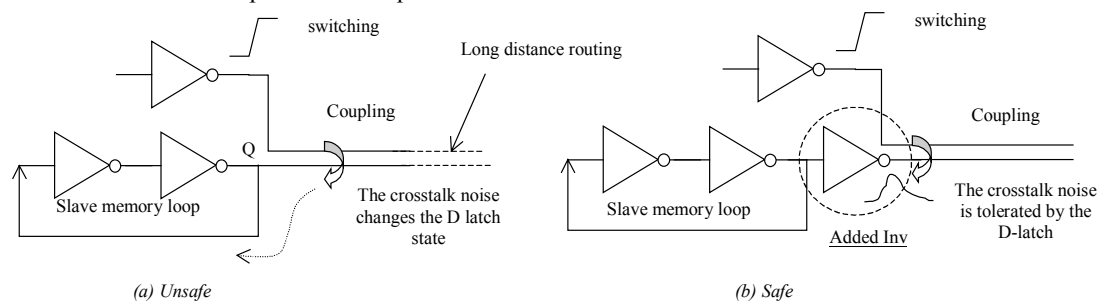


Figure 8-39 The storage loop directly connected to the output node Q may be affected by a coupling noise.

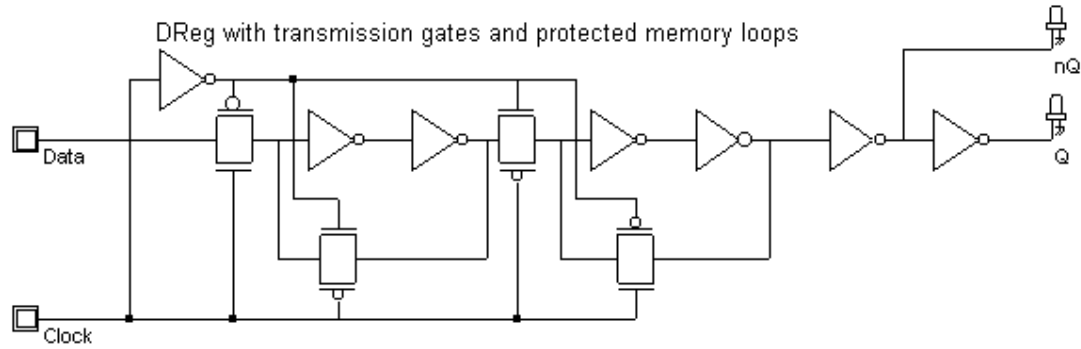


Figure 8-40 The isolated Dreg circuit (DregTgate.SCH)

5. Clock Divider

The one-bit counter is able to produce a signal featuring half the frequency of a clock. The most simple implementation consists of a *Dreg* where the output *nQ* is connected to *D*, as shown in figure 8-41. In the logic simulation, the input *clock* changes the state of *ClockDiv2* at each fall edge. The *reset* signal is active high, and sticks the output to 0.

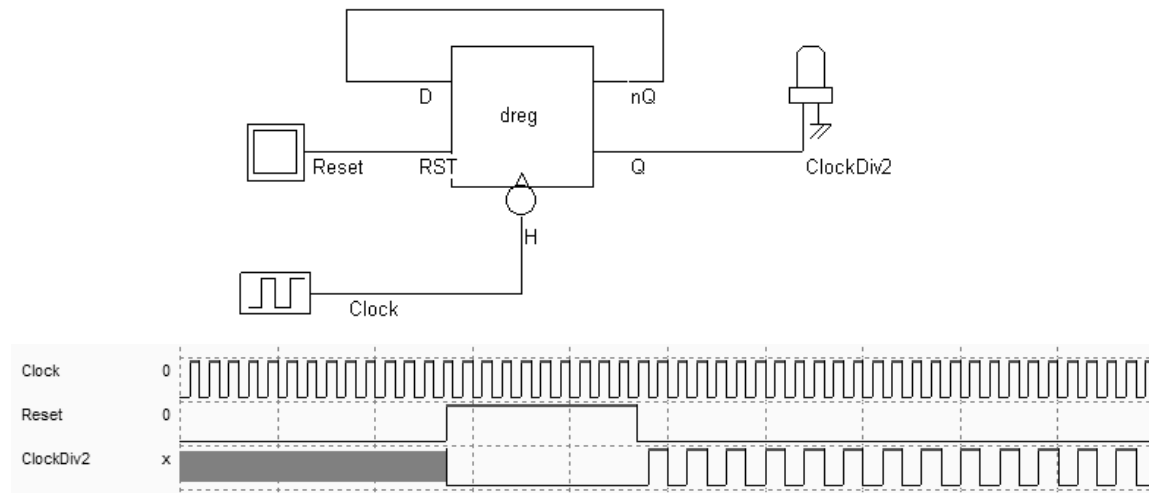


Figure 8-41 Logic simulation of the divider-by-two (ClockDiv2.SCH)

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.

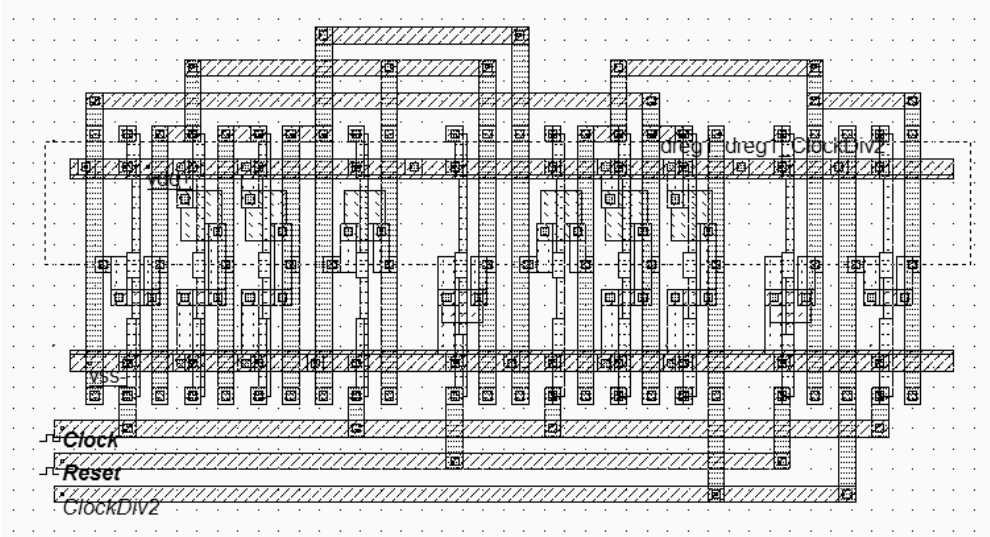


Figure 8-42 The compiled layout of the clock divider (ClockDiv2.MSK)

Firstly, the default clock assigned to the signal *Reset* should be changed into a pulse property, to avoid a cyclic reset of the divider. Secondly, the input *Clock* is reprogrammed as a piece-wise-linear where the input frequency starts from 1GHz and rises up to 10GHz.

