CHAPTER

7

DESIGNING SEQUENTIAL LOGIC CIRCUITS

Implementation techniques for flip-flops, latches, oscillators, pulse generators, and Schmitt triggers

n

Static versus dynamic realization

n

Choosing clocking strategies

7.1 Introduction

Combinational logic circuits that were described earlier have the property that the output of a logic block is only a function of the *current* input values, assuming that enough time has elapsed for the logic gates to settle. Yet virtually all useful systems require storage of state information, leading to another class of circuits called *sequential logic* circuits. In these circuits, the output not only depends upon the *current* values of the inputs, but also upon *preceding* input values. In other words, a sequential circuit remembers some of the past history of the system—it has memory.

Figure 7.1 shows a block diagram of a generic *finite state machine* (FSM) that consists of combinational logic and registers that hold the system state. The system depicted here belongs to the class of *synchronous* sequential systems, in which all registers are under control of a single global clock. The outputs of the FSM are a function of the current *Inputs* and the *Current State*. The *Next State* is determined based on the *Current State* and the current *Inputs* and is fed to the inputs of registers. On the rising edge of the clock, the *Next State* bits are copied to the outputs of the registers (after some *propagation delay*), and a new cycle begins. The register then ignores changes in the input signals until the next rising edge. In general, registers can be *positive edge-triggered* (where the input data is copied on the positive edge) or *negative edge-triggered* (where the input data is copied on the negative edge of the clock, as is indicated by a small circle at the clock input).

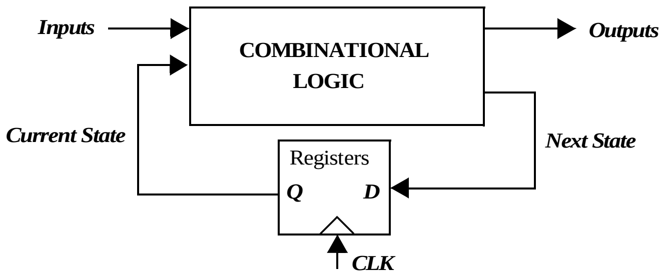


Figure 7.1 Block diagram of a finite state machine using *positive edge-triggered* registers.

This chapter discusses the CMOS implementation of the most important sequential building blocks. A variety of choices in sequential primitives and clocking methodologies exist; making the correct selection is getting increasingly important in modern digital circuits, and can have a great impact on performance, power, and/or design complexity. Before embarking on a detailed discussion on the various design options, a revision of the design metrics, and a classification of the sequential elements is necessary.

7.2 Timing Metrics for Sequential Circuits

There are three important timing parameters associated with a register as illustrated in Figure 7.2. The *set-up time* (t_{su}) is the time that the data inputs (*D* input) must be valid before the clock transition (this is, the 0 to 1 transition for a *positive edge-triggered* register). The *hold time* (t_{hold}) is the time the data input must remain valid after the clock edge. Assum-

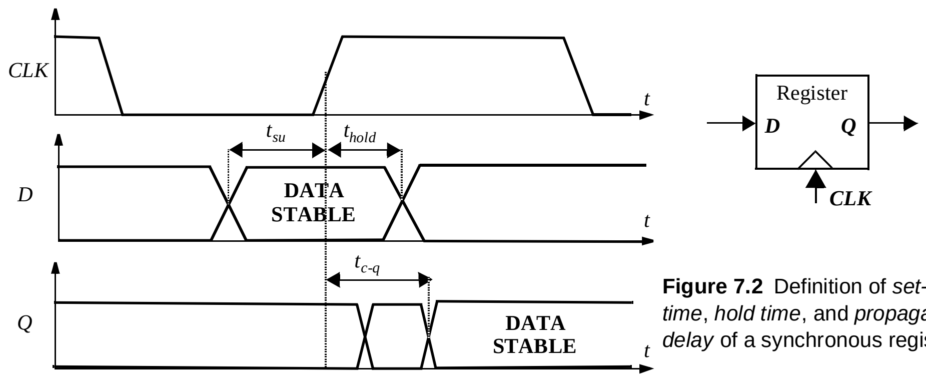


Figure 7.2 Definition of *set-up time*, *hold time*, and *propagation delay* of a synchronous register.

ing that the *set-up* and *hold*-times are met, the data at the *D* input is copied to the *Q* output after a worst-case *propagation delay* (with reference to the clock edge) denoted by $t_{c\text{-}q}$.

Given the timing information for the registers and the combination logic, some system-level timing constraints can be derived. Assume that the worst-case *propagation delay* of the logic equals t_{plogic}, while its minimum delay (also called the *contamination delay*) is t_{cd}. The minimum clock period *T*, required for proper operation of the sequential circuit is given by

$$T \geq t_{c\text{-}q} + t_{plogic} + t_{su} \tag{7.1}$$

The *hold time* of the register imposes an extra constraint for proper operation,

$$t_{cd\text{register}} + t_{cd\text{logic}} \geq t_{hold} \tag{7.2}$$

where $t_{cd\text{register}}$ is the minimum *propagation delay* (or *contamination delay*) of the register.

As seen from Eq. (7.1), it is important to minimize the values of the timing parameters associated with the register, as these directly affect the rate at which a sequential circuit can be clocked. In fact, modern high-performance systems are characterized by a very-low logic depth, and the register *propagation delay* and *set-up* times account for a significant portion of the clock period. For example, the DEC Alpha EV6 microprocessor [Gieseke97] has a maximum logic depth of 12 gates, and the register overhead stands for approximately 15% of the clock period. In general, the requirement of Eq. (7.2) is not hard to meet, although it becomes an issue when there is little or no logic between registers, (or when the clocks at different registers are somewhat out of phase due to clock skew, as will be discussed in a later Chapter).

7.3 Classification of Memory Elements

Foreground versus Background Memory

At a high level, memory is classified into background and foreground memory. Memory that is embedded into logic is foreground memory, and is most often organized as individual registers of register banks. Large amounts of centralized memory core are referred to as background memory. Background memory, discussed later in this book, achieves higher area densities through efficient use of array structures and by trading off performance and robustness for size. In this chapter, we focus on foreground memories.

Static versus Dynamic Memory

Memories can be static or dynamic. Static memories preserve the state as long as the power is turned on. Static memories are built using *positive feedback* or regeneration, where the circuit topology consists of intentional connections between the output and the input of a combinational circuit. Static memories are most useful when the register won't be updated for extended periods of time. An example of such is configuration data, loaded at power-up time. This condition also holds for most processors that use conditional clocking (i.e., gated clocks) where the clock is turned *off* for unused modules. In that case, there are no guarantees on how frequently the registers will be clocked, and static memories are needed to preserve the state information. Memory based on positive feedback fall under the class of elements called *multivibrator circuits.* The bistable element, is its most popular representative, but other elements such as monostable and astable circuits are also frequently used.

Dynamic memories store state for a short period of time—on the order of milliseconds. They are based on the principle of temporary *charge storage* on parasitic capacitors associated with MOS devices. As with dynamic logic discussed earlier, the capacitors have to be refreshed periodically to annihilate charge leakage. Dynamic memories tend to simpler, resulting in significantly higher performance and lower power dissipation. They are most useful in datapath circuits that require high performance levels and are periodically clocked. It is possible to use dynamic circuitry even when circuits are conditionally clocked, if the state can be discarded when a module goes into idle mode.

Latches vs. Registers

A latch is an essential component in the construction of an *edge-triggered* register. It is *level-sensitive* circuit that passes the *D* input to the *Q* output when the clock signal is high. This latch is said to be in *transparent* mode. When the clock is low, the input data sampled on the falling edge of the clock is held stable at the output for the entire phase, and the latch is in *hold* mode. The inputs must be stable for a short period around the falling edge of the clock to meet *set-up* and *hold* requirements. A latch operating under the above conditions is a *positive latch*. Similarly, a *negative latch* passes the *D* input to the *Q* output when the clock signal is low. The signal waveforms for a positive and negative latch are shown in Figure 7.3. A wide variety of static and dynamic implementations exists for the realization of latches.

Contrary to *level-sensitive* latches, *edge-triggered* registers only sample the input on a clock transition — 0-to-1 for a *positive edge-triggered* register, and 1-to-0 for a *negative edge-triggered* register. They are typically built using the latch primitives of Figure 7.3. A most-often recurring configuration is the *master-slave* structure that cascades a positive and negative latch. Registers can also be constructed using one-shot generators of the clock signal ("glitch" registers), or using other specialized structures. Examples of these are shown later in this chapter.

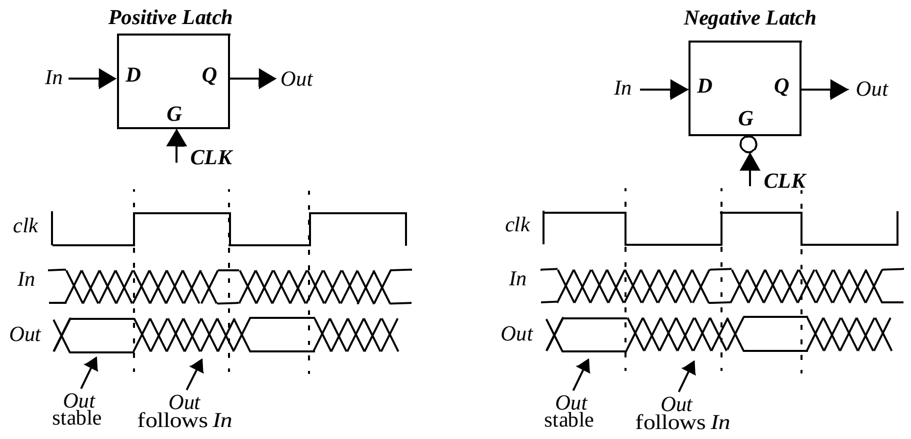


Figure 7.3 Timing of positive and negative latches.

7.4 Static Latches and Registers

7.4.1 The Bistability Principle

Static memories use positive feedback to create a *bistable circuit* — a circuit having two stable states that represent 0 and 1. The basic idea is shown in Figure 7.4a, which shows two inverters connected in cascade along with a voltage-transfer characteristic typical of such a circuit. Also plotted are the VTCs of the first inverter, that is, V_{o1} versus V_{i1}, and the second inverter (V_{o2} versus V_{o1}). The latter plot is rotated to accentuate that $V_{i2} = V_{o1}$. Assume now that the output of the second inverter V_{o2} is connected to the input of the first V_{i1}, as shown by the dotted lines in Figure 7.4a. The resulting circuit has only three possi-

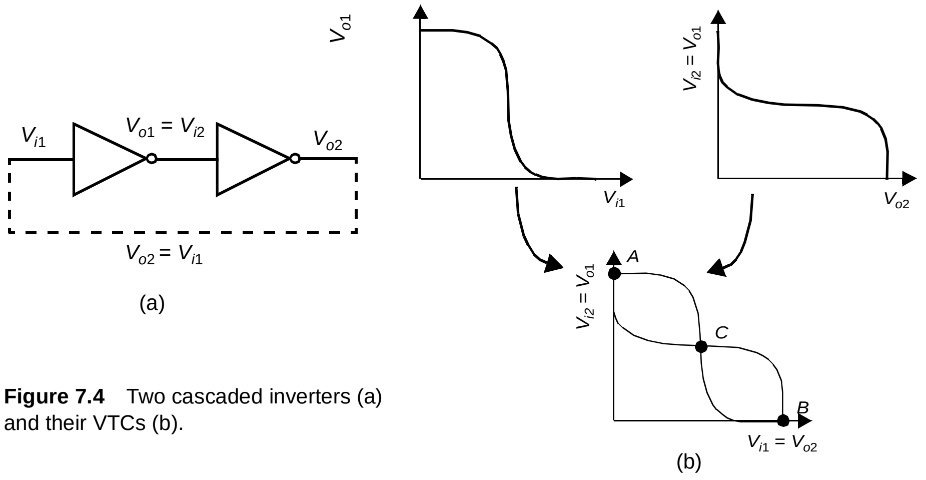


Figure 7.4 Two cascaded inverters (a) and their VTCs (b).

ble operation points (*A*, *B*, and *C*), as demonstrated on the combined VTC. The following important conjecture is easily proven to be valid:

> Under the condition that the gain of the inverter in the transient region is larger than 1, only *A* and *B* are stable operation points, and *C* is a metastable operation point.

