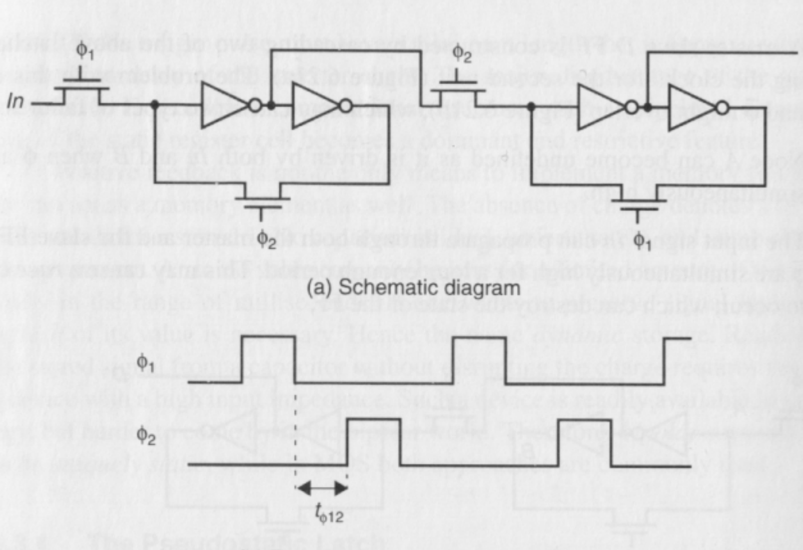


(a) Schematic diagram

(b) Two-phase nonoverlapping clocks

Figure 6.22 Pseudostatic two-phase *D* FF.

6.3.2 The Dynamic Two-Phase Flip-Flop

We can reduce the complexity of the flip-flop even more by using a fully dynamic approach, as shown in the master-slave *D* FF of Figure 6.23. The gate presented requires only six transistors, but the state has to be refreshed at periodic intervals to prevent a loss due to charge leakage. This circuit is often used in pipelined datapaths and register files for micro- or signal processors. In those modules, data storage in the latches is overwritten on a periodic base anyway, and an explicit refresh is not necessary.

A disadvantage of the circuit is that correct operation requires the availability of two nonoverlapping clocks, or four if complementary transmission gates are used. Similar to

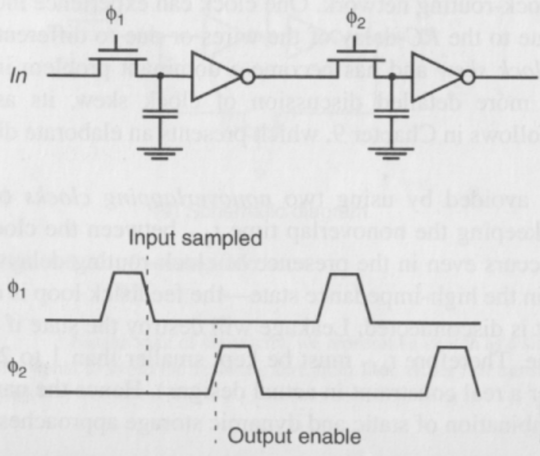


Figure 6.23 Dynamic master-slave *D* FF and corresponding timing diagram.

the pseudostatic gate, the dynamic FF is sensitive to overlap between the ϕ_1 and ϕ_2 cloc If the clocks are simultaneously high for long enough, both switches are on simul neously, which creates a race condition. It is therefore important to ensure that the no overlapping condition is valid over the complete chip by making $t_{\phi12}$ sufficiently larg Unfortunately, this has a negative impact on the circuit performance, as will be demo strated in Chapter 9. Furthermore, distributing a pair (or two pairs) of nonoverlappi clocks over a large die and ensuring that they remain nonoverlapping is cumbersome a counterproductive. A number of approaches to circumvent this constraint have therefo been devised, as will become apparent in the following sections.

Problem 6.3 Dynamic Master-Slave *D* FF

Given the dynamic master-slave *D* FF of Figure 6.23, determine the values of the set-up ti as well as the propagation delay of the register.

6.3.3 The C²MOS Latch

The pseudodynamic or dynamic *D* FFs malfunction when the clocks (ϕ-$\overline{\phi}$ or ϕ_1-ϕ_2) over for a length of time. Figure 6.24 shows an ingenious *D* FF, which is insensitive to overl This circuit is called the *C²MOS register* [Suzuki73].