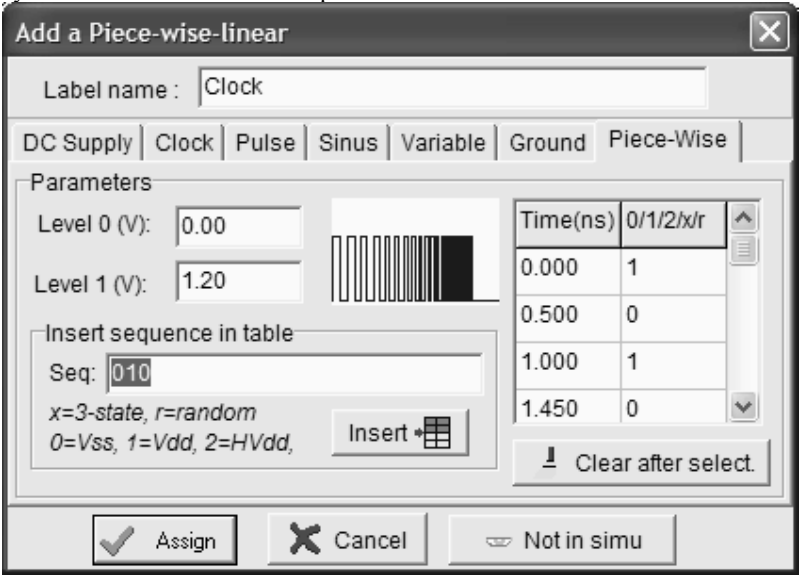


Figure 8-43 The piece-wise-linear description of the Clock with increased frequency (ClockDiv2.MSK)

The recommended simulation mode is **Frequency vs. time**. The frequency variation of the output node *ClockDiv2* appears in the upper part of figure 8-44. It can be seen that the divider circuit works correctly up to 2.5GHz, that is an input frequency of 5.0GHz.

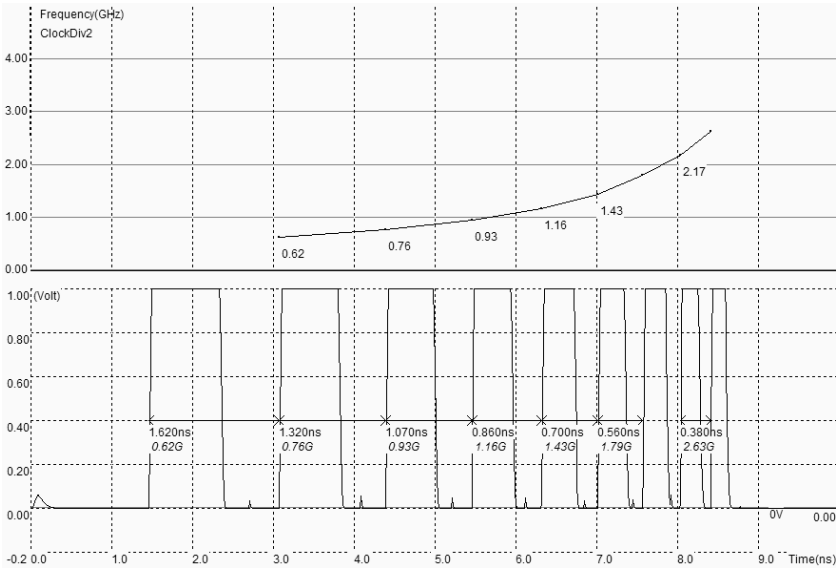


Figure 8-42 Analog simulation of the divider-by-two for the evaluation of the maximum operating frequency (ClockDiv2.MSK)

Asynchronous Counter 0..15

The *Dreg* cell connected as a one-stage counter cell may be cascaded to create a larger counter circuit. The clocking of each stage is simply carried out by the previous counter stage output, to form an asynchronous counter circuit. The 4-stage binary counter displays numbers from 0 to 15, using a chain of four *Dreg* cells, as illustrated in figure 8-43. Note the *Reset* signal common to all stages.

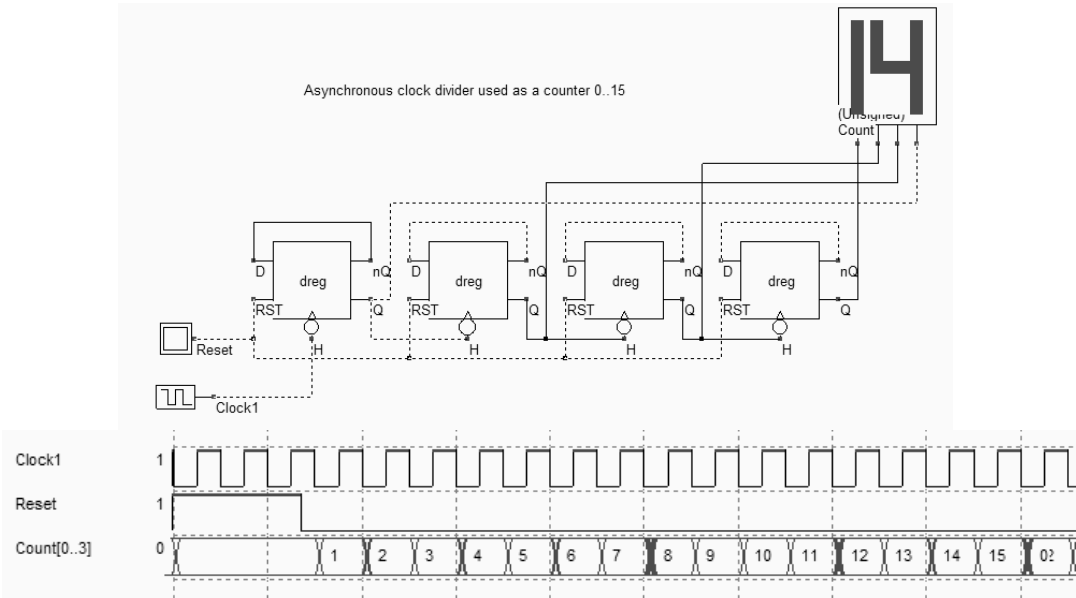


Figure 8-43. The asynchronous counter circuit principles and logic simulation (CountA16.SCH)