Suppose that the cross-coupled inverter pair is biased at point *C*. A small deviation from this bias point, possibly caused by noise, is amplified and *regenerated* around the circuit loop. This is a consequence of the gain around the loop being larger than 1. The effect is demonstrated in Figure 7.5a. A small deviation δ is applied to V_{i1} (biased in *C*). This deviation is amplified by the gain of the inverter. The enlarged divergence is applied to the second inverter and amplified once more. The bias point moves away from *C* until one of the operation points *A* or *B* is reached. In conclusion, *C* is an unstable operation point. Every deviation (even the smallest one) causes the operation point to run away from its original bias. The chance is indeed very small that the cross-coupled inverter pair is biased at *C* and stays there. Operation points with this property are termed *metastable*.

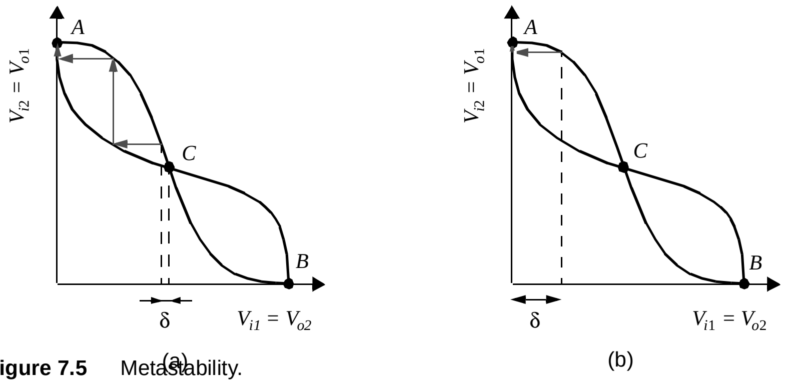


Figure 7.5 Metastability.

On the other hand, *A* and *B* are stable operation points, as demonstrated in Figure 7.5b. In these points, the **loop gain is much smaller than unity.** Even a rather large deviation from the operation point is reduced in size and disappears.

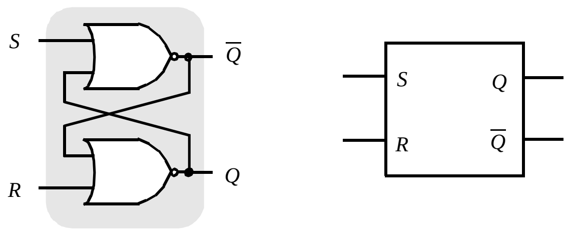
Hence the cross-coupling of two inverters results in a *bistable* circuit, that is, a circuit with two stable states, each corresponding to a logic state. The circuit serves as a memory, storing either a 1 or a 0 (corresponding to positions *A* and *B*).

In order to change the stored value, we must be able to bring the circuit from state *A* to *B* and vice-versa. Since the precondition for stability is that the loop gain *G* is smaller than unity, we can achieve this by making *A* (or *B*) temporarily unstable by increasing *G* to a value larger than 1. This is generally done by applying a trigger pulse at V_{i1} or V_{i2}. For instance, assume that the system is in position *A* ($V_{i1} = 0$, $V_{i2} = 1$). Forcing V_{i1} to 1 causes both inverters to be on simultaneously for a short time and the loop gain *G* to be larger than 1. The positive feedback regenerates the effect of the trigger pulse, and the circuit moves to the other state (*B* in this case). The width of the trigger pulse need be only a little larger than the total *propagation delay* around the circuit loop, which is twice the average *propagation delay* of the inverters.

In summary, a bistable circuit has two stable states. In absence of any triggering, the circuit remains in a single state (assuming that the power supply remains applied to the circuit), and hence remembers a value. A trigger pulse must be applied to change the state of the circuit. Another common name for a bistable circuit is *flip-flop* (unfortunately, an *edge-triggered* register is also referred to as a *flip-flop*).

7.4.2 SR Flip-Flops

The cross-coupled inverter pair shown in the previous section provides an approach to store a binary variable in a stable way. However, extra circuitry must be added to enable control of the memory states. The simplest incarnation accomplishing this is the well-know *SR —or set-reset— flip-flop*, an implementation of which is shown in Figure 7.6a. This circuit is similar to the cross-coupled inverter pair with NOR gates replacing the inverters. The second input of the NOR gates is connected to the trigger inputs (S and R), that make it possible to force the outputs Q and $\overline{Q}$ to a given state. These outputs are complimentary (except for the SR = 11 state). When both S and R are 0, the flip-flop is in a quiescent state and both outputs retain their value (a NOR gate with one of its input being 0 looks like an inverter, and the structure looks like a cross coupled inverter). If a positive (or 1) pulse is applied to the S input, the Q output is forced into the 1 state (with $\overline{Q}$ going to 0). Vice versa, a 1 pulse on R resets the flip-flop and the Q output goes to 0.



S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
1	1	0	0

Forbidden State

(a) Schematic diagram (b) Logic symbol (c) Characteristic table

Figure 7.6 NOR-based *SR* flip-flop.

These results are summarized in the *characteristic table* of the flip-flop, shown in Figure 7.6c. The characteristic table is the truth table of the gate and lists the output states as functions of all possible input conditions. When both S and R are high, both Q and $\overline{Q}$ are forced to zero. Since this does not correspond with our constraint that Q and $\overline{Q}$ must be complementary, this input mode is considered to be forbidden. An additional problem with this condition is that when the input triggers return to their zero levels, the resulting state of the latch is unpredictable and depends on whatever input is last to go low. Finally, Figure 7.6 shows the schematics symbol of the SR flip-flop.

Problem 7.1 *SR* Flip-Flop Using NAND Gates

An *SR* flip-flop can also be implemented using a cross-coupled NAND structure as shown in Figure 7.7. Derive the truth table for a such an implementation.

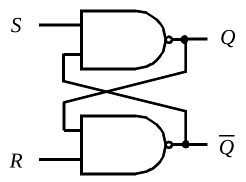


Figure 7.7 NAND-based SR flip-flop.

The SR flip-flops discussed so far are asynchronous, and do not require a clock signal. Most systems operate in a synchronous fashion with transition events referenced to a clock. One possible realization of a clocked *SR* flip-flop— a *level-sensitive* positive latch— is shown in Figure 7.8. It consists of a cross-coupled inverter pair, plus 4 extra transistors to drive the flip-flop from one state to another and to provide clocked operation. Observe that the number of transistors is identical to the implementation of Figure 7.6, but the circuit has the added feature of being clocked. The drawback of saving some transistors over a fully-complimentary CMOS implementation is that transistor sizing becomes critical in ensuring proper functionality. Consider the case where Q is high and an R pulse is applied. The combination of transistors M_4, M_7, and M_8 forms a ratioed inverter. In order to make the latch switch, we must succeed in bringing Q below the switching threshold of the inverter M_1-M_2. Once this is achieved, the positive feedback causes the flip-flop to invert states. This requirement forces us to increase the sizes of transistors M_5, M_6, M_7, and M_8.

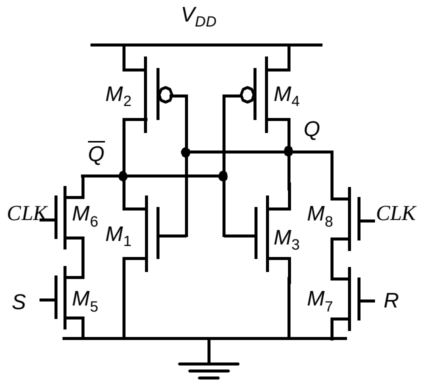


Figure 7.8 CMOS clocked *SR* flip-flop.

The presented flip-flop does not consume any static power. In steady-state, one inverter resides in the high state, while the other one is low. No static paths between V_{DD} and *GND* can exist except during switching.

Example 7.1 Transistor Sizing of Clocked SR Latch

Assume that the cross-coupled inverter pair is designed such that the inverter threshold V_M is located at $V_{DD}/2$. For a 0.25 µm CMOS technology, the following transistor sizes were selected: $(W/L)_{M1} = (W/L)_{M3} = (0.5\mu m/0.25\mu m)$, and $(W/L)_{M2} = (W/L)_{M4} = (1.5\mu m/0.25\mu m)$. Assuming $Q = 0$, we determine the minimum sizes of M_5, M_6, M_7, and M_8 to make the device switchable.

To switch the latch from the $Q = 0$ to the $Q = 1$ state, it is essential that the low level of the ratioed, pseudo-NMOS inverter $(M_5\text{-}M_6)\text{-}M_2$ be below the switching threshold of the inverter $M_3\text{-}M_4$ that equals $V_{DD}/2$. It is reasonable to assume that as long as $V_{\overline{Q}} > V_M$, V_Q equals 0, and the gate of transistor M_2 is grounded. The boundary conditions on the transistor sizes can be derived by equating the currents in the inverter for $V_{\overline{Q}} = V_{DD} / 2$, as given in Eq. (7.3) (this ignores channel length modulation). The currents are determined by the saturation current since $V_{SET} = V_{DD} = 2.5\text{V}$ and $V_M = 1.25\text{V}$. We assume that M_5 and M_6 have identical sizes and that $W/L_{5\text{-}6}$ is the effective ratio of the series-connected devices. Under this condition, the pull-down network can be modeled by a single transistor M_{56}, whose length is twice the length of the individual devices.

$$k'_n\left(\frac{W}{L}\right)_{5-6}\left((V_{DD}-V_{Tn})V_{DSATn}-\frac{V_{DSATn}^2}{2}\right) = k'_p\left(\frac{W}{L}\right)_2\left((-V_{DD}-V_{Tp})V_{DSATp}-\frac{V_{DSATp}^2}{2}\right) \tag{7.3}$$

Using the parameters for the 0.25 μm process, Eq. (7.3) results in the constraint that the effective $(W/L)_{M5\text{-}6} \geq 2.26$. This implies that the individual device ratio for M_5 or M_6 must be larger that approximately 4.5. Figure 7.9a shows the DC plot of $V_{\overline{Q}}$ as a function of the individual device sizes of M_5 and M_6. We notice that the individual device ratio of greater than 3 is sufficient to bring the $\overline{Q}$ voltage to the inverter switching threshold. The difference between the manual analysis and simulation arises due to second order effects such as DIBL and channel length modulation. Figure 7.9b, which shows the transient response for different device sizes. The plot confirms that an individual W/L ratio of greater than 3 is required to overpower the feedback and switch the state of the latch.

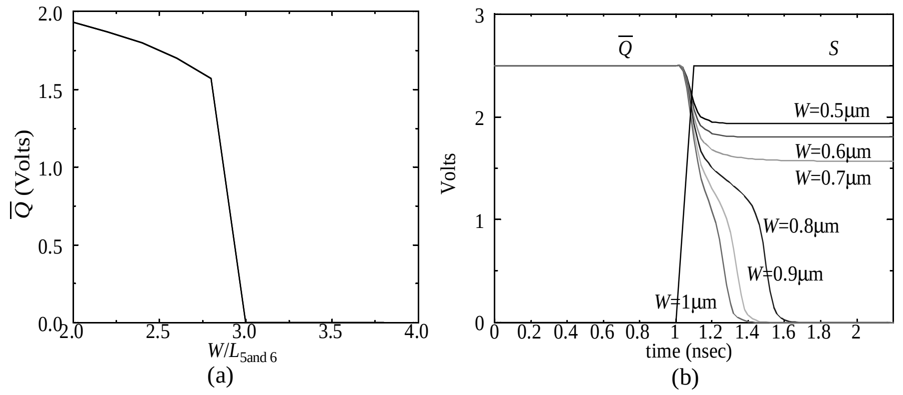


Figure 7.9 Sizing issues for *SR* flip-flop. (a) DC output voltage vs. individual pulldown device size of M_5 and M_6 with W/L_2= 1.5μm/.25μm. (b) Transient response shows that M_5 and M_6 must each have a W/L larger than 3 to swtich the *SR* flip-flop.