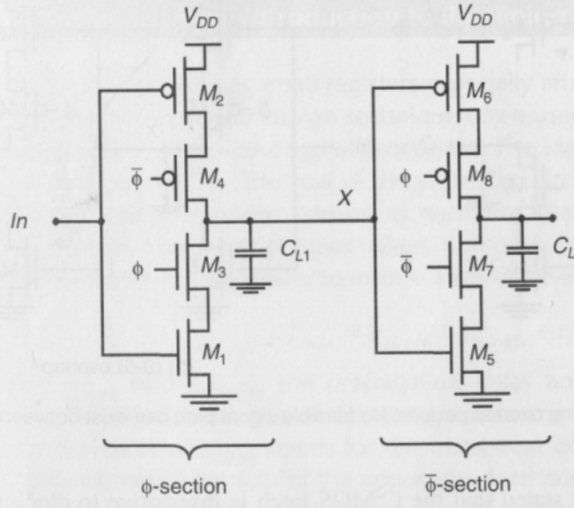


Figure 6.24 C²MOS master-slave flip-flop.

The register operates in two phases.

1. $\phi = 1$ ($\overline{\phi} = 0$): The ϕ-section acts as an inverter, because M_3 and M_4 are on. It is to be in the *evaluation mode*. Meanwhile, the $\overline{\phi}$ section is in a high-impeda mode, or in a *hold mode*. Both transistors M_7 and M_8 are off, decoupling the ou from the input. The output *D* retains its previous value stored on the output capac C_{L2}.

2. The roles are reversed during $\phi = 0$: The first section is in hold mode (M_3-M_4 off), while the second section evaluates (M_7-M_8 on). The value stored on C_{L1} propagates to the output node.

The overall circuit operates as a negative edge-triggered master-slave *D* FF. During $\phi = 1$, the input is sampled and stored on C_{L1}. When $\phi = 0$, this value is transferred to C_{L2} and appears at the output of the gate. Most important is the following observation:

> A C^{2}MOS register with (ϕ-$\bar{\phi}$) clocking is insensitive to overlap, as long as the rise and fall times of the clock edges are sufficiently small.

In order to establish the truth of this statement, we have to examine both the (0-0) and (1-1) overlap cases. In the (1-1) overlap case, the circuit simplifies to the network shown in Figure 6.25a. Inspection of this circuit reveals that a race through this network is simply not possible. An input signal cannot propagate to the output, since only the pull-down networks are enabled. As the circuit consists of a cascade of inverters, signal propagation requires one pull-up followed by a pull-down, or vice-versa, which is not feasible in the situation presented. This effectively breaks any cycle that might exist in the circuit. A similar situation occurs in the case of a (0-0) overlap (Figure 6.25b). Here, only the pull-up networks are enabled, and signal propagation from input to output is impossible.

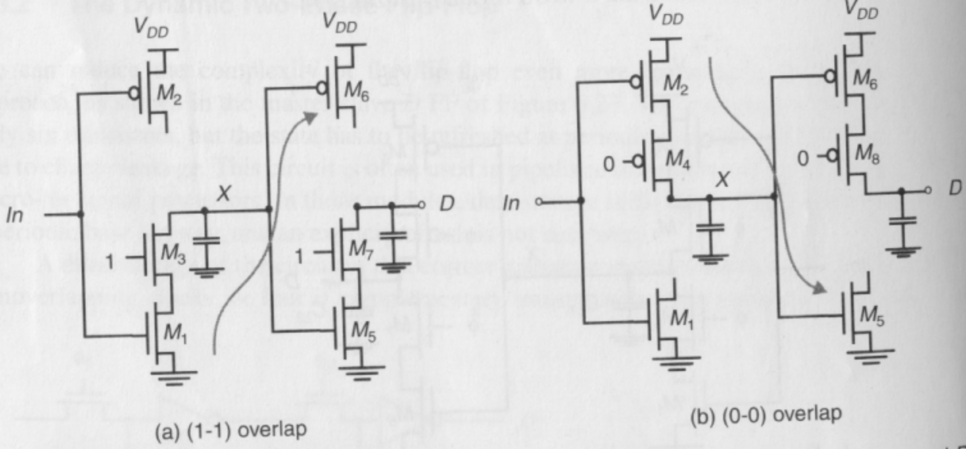


Figure 6.25 C^{2}MOS *D* FF during overlap periods. No feasible signal path can exist between *In* and *D*, as illustrated by the arrows.

In summary, it can be stated that the C^{2}MOS latch is insensitive to clock overlaps because those overlaps activate either the pull-up or the pull-down networks of the latches, but never both of them simultaneously. If the *rise and fall times of the clock* ($t_{r\phi}$, $t_{f\phi}$) are sufficiently slow, however, there exists a time slot where both the NMOS and PMOS transistors are conducting. This creates a path between input and output that can destroy the state of the circuit. Simulations have shown that the circuit operates correctly as long as the clock rise time $t_{r\phi}$ (or fall time $t_{f\phi}$) is smaller than approximately five times the propagation delay of the latch (t_{pC2MOS}). This criterion is not too stringent and is easily met in practical designs. The impact of the rise and fall times is illustrated in Figure 6.26, which

plots the simulated transient response of a C^{2}MOS *D* FF for clock slopes of respecti 0.1 and 3 nsec (for an initial propagation delay of approximately 0.3 nsec). For s clocks, the potential for a race condition exists.