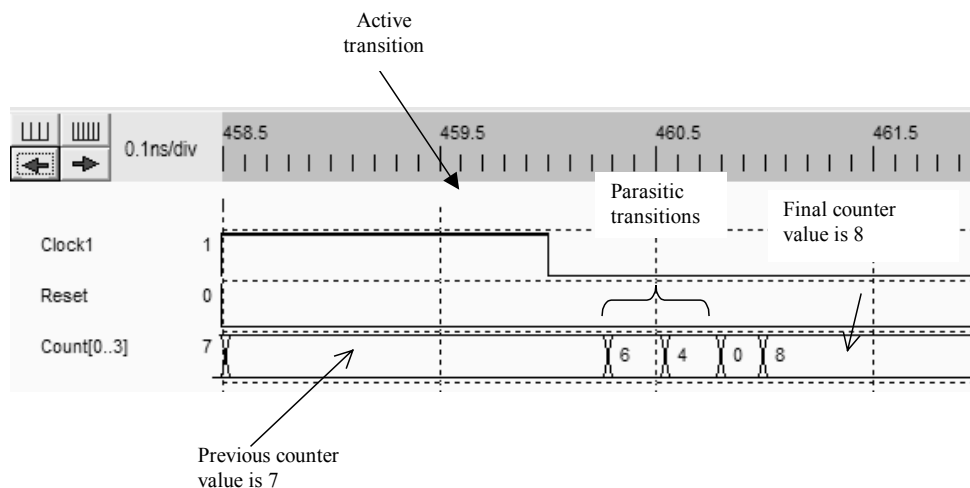


Figure 8-44 Intermediate counter values during asynchronous switching (CountAl6.SCH)

The asynchronous counter has several intermediate values during switching due to cascaded delays in the chain. Before the last stage is correctly settled, several parasitic values appear in the chronograms. In figure 8-45, a divide-by-13 counter is proposed. It uses an AND gate connected to the *Reset* control, to provoke a general reset of the *Dreg* cells once the desired number has been attained.

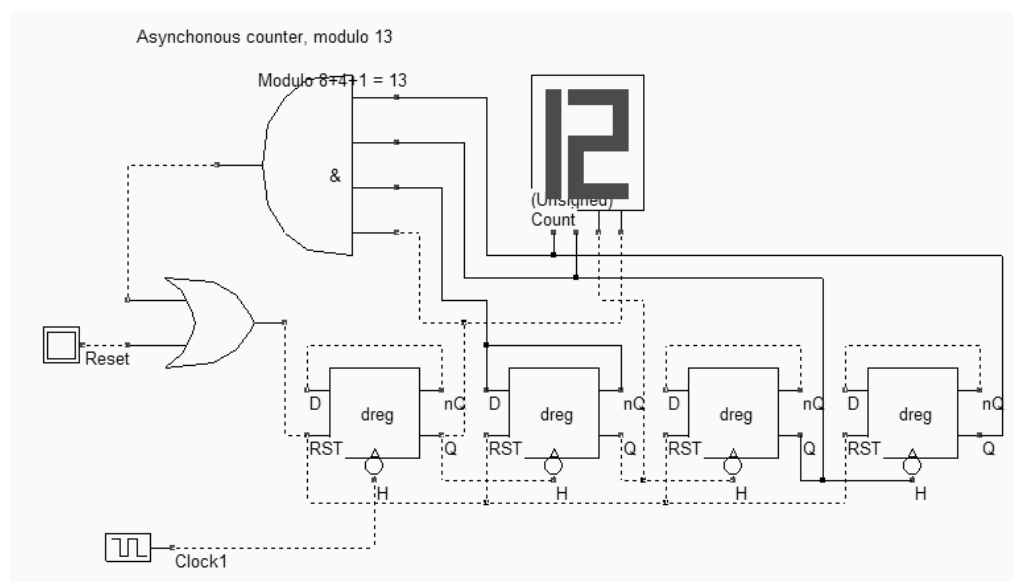


Figure 8-45 Counter 0..12 using an evaluation circuit for a forced Reset (CountA13.SCH)

An undesired *Reset* may be provoked by the AND gate in the instability time interval during which the counter chain changes its value. This is why asynchronous counters are dangerous to use, and synchronous counters are preferred.

6. Synchronous Counters

The main difference between asynchronous and synchronous counters is the clock connection. In the case of a synchronous counter, the signal *clock* is shared by all stages of the counter. The synchronous counter shown in figure 8-46 [Weste] uses one adder and one *Dreg* for each stage. The counter performs up and down counting. The *Reset* signal is shared by all *Dreg* cells.

In the simulation, we see that the counter increments the output when Up/Down is at zero. Otherwise, the outputs is decremented.