The positive feedback effect makes a manual derivation of *propagation delay* of the *SR* latch non-trivial. Some simplifications are therefore necessary. Consider, for instance, the latch of Figure 7.8, where Q and $\overline{Q}$ are set to 0 and 1, respectively. A pulse is applied at node S, causing the latch to toggle. In the first phase of the transient, node $\overline{Q}$ is being pulled down by transistors M_5 and M_6. Since node Q is initially low, the PMOS device M_2 is on while M_1 is off. The transient response is hence determined by the pseudo-NMOS inverter formed by $(M_5\text{-}M_6)$ and M_2. Once $\overline{Q}$ reaches the switching threshold of the CMOS inverter $M_3\text{-}M_4$, this inverter reacts and the positive feedback comes into action, turning M_2 off and M_1 on. This accelerates the pulling down of node $\overline{Q}$. From this analysis, we can derive that the *propagation delay* of node $\overline{Q}$ is approximately equal to the delay of the pseudo-NMOS inverter formed by (M_5-M_6) and M_2. To obtain the delay for node Q, it is sufficient to add the delay of the complementary CMOS inverter M_3-M_4.

Example 7.2 *Propagation Delay* of Static *SR* Flip-Flop

The transient response of the latch in Figure 7.8, as obtained from simulation, is plotted in Figure 7.10. The devices are sized as described in Example 7.1, and a load of a single inverter is assumed for each latch output. The flip-flop is initially in the reset state, and an *S*-pulse is applied. As we can observe, this results first in a discharging of the $\overline{Q}$ output while Q stays at 0. Once the switching threshold of the inverter M_3-M_4 is reached, the Q output starts to rise. The delay of this transient is solely determined by the M_3-M_4 inverter, which is hampered by the slow rise time at its input. From the simulation results, we can derive that $t_{p\overline{Q}}$ and t_{pQ} equal 120psec and 230psec, respectively.

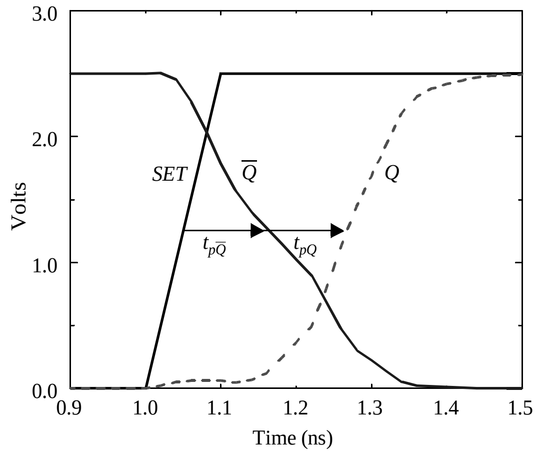


Figure 7.10 Transient response of *SR* flip-flop.

Problem 7.2 Complimentary CMOS *SR* FF

Instead of using the modified *SR* FF of Figure 7.8, it is also possible to use complementary logic to implement the clocked *SR* FF. Derive the transistor schematic (which consists of 12 transistors). This circuit is more complex, but switches faster and consumes less switching power. Explain why.

7.4.3 Multiplexer Based Latches

There are many approaches for constructing latches. One very common technique involves the use of transmission gate multiplexers. Multiplexer based latches can provide similar functionality to the *SR* latch, but has the important added advantage that the sizing of devices only affects performance and is not critical to the functionality.

Figure 7.11 shows an implementation of static positive and negative latches based on multiplexers. For a negative latch, when the clock signal is low, the input 0 of the multiplexer is selected, and the *D* input is passed to the output. When the clock signal is high, the input 1 of the multiplexer, which connects to the output of the latch, is selected. The feedback holds the output stable while the clock signal is high. Similarly in the positive latch, the *D* input is selected when clock is high, and the output is held (using feedback) when clock is low.

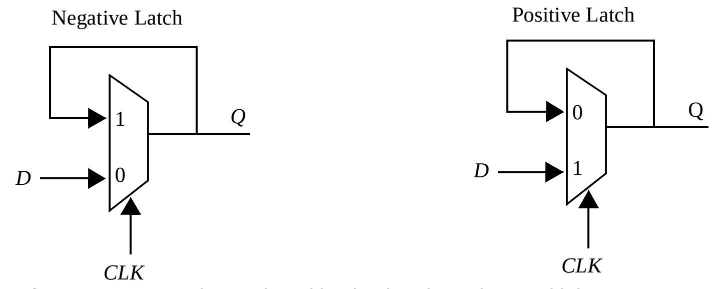


Figure 7.11 Negative and positive latches based on multiplexers.

A transistor level implementation of a positive latch based on multiplexers is shown in Figure 7.12. When *CLK* is high, the bottom transmission gate is *on* and the latch is *transparent* - that is, the *D* input is copied to the *Q* output. During this phase, the feedback loop is open since the top transmission gate is *off*. Unlike the *SR* FF, the feedback does not have to be overridden to write the memory and hence sizing of transistors is not critical for realizing correct functionality. The number of transistors that the clock touches is important since it has an *activity factor* of 1. This particular latch implementation is not particularly efficient from this metric as it presents a load of 4 transistors to the *CLK* signal.

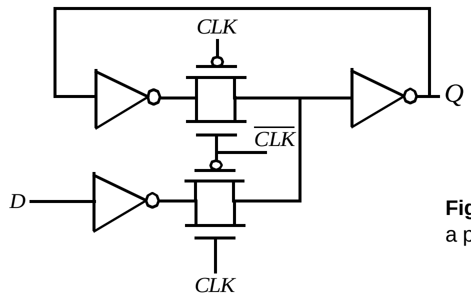
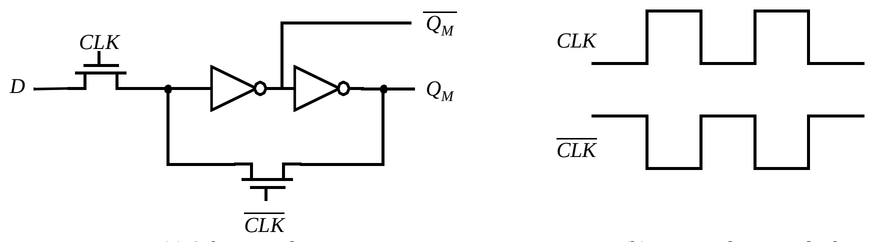


Figure 7.12 Transistor level implementation of a positive latch built using transmission gates.

It is possible to reduce the clock load to two transistors by using implement multiplexers using NMOS only pass transistor as shown in Figure 7.13. The advantage of this approach is the reduced clock load of only two NMOS devices. When *CLK* is high, the latch samples the *D* input, while a low clock-signal enables the feedback-loop, and puts the latch in the hold mode. While attractive for its simplicity, the use of NMOS only pass transistors results in the passing of a degraded high voltage of V_{DD}-V_{Tn} to the input of the first inverter. This impacts both noise margin and the switching performance, especially in the case of low values of V_{DD} and high values of V_{Tn}. It also causes static power dissipation in first inverter, as already pointed out in Chapter 6. Since the maximum input-voltage to the inverter equals V_{DD}-V_{Tn}, the PMOS device of the inverter is never turned off, resulting is a static current flow.



(a) Schematic diagram (b) Nonoverlapping clocks

Figure 7.13 Multiplexer based NMOS latch using NMOS only pass transistors for multiplexers.

7.4.4 Master-Slave Based Edge Triggered Register

The most common approach for constructing an *edge-triggered* register is to use a *master-slave* configuration as shown in Figure 7.14. The register consists of cascading a negative latch (master stage) with a positive latch (slave stage). A multiplexer based latch is used in this particular implementation, though any latch can be used to realize the master and slave stages. On the low phase of the clock, the master stage is *transparent* and the D input is passed to the master stage output, Q_M. During this period, the slave stage is in the *hold* mode, keeping its previous value using feedback. On the rising edge of the clock, the master slave stops sampling the input, and the slave stage starts sampling. During the high phase of the clock, the slave stage samples the output of the master stage (Q_M), while the master stage remains in a *hold* mode. Since Q_M is constant during the high phase of the clock, the output Q makes only one transition per cycle. The value of Q is the value of D right before the rising edge of the clock, achieving the *positive edge-triggered* effect. A *negative edge-triggered* register can be constructed using the same principle by simply switching the order of the positive and negative latch (i.e., placing the positive latch first).

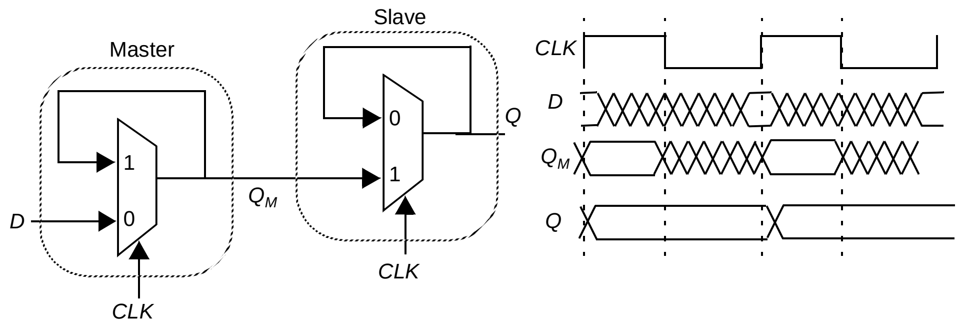


Figure 7.14 *Positive edge-triggered* register based on a *master-slave* configuration.

A complete transistor level implementation of a the *master-slave positive edge-triggered* register is shown in Figure 7.15. The multiplexer is implemented using transmission gates as discussed in the previous section. When clock is low ($\overline{CLK} = 1$), T_1 is *on* and T_2 is *off*, and the D input is sampled onto node Q_M. During this period, T_3 is *off* and T_4 is *on* and the cross-coupled inverters (I_5, I_6) holds the state of the slave latch. When the clock goes high, the master stage stops sampling the input and goes into a *hold* mode. T_1 is *off* and T_2

is *on*, and the cross coupled inverters I_3 and I_4 holds the state of Q_M. Also, T_3 is *on* and T_4 is *off*, and Q_M is copied to the output Q.

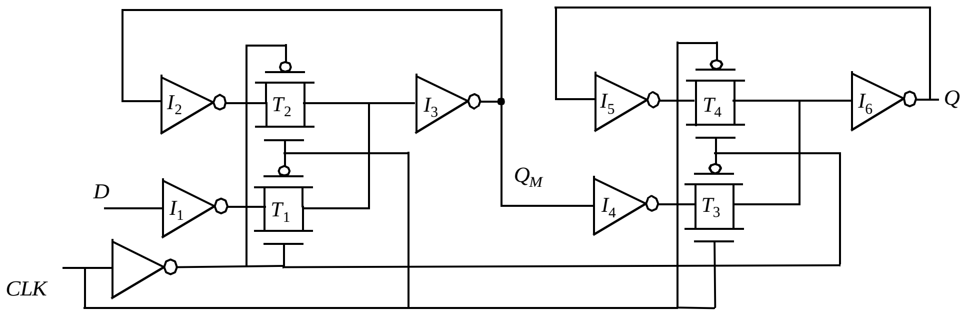


Figure 7.15 Transistor-level implementation of a *master-slave postive edge-triggered* register using multiplexers.