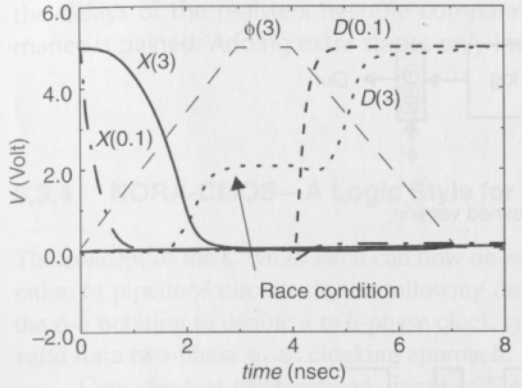


Figure 6.26 Transient response c C^{2}MOS FF for 0.1 nsec and 3 nsec clock rise (fall) times assuming *In* =

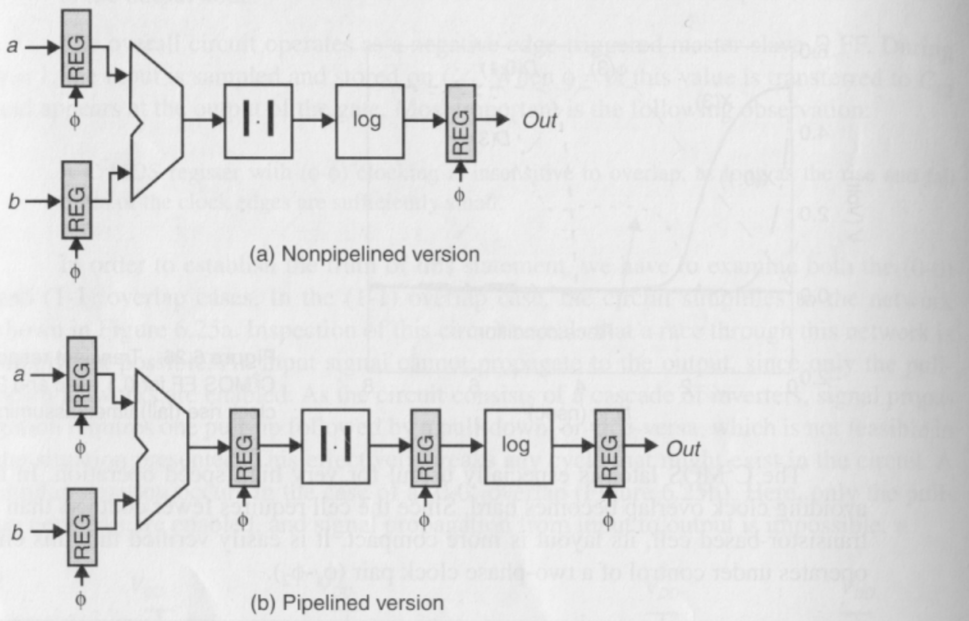
The C^{2}MOS latch is especially useful for very high speed operation. In that c avoiding clock overlap becomes hard. Since the cell requires fewer contacts than the p transistor-based cell, its layout is more compact. It is easily verified that this circuit operates under control of a two-phase clock pair (ϕ_1-ϕ_2).

Sideline: A Brief Introduction to Pipelining

The need for fast, yet small registers especially arises in pipelined datapath structu *Pipelining* is a popular design technique often used to accelerate the operation of datapaths of micro- and signal processors. The idea is easily explained with the ex ple of Figure 6.27a. The goal of the presented circuit is to compute log(|*a* – *b*|), wh both *a* and *b* represent streams of numbers, that is, the computation must be formed on a large set of input values. Using Eq. (6.1), we can determine the min clock period T_{min} necessary to ensure a correct evaluation of the results.

$$T_{min} = t_{p,reg} + t_{p,logic} + t_{setup,reg} \quad ($$

with $t_{p,reg}$ and $t_{setup,reg}$ the propagation delay and the setup times of the regis respectively. For the sake of simplicity, we assume that the registers are edge-trigg *D* FFs. The term $t_{p,logic}$ stands for the worst-case delay path through the combinat network, which consists of the adder, absolute value, and logarithm functions. The ter delay is generally much larger than the delays associated with the registers dominates the circuit performance. *Pipelining* is a means to break this performa bottleneck. Assume that we introduce registers between the logic blocks, as show Figure 6.27b. This causes the computation for one set of input data to spread ov number of clock periods, as shown in Table 6.1. The result for the data set (a_1, b_1) appears at the output after three clock-periods. At that time, the circuit has alre performed parts of the computations for the next data sets, (a_2, b_2) and (a_3, b_3). The computation is performed in an assembly-line fashion, hence the name pipeline.