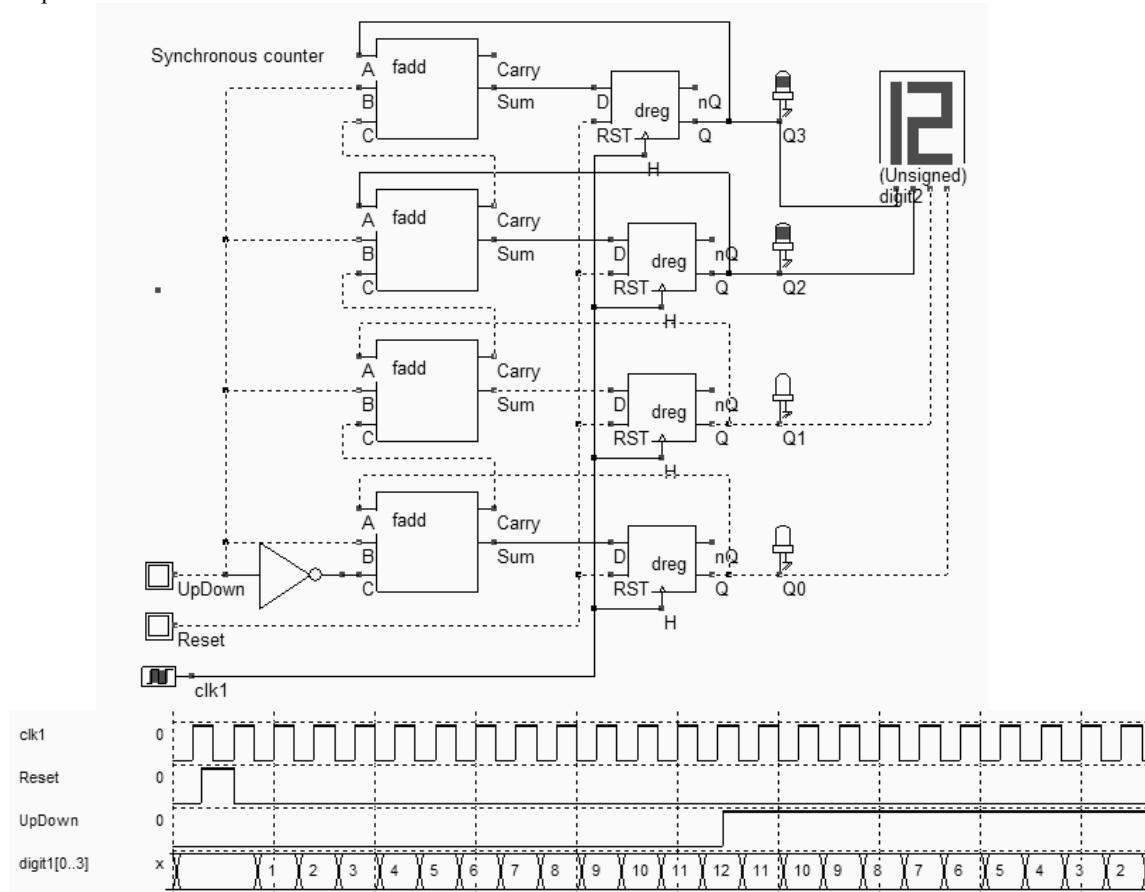


Figure 8-46 Design of a synchronous Up/Down counter using a Full adder and a Dreg (CounterUpDown.SCH)

Counter Model

The behavioral model of a counter is given in table 8-1. The model description uses the Verilog format, which is very similar to the internal format used to describe models in DSCH.

```
module countUp(clock,reset,count);
  input clock,reset;
  output [3:0] count;
  reg [3:0] count;    // declares a register type for Count

  always
  begin
```

```

    if (reset==0) // asynchronous Reset of the counter
        count = 0;
    @(negedge clock); // At a fall edge of the clock
    if (count==9)
        count = 0; // After 9, 0
    else
        count = count+1; // otherwise increment the counter
    end
endmodule

```

Table 8-1 The behavioral model of the counter

The *count* value is declared as a register, which is equivalent to a variable in a programming language. The count register is assigned immediate values such as 0, or assigned an incremented value, which requires an adder circuit. Notice that Verilog understands the addition "+", the subtraction "-", and several other operators. The value of *count* remains unchanged until a new assignment is executed. The register *count* is changed on a negative edge of the input clock.

7. Shift registers

Shift registers are used in many integrated circuits to convert serial logic data into parallel data. In many cases, serial data links are preferred for communication between blocs, as it reduces the number of pins and the size of buses with a positive impact on the interconnection cost. However, the serial link is slower than the parallel interface. Edge sensitive *Dreg* cells are very useful for converting serial data stream into parallel data stream. In Verilog, the shift operation symbol is "<<" for shift left and ">>" for shift right.

In the circuit shown figure 8-47, four *Dreg* cells are chained by connecting the Q output to the *D* input of the next stage. The serial data enters by the input *DataIn*. At each fall edge of *Clock*, the register copies the input onto the output. In the upper configuration, a '1' has propagated to the first stage. At the next active edge of the clock, it has propagated to the second stage, while a new one passes through the first stage. Then, a zero is presented on *DataIn*. At the end, the four serial information (1,1,0,0) appears in (*D0,D1,D2,D3*).

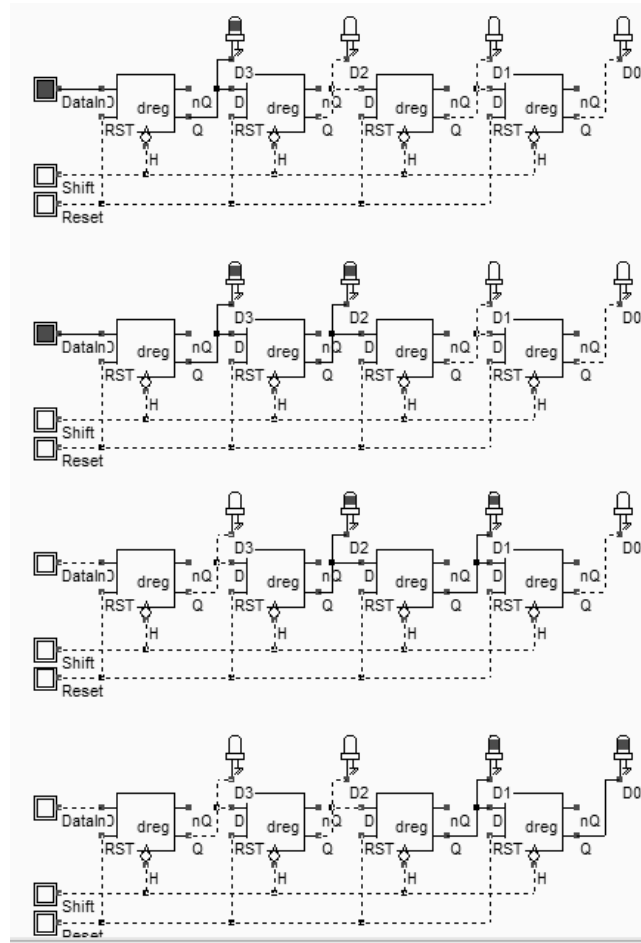


Figure 8-47 Using Dreg cells to build shift registers (ShiftReg4.SCH)

Programmable delay line

Shift registers may also be used to construct a programmable delay line. The principles of this circuit are to pass the input signal directly, or to delay the information for one period of clock through a *Dreg* cell. The decision circuit is a multiplexor, controlled by the user. The simulation confirms that the input data string *DataIn* is delayed over one, two, three or four clock periods, depending on the number of active *AddDt*.