Problem 7.3Optimization of the Master Slave Register

It is possible to remove the inverters I_1 and I_2 from Figure 7.3 without loss of functionality. Is there any advantage in including these inverters in the implementation?

Timing Properties of the multiplexer Bases Master-Slave Register. As discussed earlier, there are three important timing metrics in registers: the *set up* time, the *hold time* and the *propagation delay*. It is important to understand these factors that affect the timing parameters and develop the intuition to manually estimate the parameters. Assume that the *propagation delay* of each inverter is t_{pd_inv} and the *propagation delay* of the transmission gate is t_{pd_tx}. Also assume that the *contamination delay* is 0 and the inverter delay to derive $\overline{CLK}$ from CLK has a delay equal to 0.

The *set-up time* is the time before the rising edge of the clock that the input data D must become valid. Another way to ask the question is how long before the rising edge does the D input have to be stable such that Q_M samples the value reliably. For the transmission gate multiplexer-based register, the input D has to propagate through I_1, T_1, I_3 and I_2 before the rising edge of the clock. This is to ensure that the node voltages on both terminals of the transmission gate T_2 are at the same value. Otherwise, it is possible for the cross-coupled pair I_2 and I_3 to settle to an incorrect value. The *set-up time* is therefore equal to $3 * t_{pd\_inv} + t_{pd\_tx}$.

The *propagation delay* is the time for the value of Q_M to propagate to the output Q. Note that since we included the delay of I_2 in the *set-up time*, the output of I_4 is valid before the rising edge of clock. Therefore the delay $t_{c\text{-}q}$ is simply the delay through T_3 and I_6 ($t_{c\text{-}q} = t_{pd\_tx} + t_{pd\_inv}$).

The *hold time* represents the time that the input must be held stable after the rising edge of the clock. In this case, the transmission gate T_1 turns off when clock goes high and therefore any changes in the D-input after clock going high are not seen by the input. Therefore, the *hold time* is 0.

Example 7.3 Timing analysis using SPICE.

To obtain the *set-up time* of the register using SPICE, we progressively skew the input with respect to the clock edge until the circuit fails. Figure 7.16 shows the *set-up time* simulation assuming a skew of 210 psec and 200 psec. For the 210 psec case, the correct value of input *D* is sampled (in this case, the *Q* output remains at the value of V_{DD}). For a skew of 200 psec, an incorrect value propagates to the output (in this case, the *Q* output transitions to 0). Node Q_M starts to go high while the output of I_2 (the input to transmission gate T_2) starts to fall. However, the clock is enabled before the two nodes across the transmission gate (T_2) settle to the same value and therefore, results in an incorrect value written into the master latch.The *set-up time* for this register is therefore 210 psec.

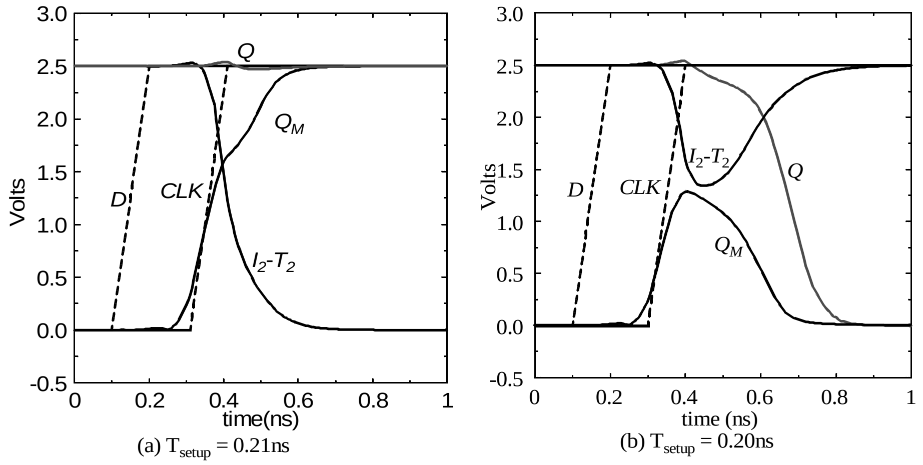


Figure 7.16 *Set-up time* simulation.

In a similar fashion, the *hold time* can be simulated. The *D* input edge is once again skewed relative to the clock signal till the circuit stop functioning. For this design, the *hold time* is 0 - i.e., the inputs can be changed on the clock edge. Finally, for the *propagation delay*, the inputs are transitioned at least a *set-up time* before the rising edge of the clock and the delay is measured from the 50% point of the *CLK* edge to the 50% point of the *Q* output. From this simulation (Figure 7.17), $t_{c\text{-}q\,(lh)}$ was 160psec and $t_{c\text{-}q(hl)}$ was 180psec.

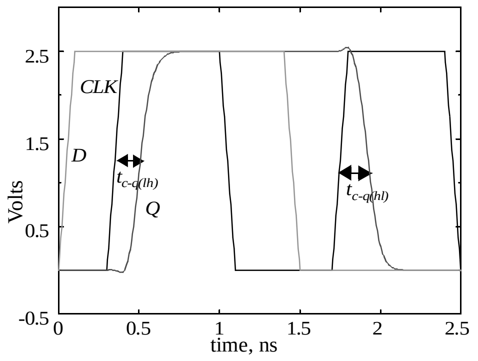


Figure 7.17 Simulation of *propagation delay*.

As mentioned earlier, the drawback of the transmission gate register is the high capacitive load presented to the clock signal. The clock load per register is important since it directly impacts the power dissipation of the clock network. Ignoring the overhead required to invert the clock signal (since the buffer inverter overhead can be amortized over multiple register bits), each register has a clock load of 8 transistors. One approach to reduce the clock load at the cost of robustness is to make the circuit ratioed. Figure 7.18 shows that the feedback transmission gate can be eliminated by directly cross coupling the inverters.

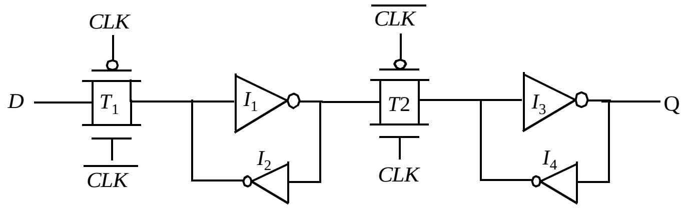


Figure 7.18 Reduced load clock load static *master-slave* register.

The penalty for the reduced clock load is increased design complexity. The transmission gate (T_1) and its source driver must overpower the feedback inverter (I_2) to switch the state of the cross-coupled inverter.The sizing requirements for the transmission gates can be derived using a similar analysis as performed for the *SR* flip-flop. The input to the inverter I_1 must be brought below its switching threshold in order to make a transition. If minimum-sized devices are to be used in the transmission gates, it is essential that the transistors of inverter I_2 should be made even weaker. This can be accomplished by making their channel-lengths larger than minimum. Using minimum or close-to-minimum-size devices in the transmission gates is desirable to reduce the power dissipation in the latches and the clock distribution network.

Another problem with this scheme is the *reverse conduction* — this is, the second stage can affect the state of the first latch. When the slave stage is *on* (Figure 7.19), it is possible for the combination of T_2 and I_4 to influence the data stored in I_1-I_2 latch. As long as I_4 is a weak device, this is fortunately not a major problem.

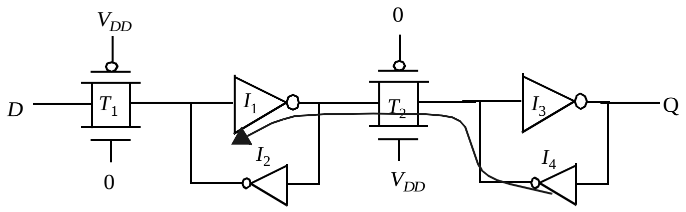


Figure 7.19 Reverse conduction possible in the transmission gate.

7.4.5 Non-ideal clock signals

So far, we have assumed that $\overline{CLK}$ is a perfect inversion of CLK, or in other words, that the delay of the generating inverter is zero. Even if this were possible, this would still not be a good assumption. Variations can exist in the wires used to route the two clock signals, or the load capacitances can vary based on data stored in the connecting latches. This effect, known as *clock skew* is a major problem, and causes the two clock signals to overlap as is shown in Figure 7.20b. *Clock-overlap* can cause two types of failures, as illustrated for the NMOS-only negative *master-slave* register of Figure 7.20a.

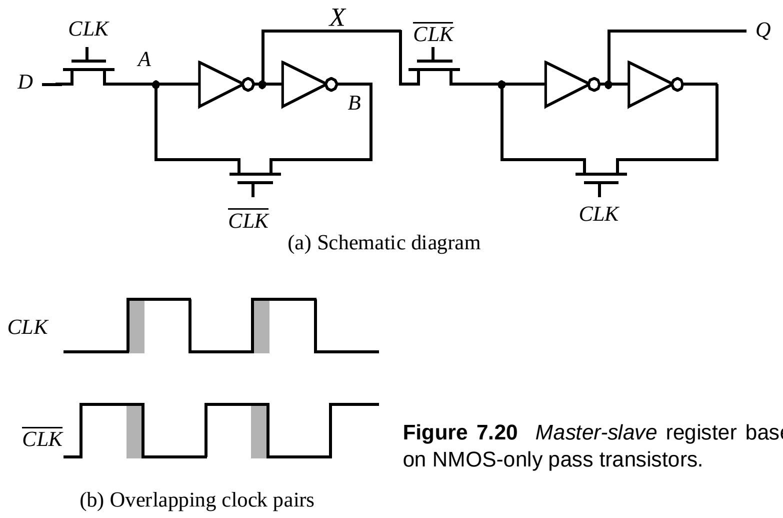
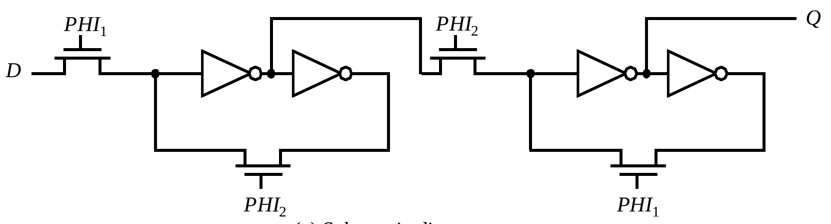


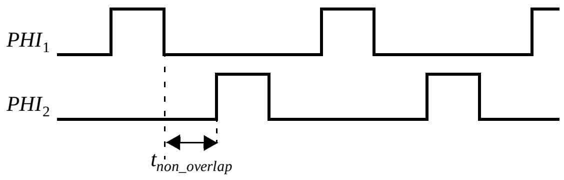
Figure 7.20 *Master-slave* register based on NMOS-only pass transistors.

- When the clock goes high, the slave stage should stop sampling the master stage output and go into a *hold* mode. However, since *CLK* and $\overline{CLK}$ are both high for a short period of time (the *overlap period*), both sampling pass transistors conduct and there is a direct path from the *D* input to the *Q* output. As a result, data at the output can change on the rising edge of the clock, which is undesired for a *negative edge-triggered* register. The is know as a *race* condition in which the value of the output *Q* is a function of whether the input *D* arrives at node *X* before or after the falling edge of $\overline{CLK}$. If node *X* is sampled in the metastable state, the output will switch to a value determined by noise in the system.
- The primary advantage of the multiplexer-based register is that the feedback loop is open during the sampling period, and therefore sizing of devices is not critical to functionality. However, if there is clock overlap between *CLK* and $\overline{CLK}$, node *A* can be driven by both *D* and *B*, resulting in an undefined state.