(a) Nonpipelined version

(b) Pipelined version

Figure 6.27 Datapath for the computation of log(|a + b|).

Table 6.1 Example of pipelined computations.

Clock Period	Adder	Absolute Value	Logarithm
1	$a_1 + b_1$		
2	$a_2 + b_2$	$\|a_1 + b_1\|$	
3	$a_3 + b_3$	$\|a_2 + b_2\|$	$\log(\|a_1 + b_1\|)$
4	$a_4 + b_4$	$\|a_3 + b_3\|$	$\log(\|a_2 + b_2\|)$
5	$a_5 + b_5$	$\|a_4 + b_4\|$	$\log(\|a_3 + b_3\|)$

The advantage of the pipelined operation becomes apparent when examining the maximum clock speed (or the minimum clock period) of the modified circuit. The combinational circuit block has been partitioned into three sections, each of which has a smaller propagation delay than the original function. This effectively reduces the value of the minimum allowable clock period:

$$T_{min,\,pipe} = t_{p,\,reg} + max(t_{p,\,add}, t_{p,\,abs}, t_{p,\,\log}) + t_{setup,\,reg} \qquad (6.6)$$

Suppose that all logic blocks have approximately the same propagation delay, and that the latch delays are still ignorable with respect to the logic delays. The pipelined network outperforms the original circuit by a factor of three under these assump-

tions, or $T_{pipe,min} = T_{min}/3$. The increased performance comes at the relatively small
of two additional registers and an increased latency.[2] This explains why pipelinin
popular in the implementation of very high performance datapaths. Be aware, h
ever, that adding extra pipeline stages only makes sense up to a certain point. W
the delays of the registers become comparable to the logic delays, no extra per
mance is gained. Adding extra stages only increases the hardware overhead.

6.3.4 NORA-CMOS—A Logic Style for Pipelined Structures

The concept of the C^{2}MOS latch can now be extended to support the effective implen
tation of pipelined circuits. In the following discussion, we use without loss of genera
the ϕ-$\overline{\phi}$ notation to denote a two-phase clock system. All the results presented are equ
valid for a two-phase ϕ_1-ϕ_2 clocking approach.

Consider first the pipelined circuit of Figure 6.28. The pipeline registers are im
mented using pass-transistor-based D latches. When the clocks ϕ and $\overline{\phi}$ are nonover
ping, correct pipeline operation is obtained. Input data is sampled on C_1 at the nega

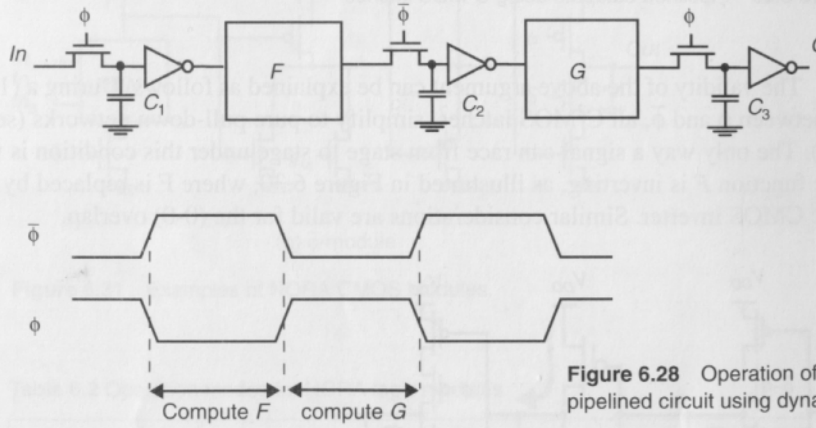


Figure 6.28 Operation of two-phase
pipelined circuit using dynamic registe

edge of ϕ; the result of the first logic block is stored on C_2 when $\overline{\phi}$ goes low, and so on.
nonoverlapping of the clocks ensures correct operation. When the clocks do overlap, h
ever, a race condition can occur. Under correct conditions, the value stored on C_2 is
result of passing the previous input (stored on the falling edge of ϕ on C_1) through
logic function F. When ϕ overlaps $\overline{\phi}$, the next input is already being applied to F, an
effect might propagate to C_2 before $\overline{\phi}$ goes low. In other words, a *race* develops betv
the previous input and the current one. Which value wins depends upon the logic func

[2] Latency is defined here as the number of clock cycles it takes for the data to propagate from the in
the output. For the example at hand, pipelining increases the latency from 1 to 3. An increased latency is in
eral acceptable, but can cause a global performance degradation if not treated with care.