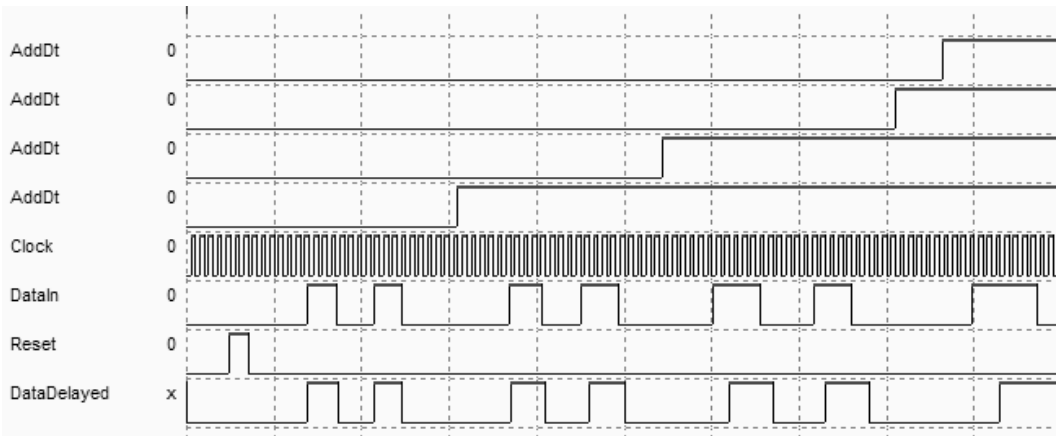
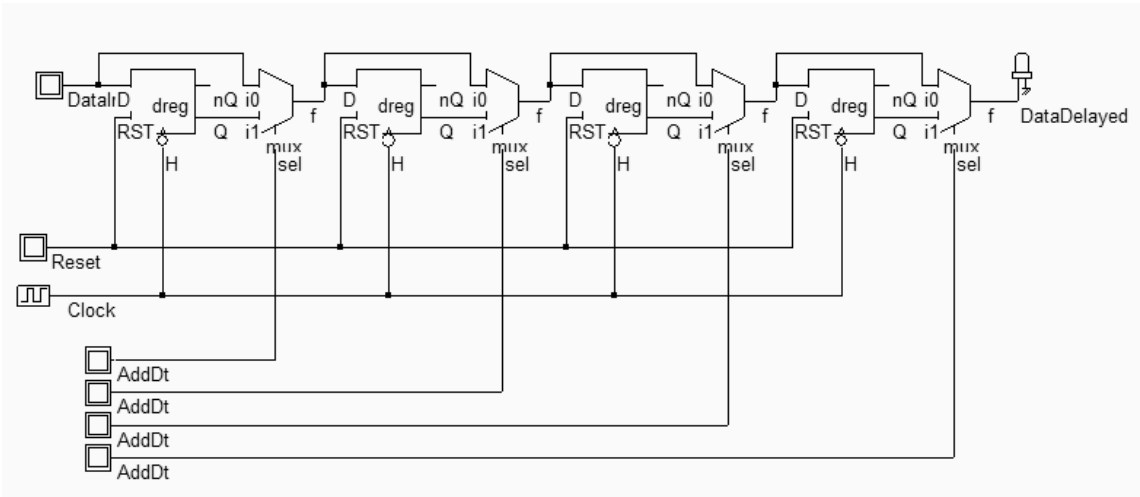


Figure 8-48 A programmable delay line using Dreg (DelayLine.SCH)

8. A 24 Hours Clock

We describe in this chapter a circuit to generate a clock that counts hours and minutes. The basic components are described in figure 8-49. The main clock is divided by 60 to create minutes, which is again divided by 60 to create hours. Several counters are needed for this circuit: a counter up to 10 and 6 for minutes, and a counter up to 24 for hours.

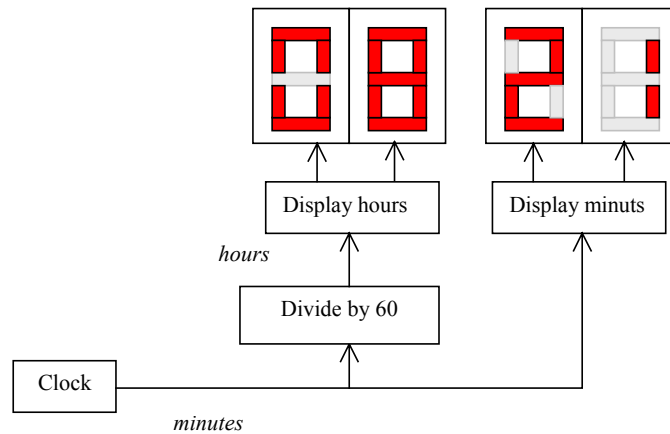


Figure 8.49 Principles of a 24 Hours clock circuit

Basic Cell

A possible implementation is based on synchronous counters. In that case, the basic element, called *T-latch*, has two ways of resetting the output Q . The synchronous reset (\sim Sclear) is effective on a fall edge of the clock, while the asynchronous *Reset* assigns a zero to Q independently of the clock. The synchronous counter makes an extensive use of the synchronous *Reset*, while the asynchronous *Reset* is kept for initializing the whole hardware, and for starting with a determined state.