Those problems can be avoided by using two *non-overlapping clocks* PHI_1 and PHI_2 instead (Figure 7.21), and by keeping the nonoverlap time $t_{non\_overlap}$ between the clocks large enough such that no overlap occurs even in the presence of clock-routing delays. During the nonoverlap time, the *FF* is in the high-impedance state—the feedback loop is open, the loop gain is zero, and the input is disconnected. Leakage will destroy the state if this condition holds for too long a time. Hence the name *pseudostatic*: the register employs a combination of static and dynamic storage approaches depending upon the state of the clock.



(a) Schematic diagram



(b) Two-phase nonoverlapping clocks

Figure 7.21 Pseudostatic two-phase *D* register.

Problem 7.4 Generating non-overlapping clocks

Figure 7.22 shows one possible implementation of the clock generation circuitry for generating a two-phase non-overlapping clocks. Assuming that each gate has a unit gate delay, derive the timing relationship between the input clock and the two output clocks. What is the non-overlap period? How can this period be increased if needed?

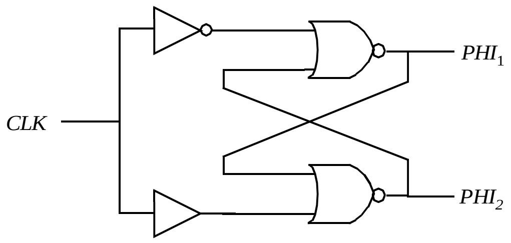


Figure 7.22 Circuitry for generating a two phase non-overlapping clock

7.4.6 Low-Voltage Static Latches

The scaling of supply voltages is critical for low power operation. Unfortunately, certain latch structures don't function at reduced supply voltages. For example, without the scaling of device thresholds, NMOS only pass transistors (e.g., Figure 7.21) don't scale well with supply voltage due to its inherent threshold drop. At very low power supply voltages, the input to the inverter cannot be raised above the switching threshold, resulting in incorrect evaluation. Even with the use of transmission gates, performance degrades significantly at reduced supply voltages.

Scaling to low supply voltages hence requires the use of reduced threshold devices. However, this has the negative effect of exponentially increasing the sub-threshold leakage power as discussed in Chapter 6. When the registers are constantly accessed, the leakage energy is typically insignificant compared to the switching power. However, with the use of conditional clocks, it is possible that registers are idle for extended periods and the leakage energy expended by registers can be quite significant.

Many solutions are being explored to address the problem of high leakage during idle periods. One approach for this involves the use of Multiple Threshold devices as shown in Figure 7.23 [Mutoh95]. Only the negative latch is shown here. The shaded inverters and transmission gates are implemented in low-threshold devices. The low-threshold inverters are gated using high threshold devices to eliminate leakage.

During normal mode of operation, the sleep devices are tuned *on*. When clock is low, the *D* input is sampled and propagates to the output. When clock is high, the latch is in the *hold* mode. The feedback transmission gate conducts and the cross-coupled feedback is enabled. Note there is an extra inverter, needed for storage of state when the latch is in the *sleep* state. During idle mode, the high threshold devices in series with the low threshold inverter are turned *off* (the *SLEEP* signal is high), eliminating leakage. It is assumed that clock is in the high state when the latch is in the sleep state. The feedback low-threshold transmission gate is turned *on* and the cross-coupled high-threshold devices maintains the state of the latch.

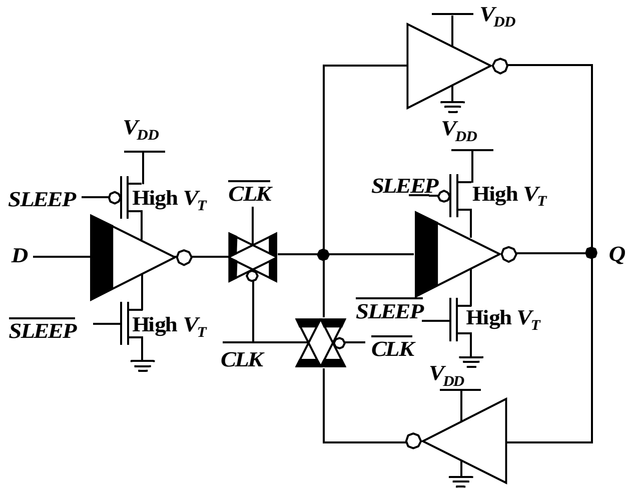


Figure 7.23 One solution for the leakage problem in low-voltage operation using MTCMOS.

Problem 7.5Transistor minimization in the MTCMOS register

Unlike combination logic, both flavors of high threshold devices in series are required to eliminate the leakage of low threshold gates. Explain why this is the case. Hint: Eliminate the high V_T NMOS or high V_T PMOS of the low threshold inverter on the right of Figure 7.23 and investigate potential leakage paths.

7.5 Dynamic Latches and Registers

Storage in a static sequential circuit relies on the concept that a cross-coupled inverter pair produces a bistable element and can thus be used to memorize binary values. This approach has the useful property that a stored value remains valid as long as the supply voltage is applied to the circuit, hence the name *static*. The major disadvantage of the static gate, however, is its complexity. When registers are used in computational structures that are constantly clocked such as pipelined datapath, the requirement that the memory should hold state for extended periods of time can be significantly relaxed.

This results in a class of circuits based on temporary storage of charge on parasitic capacitors. The principle is exactly identical to the one used in dynamic logic — charge stored on a capacitor can be used to represent a logic signal. The absence of charge denotes a 0, while its presence stands for a stored 1. No capacitor is ideal, unfortunately, and some charge leakage is always present. A stored value can hence only be kept for a limited amount of time, typically in the range of milliseconds. If one wants to preserve signal integrity, a periodic *refresh* of its value is necessary. Hence the name *dynamic* storage. Reading the value of the stored signal from a capacitor without disrupting the charge requires the availability of a device with a high input impedance.

7.5.1 Dynamic Transmission-Gate Based Edge-triggred Registers

A fully dynamic *positive edge-triggered* register based on the *master-slave* concept is shown in Figure 7.24. When $CLK = 0$, the input data is sampled on storage node 1, which has an equivalent capacitance of C_1 consisting of the gate capacitance of I_1, the junction capacitance of T_1, and the overlap gate capacitance of T_1. During this period, the slave stage is in a *hold* mode, with node 2 in a high-impedance (floating) state. On the rising edge of clock, the transmission gate T_2 turns on, and the value sampled on node 1 right before the rising edge propagates to the output Q (note that node 1 is stable during the high phase of the clock since the first transmission gate is turned *off*). Node 2 now stores the inverted version of node 1. This implementation of an *edge-triggered* register is very efficient as it requires only 8 transistors. The sampling switches can be implemented using NMOS-only pass transistors, resulting in an even-simpler 6 transistor implementation. The reduced transistor count is attractive for high-performance and low-power systems.

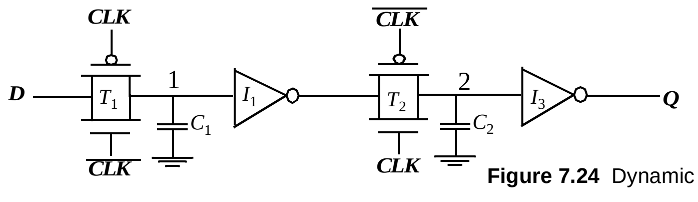


Figure 7.24 Dynamic *edge-triggered* register.

The *set-up time* of this circuit is simply the delay of the transmission gate, and corresponds to the time it takes node 1 to sample the D input. The *hold time* is approximately zero, since the transmission gate is turned *off* on the clock edge and further inputs changes are ignored. The *propagation delay* (t_{c-q}) is equal to two inverter delays plus the delay of the transmission gate T_2.

One important consideration for such a dynamic register is that the storage nodes (i.e., the state) has to be refreshed at periodic intervals to prevent a loss due to charge leakage, due to diode leakage as well as sub-threshold currents. In datapath circuits, the refresh rate is not an issue since the registers are periodically clocked, and the storage nodes are constantly updated.

Clock overlap is an important concern for this register. Consider the clock waveforms shown in Figure 7.25. During the 0-0 overlap period, the NMOS of T_1 and the PMOS of T_2 are simultaneously on, creating a direct path for data to flow from the D input of the register to the Q output. This is known as a *race condition*. The output Q can change on the falling edge if the overlap period is large — obviously an undesirable effect for a *positive edge-triggered* register. The same is true for the 1-1 overlap region, where an input-output path exists through the PMOS of T_1 and the NMOS of T_2. The latter case is taken care off by enforcing a *hold* time constraint. That is, the data must be stable during the high-high overlap period. The former situation (0-0 overlap) can be addressed by making sure that there is enough delay between the D input and node 2 ensuring that new data sampled by the master stage does not propagate through to the slave stage. Generally the built in single inverter delay should be sufficient and the overlap period constraint is given as:

$$t_{overlap0-0} < t_{T1} + t_{I1} + t_{T2} \tag{7.4}$$

Similarly, the constraint for the 1-1 overlap is given as:

$$t_{hold} > t_{overlap1-1} \tag{7.5}$$

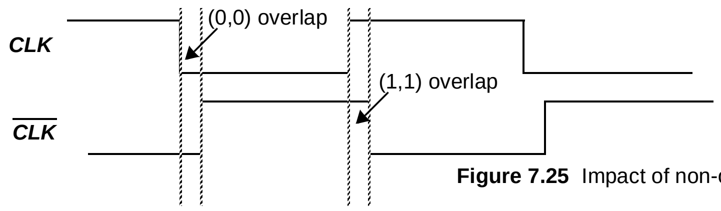


Figure 7.25 Impact of non-overlapping clocks.

7.5.2 C^{2}MOS Dynamic Register: A Clock Skew Insensitive Approach

The C^{2}MOS Register

Figure 7.26 shows an ingenious *positive edge-triggered* register based on the *master-slave* concept which is insensitive to clock overlap. This circuit is called the C^{2}MOS (Clocked CMOS) *register* [Suzuki73]. The register operates in two phases.

1. $CLK = 0$ ($\overline{CLK} = 1$): The first tri-state driver is turned on, and the master stage acts as an inverter sampling the inverted version of D on the internal node X. The master stage is in the *evaluation mode*. Meanwhile, the slave section is in a high-impedance mode, or in a *hold mode*. Both transistors M_7 and M_8 are off, decoupling the output from the input. The output Q retains its previous value stored on the output capacitor C_{L2}.

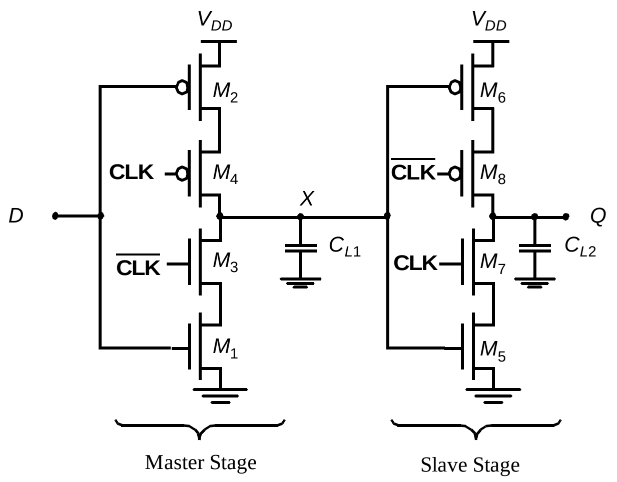


Figure 7.26 C^{2}MOS *master-slave positive edge-triggered register*.

2. The roles are reversed when $CLK = 1$: The master stage 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 through the slave stage which acts as an inverter.

The overall circuit operates as a *positive edge-triggered master-slave* register — very similar to the transmission-gate based register presented earlier. However, there is an important difference:

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

To prove the above statement, we examine both the (0-0) and (1-1) overlap cases (Figure 7.25). In the (0-0) overlap case, the circuit simplifies to the network shown in Figure 7.27a in which both PMOS devices are *on* during this period. The question is does any new data sampled during the overlap window propagate to the output Q. This is not desirable since data should not change on the negative edge for a *positive edge-triggered register.* Indeed new data is sampled on node X through the series PMOS devices M_2-M_4, and node X can make a 0-to-1 transition during the overlap period. However, this data cannot propagate to the output since the NMOS device M_7 is turned off. At the end of the overlap period, $\overline{CLK}$=1 and both M_7 and M_8 turn *off*, putting the slave stage is in the *hold* mode. Therefore, any new data sampled on the falling clock edge is not seen at the slave output Q, since the slave state is off till the next rising edge of the clock. 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.

The (1-1) overlap case (Figure 7.27b), where both NMOS devices M_3 and M_7 are turned on, is somewhat more contentious. The question is again if new data sampled during the overlap period (right after clock goes high) propagates to the Q output. A *positive edge-triggered* register may only pass data that is presented at the input *before* the rising edge. If the D input changes during the overlap period, node X can make a 1-to-0 transition, but cannot propagate to the output. However, as soon as the overlap period is over, the PMOS M_8 is turned on and the 0 propagates to output. This effect is not desirable. The

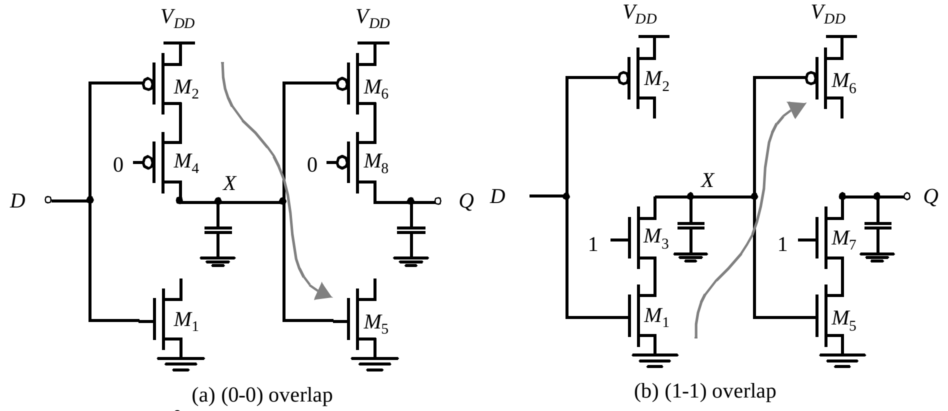


Figure 7.27 C^{2}MOS D FF during overlap periods. No feasible signal path can exist between *In* and *D*, as illustrated by the arrows.

problem is fixed by imposing a *hold time* constraint on the input data, D, or, in other words, the data D should be stable during the overlap period.

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* 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 (or fall time) is smaller than approximately five times the *propagation delay* of the register. 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 7.28, which plots the simulated transient response of a C^{2}MOS D FF for clock slopes of respectively 0.1 and 3 nsec. For slow clocks, the potential for a *race condition* exists.

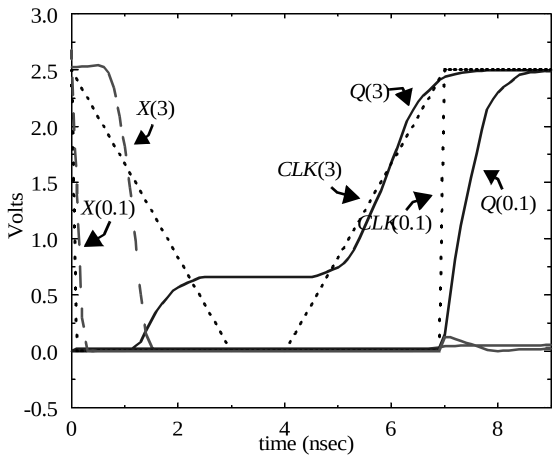


Figure 7.28 Transient response of C^{2}MOS FF for 0.1 nsec and 3 nsec clock rise (fall) times assuming *In* = 1.

Dual-edge Triggered Registers

So far, we have focused on *edge-triggered* registers that sample the input data on only one of the clock edges (rising or falling). It is also possible to design sequential circuits that sample the input on both edges. The advantage of this scheme is that a lower frequency clock (half of the original rate) is distributed for the same functional throughput, resulting in power savings in the clock distribution network. Figure 7.29 shows a modification of the C^2MOS register to enable sampling on both edges [REFERENCE]. It consists of two parallel *master-slave* based *edge-triggered* registers, whose outputs are multiplexed using the tri-state drivers.

When clock is high, the positive latch composed of transistors M_1-M_4 is sampling the inverted D input on node X. Node Y is held stable, since devices M_9 and M_{10} are turned *off*. On the falling edge of the clock, the top slave latch M_5-M_8 turns on, and drives the inverted value of X to the Q output. During the low phase, the bottom master latch (M_1, M_4, M_9, M_{10}) is turned on, sampling the inverted D input on node Y. Note that the devices M_1 and M_4 are reused, reducing the load on the D input. On the rising edge, the bottom slave latch conducts, and drives the inverted version of Y on node Q. Data hence changes on both edges. Note that the slave latches operate in a complementary fashion — this is, only one of them is turned on during each phase of the clock.

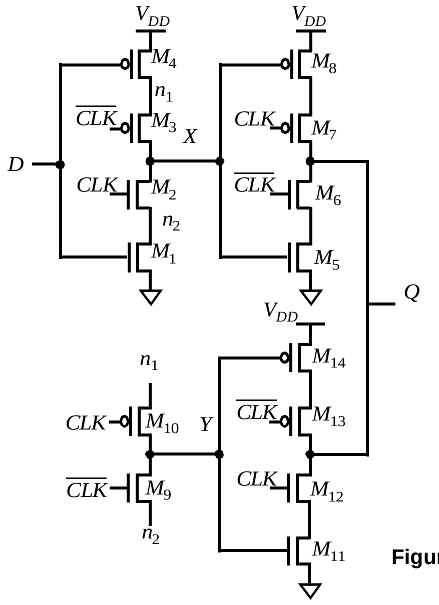


Figure 7.29 C^2MOS based dual-edge triggered register.

Problem 7.6 Dual-edge Registers

Determine how the adoption of dual-edge registers influences the power-dissipation in the clock-distribution network.

7.5.3 True Single-Phase Clocked Register (TSPCR)

In the two-phase clocking schemes described above, care must be taken in routing the two clock signals to ensure that overlap is minimized. While the C^2MOS provides a skew-tolerant solution, it is possible to design registers that only use a single phase clock. The *True Single-Phase Clocked Register* (TSPCR) proposed by Yuan and Svensson uses a **single clock** (without an inverse clock) [Yuan89]. The basic single-phase positive and negative latches are shown in Figure 7.30. For the positive latch, when *CLK* is high, the latch is in

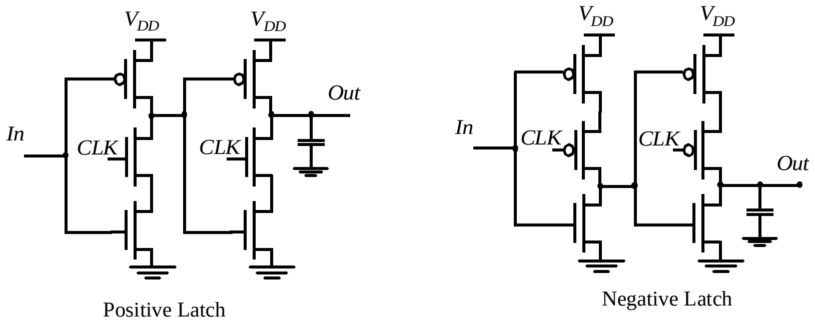


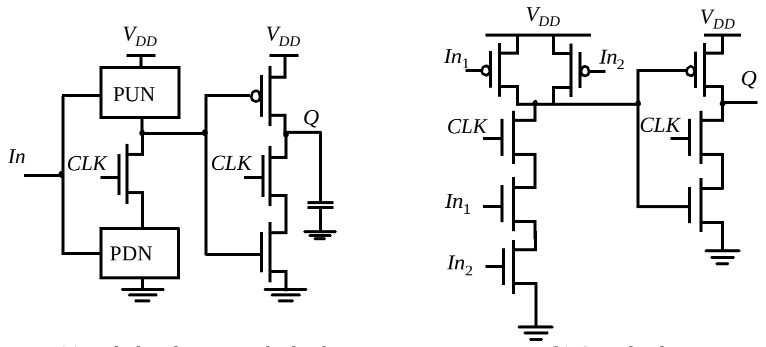
Figure 7.30 True Single Phase Latches.

the *transparent mode* and corresponds to two cascaded inverters; the latch is non-inverting, and propagates the input to the output. On the other hand, when *CLK* = 0, both inverters are disabled, and the latch is in *hold-mode*. Only the pull-up networks are still active, while the pull-down circuits are deactivated. As a result of the dual-stage approach, no signal can ever propagate from the input of the latch to the output in this mode. A register can be constructed by cascading positive and negative latches. The clock load is similar to a conventional transmission gate register, or C^2MOS register. The main advantage is the use of a single clock phase. The disadvantage is the slight increase in the number of transistors — 12 transistors are required.

TSPC offers an additional advantage: the possibility of embedding logic functionality into the latches. This reduces the delay overhead associated with the latches. Figure 7.31a outlines the basic approach for embedding logic, while Figure 7.31b shows an example of a positive latch that implements the AND of In_1 and In_2 in addition to performing the latching function. While the *set-up time* of this latch has increased over the one shown in Figure 7.30, the overall performance of the digital circuit (that is, the clock period of a sequential circuit) has improved: the increase in *set-up time* is typically smaller than the delay of an AND gate. This approach of embedding logic into latches has been used extensively in the design of the EV4 DEC Alpha microprocessor [Dobberpuhl92] and many other high performance processors.

Example 7.4 Impact of embedding logic into latches on performance

Consider embedding an AND gate into the TSPC latch, as shown in Figure 7.31b. In a 0.25 μm, the *set-up time* of such a circuit using minimum-size devices is 140 psec. A conventional approach, composed of an AND gate followed by a positive latch has an effective *set-up time* of 600 psec (we treat the AND plus latch as a black box that performs the AND+latching

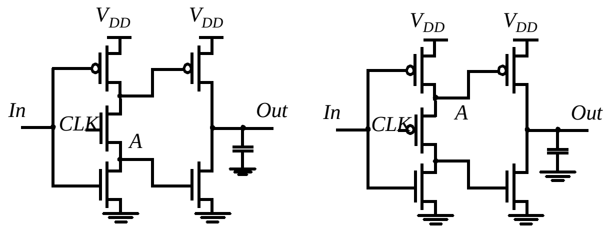


(a) Including logic into the latch (b) AND latch

Figure 7.31 Adding logic to the TSPC approach.

functions). The embedded logic approach hence results in significant performance improvements.

The TSPC latch circuits can be further reduced in complexity as illustrated in Figure 7.32, where only the first inverter is controlled by the clock. Besides the reduced number of transistors, these circuits have the advantage that the clock load is reduced by half. On the other hand, not all node voltages in the latch experience the full logic swing. For instance, the voltage at node *A* (for V_{in} = 0 V) for the positive latch maximally equals $V_{DD} - V_{Tn}$, which results in a reduced drive for the output NMOS transistor and a loss in performance. Similarly, the voltage on node *A* (for $V_{in} = V_{DD}$) for the negative latch is only driven down to $|V_{Tp}|$. This also limits the amount of V_{DD} scaling possible on the latch.



(a) Positive Latch (b) Negative Latch

Figure 7.32 Simplified TSPC latch (also called split-output).