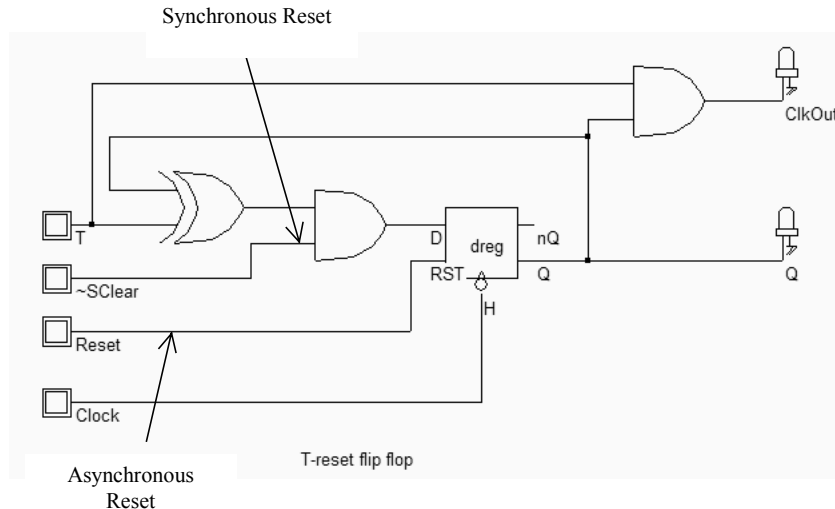


Figure 8-50 The T-Flip flop used for synchronous counters (ClkBascT.SCH)

The T-Flip flop architecture is derived from the synchronous counter of figure 8-46. The full-adder circuit is replaced by a half adder as the decrement function is useless. The XOR and AND cells of the half adder circuit may be identified in the T-flip flop design.

Synchronous Counter

Consider the counter design shown in figure 8-51. Four T-Flip-Flop circuits have been cascaded to create a synchronous counter up to 15. The problem is to build an interrupting circuit which resets the registers once the number 9 is attained. This is done with a NAND gate situated on the right lower side of the schematic diagram. The NAND output is connected to the synchronous *Clear* of the latches. When asserted, it should be held during one complete period of clock before the circuit is reset. The circuit starts to count if Enable is active (High level) and if the asynchronous Reset is inactive (Low level). The synchronous clear $\sim\text{Clear}$ should also be high. In that condition, the circuits count from 0 to 9 and then reset. Notice in the chronograms of figure 8-52 the aspect of *Sup9* which corresponds to a clean pulse during one period of clock. This signal will be used to control the next stages of the counters.

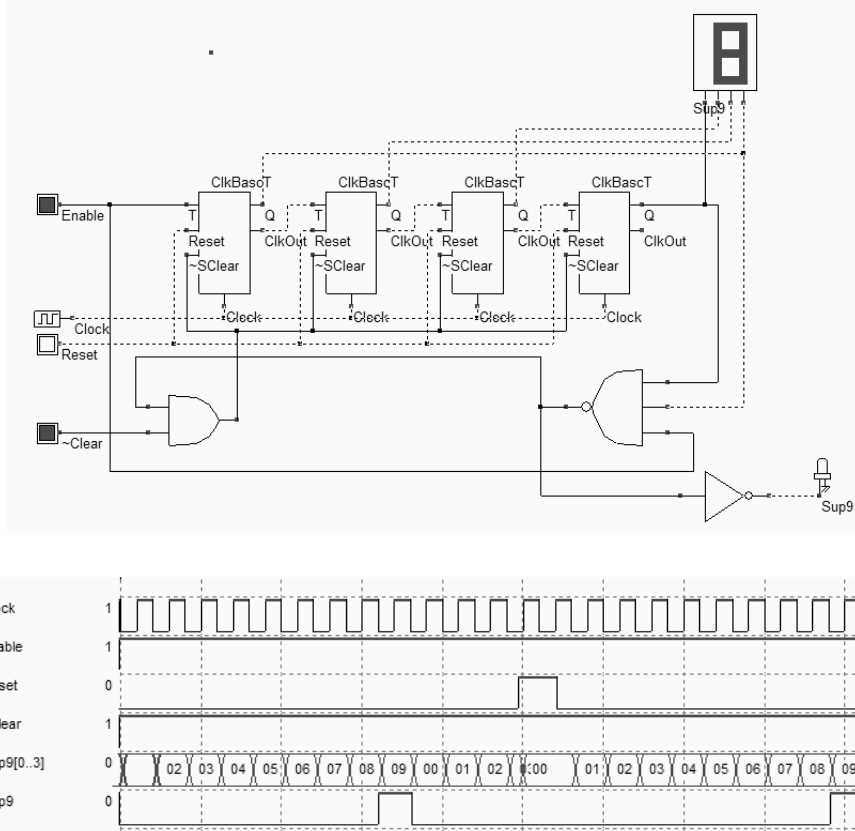


Figure 8-51 Divider by 10 (ClkDiv 10.SCH)

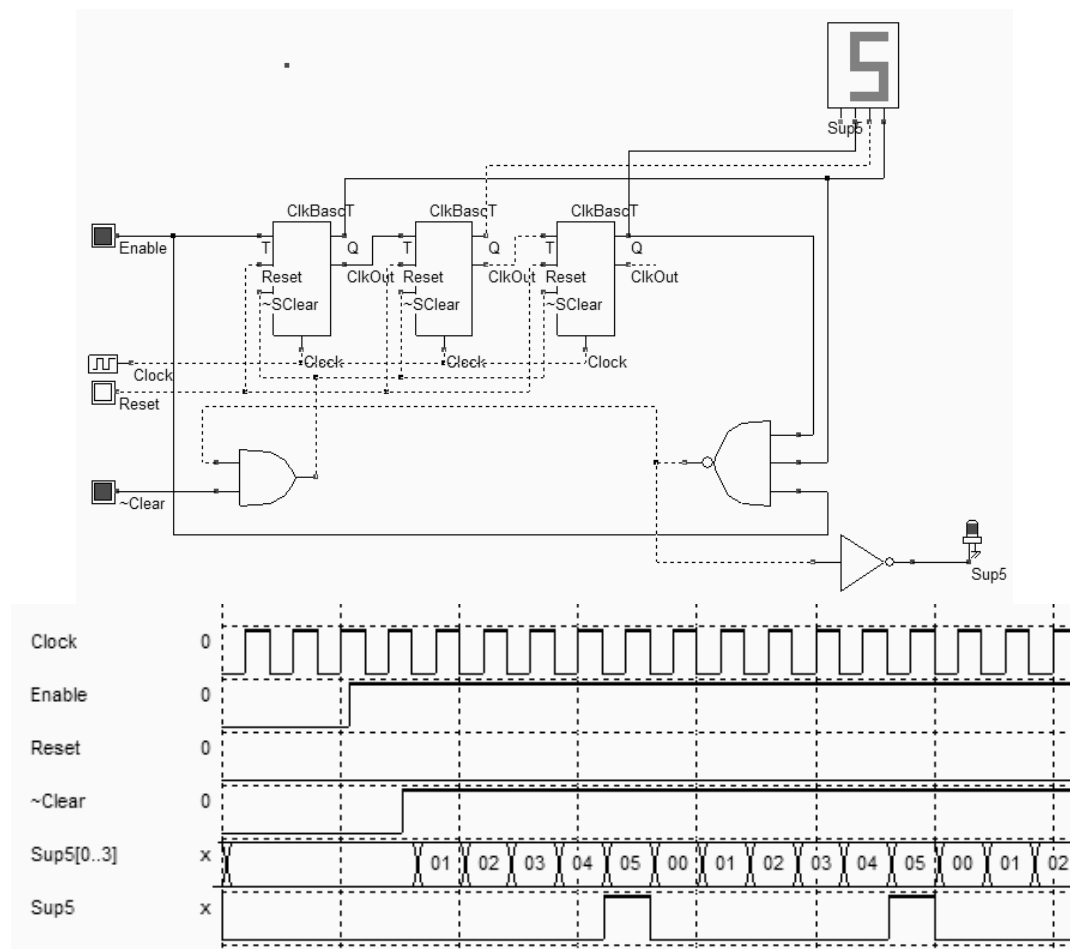


Figure 8-52 Divider by 6 (ClkDiv_6.SCH)

The synchronous counter shown in figure 8-52 is used to count from 0 to 5. The structure is very similar to the counter from 0 to 9, except that only three T-flip flop circuits are needed. We combine the two counters to create the minute counter from 0 to 59. The most important connection is between the output *Sup9* and the Enable input of *Clk_Div6*. When *Sup9* is asserted, the counter from 0 to 5 is activated, and the next clock edge will increase the slave counter. Otherwise, only the master counter from 0 to 9 is working.

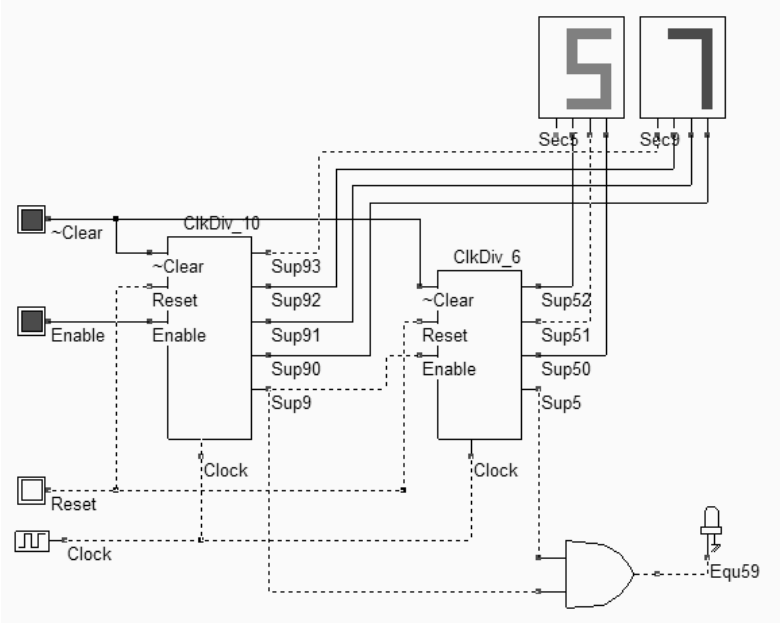


Figure 8-53 The minute counter (ClkDiv_60.SCH)

The circuit for counting hours reuses the circuits *ClkDiv_10* and needs a new circuit *ClkDiv_3*. The signal *Equ23* is generated when the number 3 appears in the master counter [0..9] and the number 2 appears in the slave counter [0..2] (Figure 8-54).

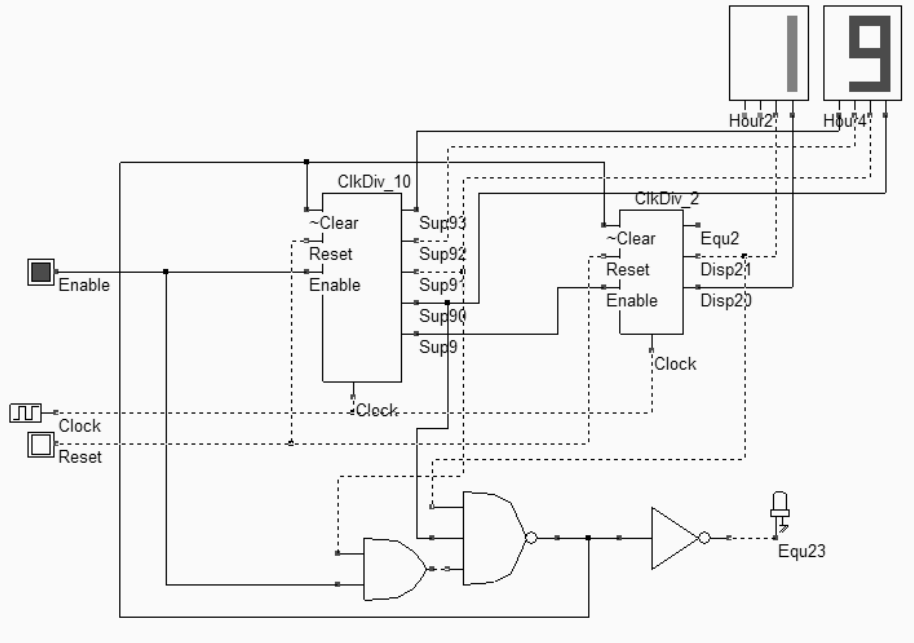


Figure 8-54 The hour counter (ClkDiv_24.SCH)