Figure 7.33 shows the design of a specialized *single-phase edge-triggered register*. When *CLK* = 0, the input inverter is sampling the inverted *D* input on node *X*. The second (dynamic) inverter is in the precharge mode, with M_6 charging up node *Y* to V_{DD}. The third inverter is in the *hold* mode, since M_8 and M_9 are *off*. Therefore, during the low phase of the clock, the input to the final (static) inverter is holding its previous value and the output *Q* is stable. On the rising edge of the clock, the dynamic inverter M_4-M_6 evaluates. If *X* is high on the rising edge, node *Y* discharges. The third inverter M_7-M_8 is on during the high phase, and the node value on *Y* is passed to the output *Q*. On the positive phase of the clock, note that node *X* transitions to a low if the *D* input transitions to a high level. Therefore, the input must be kept stable till the value on node *X* before the rising edge of the clock propagates to *Y*. This represents the *hold time* of the register (note that the *hold time* less than 1 inverter delay since it takes 1 delay for the input to affect node *X*). The *propa-*

gation delay of the register is essentially three inverters since the value on node X must propagate to the output Q. Finally, the *set-up time* is the time for node X to be valid, which is one inverter delay.

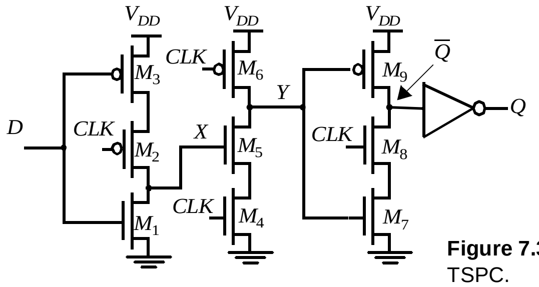
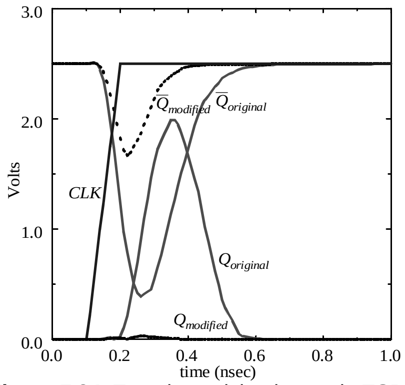


Figure 7.33 *Positive edge-triggered* register TSPC.

WARNING: Similar to the C^2MOS latch, the TSPC latch malfunctions when the *slope of the clock* is not sufficiently steep. Slow clocks cause both the NMOS and PMOS clocked transistors to be on simultaneously, resulting in undefined values of the states and race conditions. The clock slopes should therefore be carefully controlled. If necessary, local buffers must be introduced to ensure the quality of the clock signals.

Example 7.5 TSPC Edge-Triggered Register

Transistor sizing is critical for achieving correct functionality in the TSPC register. With improper sizing, glitches may occur at the output due to a *race condition* when the clock transitions from low to high. Consider the case where D is low and $\overline{Q}$=1 (Q=0). While CLK is low, Y is pre-charged high turning on M_7. When CLK transitions from low to high, nodes Y and Q start to discharge simultaneously (through M_4-M_5 and M_7-M_8, respectively). Once Y is sufficiently low, the trend on $\overline{Q}$ is reversed and the node is pulled high anew through M_9. In a sense, this chain of events is comparable to what would happen if we chain dynamic logic gates. Figure 7.34 shows the transient response of the circuit of Figure 7.34 for different sizes of devices in the final two stages.



	M_4, M_5	M_7, M_8
Original Width	0.5μm	2μm
Modified Width	1μm	1μm

Figure 7.34 Transistor sizing issues in TSPC (for the register of Figure 7.33).

This glitch may be the cause of fatal errors, as it may create unwanted events (for instance, when the output of the latch is used as a clock signal input to another register). It also reduces the *contamination delay* of the register. The problem can be corrected by resizing the relative strengths of the pull-down paths through M_4-M_5 and M_7-M_8, so that Y discharges much faster than $\overline{Q}$. This is accomplished by reducing the strength of the M_7-M_8 pulldown path, and by speeding up the M_4 -M_5 pulldown path.

7.6 Pulse Registers

Until now, we have used the *master-slave* configuration to create an *edge-triggered* register. A fundamentally different approach for constructing a register uses pulse signals. The idea is to construct a short pulse around the rising (or falling) edge of the clock. This pulse acts as the clock input to a latch (e.g., a TSPC flavor is shown in Figure 7.35a), sampling the input only in a short window. Race conditions are thus avoided by keeping the opening time (i.e, the *transparent* period) of the latch very short. The combination of the glitch-generation circuitry and the latch results in a *positive edge-triggered* register.

Figure 7.35b shows an example circuit for constructing a short intentional glitch on each rising edge of the clock [Kozo96]. When CLK = 0, node X is charged up to V_{DD} (M_N is *off* since $CLKG$ is low). On the rising edge of the clock, there is a short period of time when both inputs of the AND gate are high, causing $CLKG$ to go high. This in turn activates M_N, pulling X and eventually $CLKG$ low (Figure 7.35c). The length of the pulse is controlled by the delay of the AND gate and the two inverters. Note that there exists also a delay between the rising edges of the input clock (CLK) and the glitch clock ($CLKG$) — also equal to the delay of the AND gate and the two inverters. If every register on the chip uses the same clock generation mechanism, this sampling delay does not matter. However, process variations and load variations may cause the delays through the glitch clock circuitry to be different. This must be taken into account when performing timing verification and clock skew analysis (which is the topic of a later Chapter).

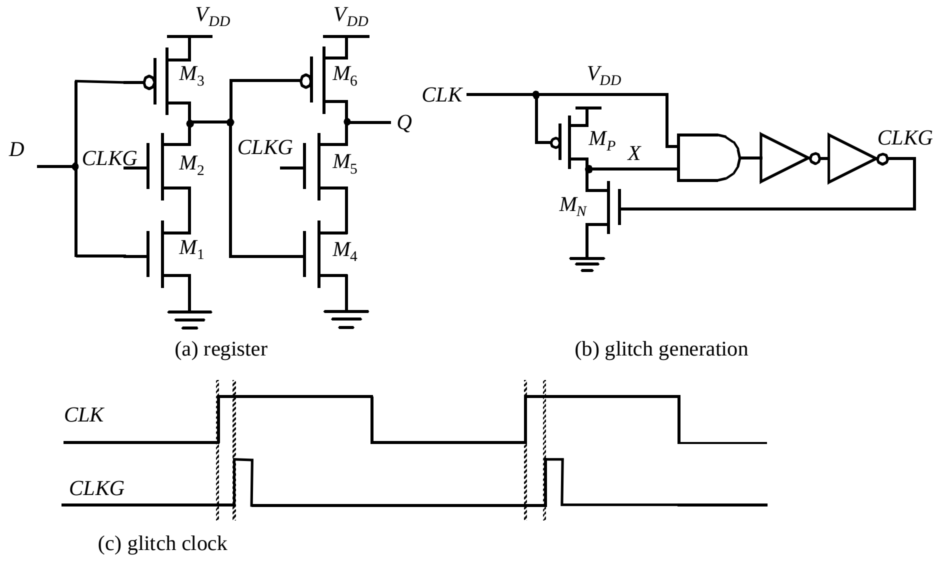


Figure 7.35 Glitch latch - timing generation and register.

If *set-up time* and *hold time* are measured in reference to the rising edge of the glitch clock, the *set-up time* is essentially zero, the *hold time* is equal to the length of the pulse (if the *contamination delay* is zero for the gates), and the *propagation delay* ($t_{c\text{-}q}$) equals two gate delays. The advantage of the approach is the reduced clock load and the small number of transistors required. The glitch-generation circuitry can be amortized over multiple register bits. The disadvantage is a substantial increase in verification complexity. This has prevented a wide-spread use. They do however provide an alternate approach to conventional schemes, and have been adopted in some high performance processors (e.g., [Kozo96]).

Another version of the pulsed register is shown in Figure 7.36 (as used in the AMD-K6 processor [Partovi96]). When the clock is low, M_3 and M_6 are *off* and device P_1 is turned on. Node X is precharged to V_{DD}, the output node (Q) is decoupled from X and is held at its previous state. $\overline{CLKD}$ is a delay-inverted version of CLK. On the rising edge of the clock, M_3 and M_6 turn *on* while devices M_1 and M_4 stay on for a short period, determined by the delay of the three inverters. During this interval, the circuit is *transparent* and the input data D is sampled by the latch. Once $\overline{CLKD}$ goes low, node X is decoupled from the D input and is either held or starts to precharge to V_{DD} by PMOS device P_2. On the falling edge of the clock, node X is held at V_{DD} and the output is held stable by the cross-coupled inverters.

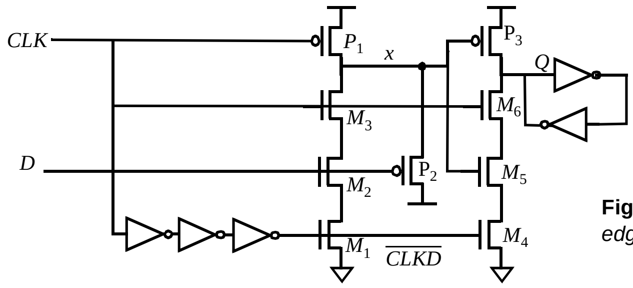


Figure 7.36 Flow-through *positive edge-triggered* register.

Note that this circuit also uses a one-shot, but the one-shot is integrated into the register. The *transparency* period also determines the *hold time* of the register. The window must be wide enough for the input data to propagate to the Q output. In this particular circuit, the *set-up time* can be negative. This is the case if the *transparency* window is longer than the delay from input to output. This is attractive, as data can arrive at the register even after the clock goes high, which means that time is borrowed from the previous cycle.

Example 7.6 *Set-up time* of glitch register

The glitch register of Figure 7.36 is transparent during the (1-1) overlap of CLK and $\overline{CLKD}$. As a result, the input data can actually change after the rising edge of the clock, resulting in a negative *set-up time* (Figure 7.37). The D-input transitions to low after the rising edge of the clock, and transitions high before the falling edge of $\overline{CLKD}$ (this is, during the transparency period). Observe how the output follows the input. The output Q does go to the correct value of V_{DD} as long as the input D is set up correctly some time before the falling edge of $\overline{CLKD}$. When the negative *set-up time* is exploited, there can be no gurantees on the monotonic behavior of the output. That is, the output can have multiple transitions around the rising edge, and therefore, the output of the register should not be used as a clock to other registers.

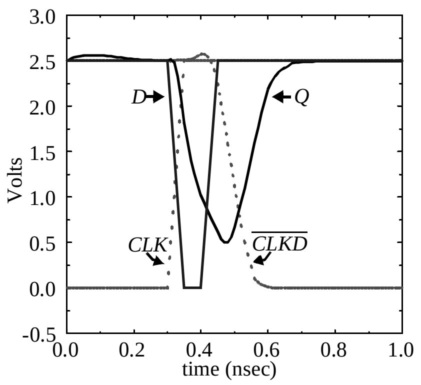


Figure 7.37 Simulation showing a negative *set-up time* for the glitch register.

Problem 7.7 Converting a glitch register to a conditional glitch register

Modify the circuit in Figure 7.36 so that it takes an additional *Enable* input. The goal is to convert the register to a conditional register which latches only when the enable signal is asserted.