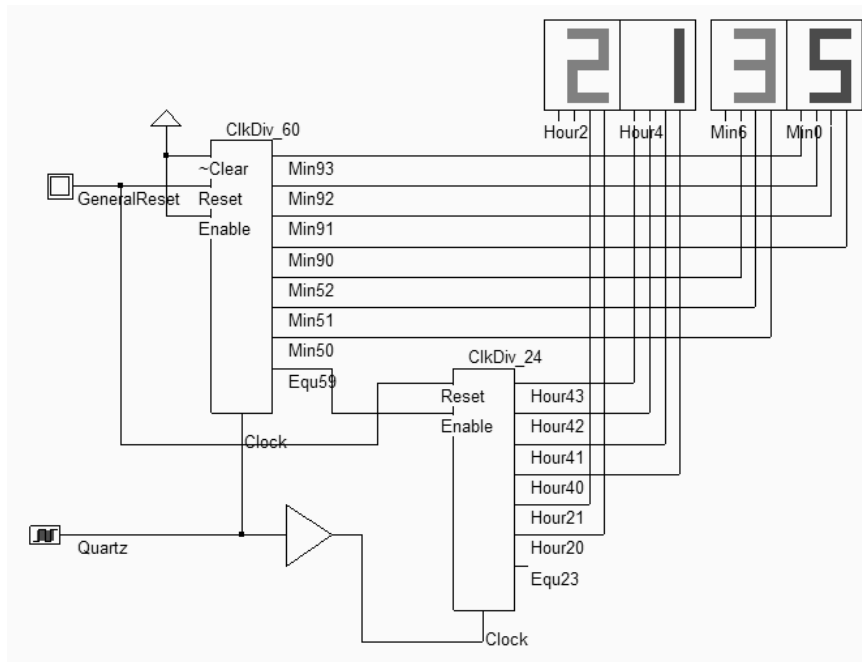


Figure 8-55 The complete schematic diagram of the 24H00 clock counter (Clk24H00.SCH)

The final schematic diagram of the clock is shown in figure 8-55. The circuit works properly up to a maximum frequency. If we increase the frequency of the main clock, the clock resets erratically. This is due to the cumulated delay, which is the sum of all delay stages. Also notice the buffer on the clock signal, used to delay the fall edge of the clock on the *ClkDiv_24* circuit, as the enable signal is issued from the *ClkDiv_60* circuit through *Equ59* with a significant delay.

9. Conclusion

In this chapter, we have introduced the elementary latch based on two inverters, and the RS latch with its NAND and NOR implementation. The D-latch has also been detailed, and the timing performances have been analyzed at layout level. The edge-triggered register was described from a behavioral and structural point of view. Several versions of this register have been reviewed, and some main applications have been illustrated: the clock divider, the asynchronous counter, the shift register and the programmable delay line. At the end of this chapter, synchronous counter circuits have been presented, with an application to a complete 24 hours clock.

References

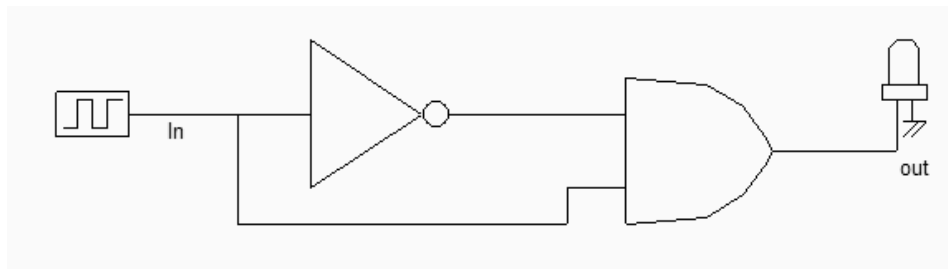
[xxx] V. Stojanovic, V. Oklobdzija "Comparative Analysis of Master Slave Latch & Flip Flops for High Performance and Low Power Systems", IEEE Journal of Solid State Circuits, Vol 3, April 1999, pp 536-548

[Weste] <td>

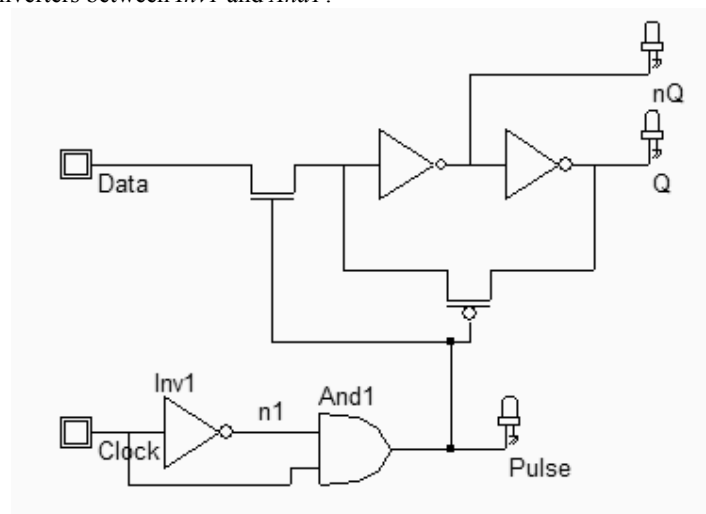
[Uyemura] <tb>

EXERCISES

- 8.1 Compare the current consumption and switching performances of two versions of D-latch: the complex gate implementation and the AND/NOR implementation.
- 8.2 Using DSCH, realize the following schematic diagram, and simulate its behaviour with a clock at the input *In*. What is the functionality of that circuit?

*Figure 8-56 Circuit to be analyzed*

- 8.3 How does this latch work? Implement this schematic diagram using Microwind. What is the effect of :
- Degrading the strength of inverter *Inv1*?
 - Adding a physical capacitor on node *n1*?
 - Adding more inverters between *Inv1* and *And1*?

*Figure 8-57 Latch to be analyzed*