7.7 Sense-Amplifier Based Registers

So far, we have presented two fundamental approaches towards building *edge-triggered* registers: the *master-slave* concept and the glitch technique. Figure 7.38 introduces another technique that uses a *sense amplifier* structure to implement an *edge-triggered* register [Montanaro96]. *Sense amplifier* circuits accept small input signals and amplify them to generate rail-to-rail swings. As we will see, *sense amplifier* circuits are used extensively in memory cores and in low swing bus drivers to amplify small voltage swings

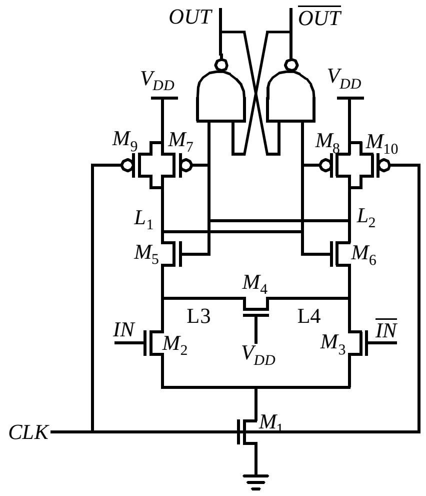


Figure 7.38 *Positive edge-triggered* register based on sense-amplifier.

present in heavily loaded wires. There are many techniques to construct these amplifiers, with the use of feedback (e.g., cross-coupled inverters) being one common approach. The circuit shown in Figure 7.38 uses a precharged front-end amplifier that samples the differential input signal on the rising edge of the clock signal. The outputs of front-end are fed into a NAND cross-coupled *SR FF* that holds the data and gurantees that the differential outputs switch only once per clock cycle. The differential inputs in this implementation don't have to have rail-to-rail swing and hence this register can be used as a receiver for a reduced swing differential bus.

The core of the front-end consists of a cross-coupled inverter (M_5-M_8) whose outputs (L_1 and L_2) are precharged using devices M_9 and M_{10} during the low phase of the clock. As a result, PMOS transistors M_7 and M_8 to be turned *off* and the NAND *FF* is holding its previous state. Transistor M_1 is similar to an evaluate switch in dynamic circuits and is turned *off* ensuring that the differential inputs don't affect the output during the low phase of the clock. On the rising edge of the clock, the evaluate transistor turns *on* and the differential input pair (M_2 and M_3) is enabled, and the difference between the input signals is amplified on the output nodes on L_1 and L_2. The cross-coupled inverter pair flips to one of its the stable states based on the value of the inputs. For example, if *IN* is 1, L_1 is pulled to 0, and L_2 remains at V_{DD}. Due to the amplifying properties of the input stage, it is *not* necessary for the input to swing all the way up to V_{DD} and enables the use of low-swing signaling on the input wires.

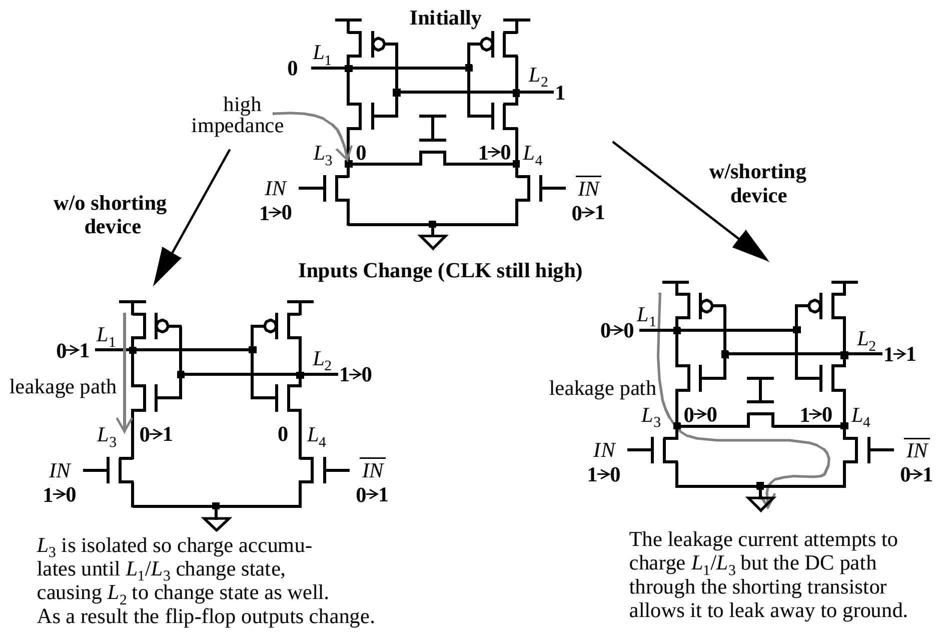

Initially
0
L_1
L_2
1
high impedance
L_3
0
1→0
L_4
IN
1→0
$\overline{IN}$
0→1
w/o shorting device
w/shorting device
Inputs Change (CLK still high)
L_1
0→1
L_2
1→0
leakage path
L_3
0→1
0
L_4
IN
1→0
$\overline{IN}$
0→1
L_3 is isolated so charge accumulates until L_1/L_3 change state, causing L_2 to change state as well. As a result the flip-flop outputs change.
L_1
0→0
L_2
1→1
leakage path
L_3
0→0
1→0
L_4
IN
1→0
$\overline{IN}$
0→1
The leakage current attempts to charge L_1/L_3 but the DC path through the shorting transistor allows it to leak away to ground.

Figure 7.39 The need for the shorting transistor M_4.

The shorting transistor, M_4, is used to provide a DC leakage path from either node L_3, or L_4, to ground. This is necessary to accommodate the case where the inputs change their value after the positive edge of *CLK* has occurred, resulting in either L_3 or L_4 being left in a high-impedance state with a logical low voltage level stored on the node. Without the leakage path that node would be susceptible to charging by leakage currents. The latch could then actually change state prior to the next rising edge of *CLK*! This is best illustrated graphically, as shown in Figure 7.39.

7.8 Pipelining: An approach to optimize sequential circuits

Pipelining is a popular design technique often used to accelerate the operation of the datapaths in digital processors. The idea is easily explained with the example of Figure 7.40a. The goal of the presented circuit is to compute log(|$a - b$|), where both a and b represent streams of numbers, that is, the computation must be performed on a large set of input values. The minimal clock period T_{min} necessary to ensure correct evaluation is given as:

$$T_{min} = t_{c\text{-}q} + t_{pd,\text{logic}} + t_{su} \tag{7.6}$$

where $t_{c\text{-}q}$ and t_{su} are the *propagation delay* and the *set-up time* of the register, respectively. We assume that the registers are *edge-triggered D* registers. The term $t_{pd,logic}$ stands for the worst-case delay path through the combinatorial network, which consists of the adder, absolute value, and logarithm functions. In conventional systems (that don't push the edge of technology), the latter delay is generally much larger than the delays associated with the registers and dominates the circuit performance. Assume that each logic module has an equal *propagation delay.* We note that each logic module is then active for only 1/3 of the clock period (if the delay of the register is ignored). For example, the adder unit is active during the first third of the period and remains idle—this is, it does no useful computation— during the other 2/3 of the period. *Pipelining* is a technique to improve the resource utilization, and increase the functional throughput. Assume that we introduce registers between the logic blocks, as shown in Figure 7.40b. This causes the computation for one set of input data to spread over a number of clock periods, as shown in Table 7.1. The

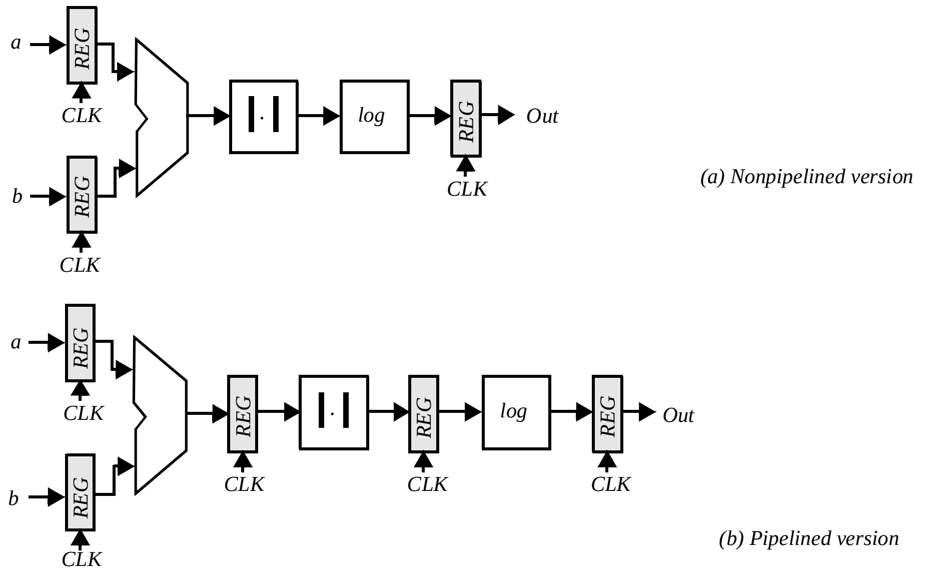


Figure 7.40 Datapath for the computation of log(|$a + b$|).

result for the data set (a_1, b_1) only appears at the output after three clock-periods. At that time, the circuit has already 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.

Table 7.1 Example of pipelined computations.

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

The advantage of pipelined operation becomes apparent when examining 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:

$$T_{min,\text{pipe}} = t_{c\text{-}q} + \max(t_{pd,\text{add}}, t_{pd,\text{abs}}, t_{pd,\text{log}}) \tag{7.7}$$

Suppose that all logic blocks have approximately the same *propagation delay*, and that the register overhead is small with respect to the logic delays. The pipelined network outperforms the original circuit by a factor of three under these assumptions, or $T_{min,pipe} = T_{min}/3$. The increased performance comes at the relatively small cost of two additional registers, and an increased latency.[1] This explains why pipelining is popular in the implementation of very high-performance datapaths.

7.8.1 Latch- vs. Register-Based Pipelines

Pipelined circuits can be constructed using *level-sensitive* latches instead of *edge-triggered* registers. Consider the pipelined circuit of Figure 7.41. The pipeline system is implemented based on pass-transistor-based *positive and negative* latches instead of *edge-triggered* registers. That is, logic is introduced between the master and slave latches of a *master-slave* system. In the following discussion, we use without loss of generality the CLK-$\overline{CLK}$ notation to denote a two-phase clock system. Latch-based systems give significantly more flexibility in implementing a pipelined system, and often offers higher performance. When the clocks CLK and $\overline{CLK}$ are nonoverlapping, correct pipeline operation is obtained. Input data is sampled on C_1 at the negative edge of CLK and the computation of logic block F starts; the result of the logic block F is stored on C_2 on the falling edge of $\overline{CLK}$, and the computation of logic block G starts. The nonoverlapping of the clocks

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

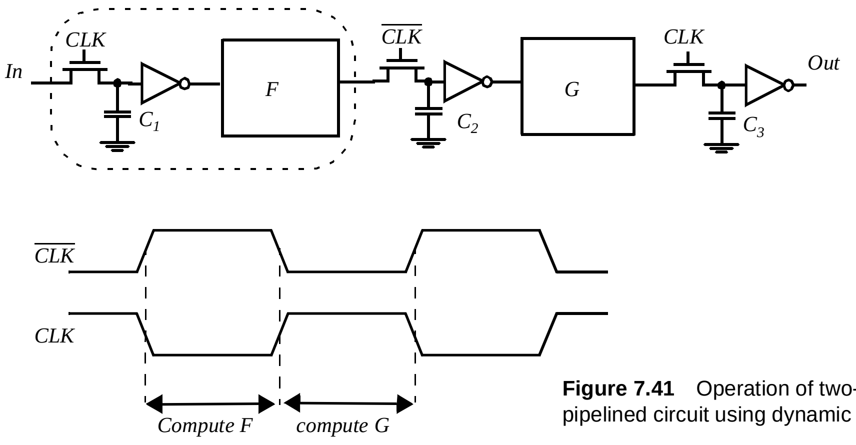


Figure 7.41 Operation of two-phase pipelined circuit using dynamic registers.

ensures correct operation. The value stored on C_2 at the end of the *CLK* low phase is the result of passing the previous input (stored on the falling edge of *CLK* on C_1) through the logic function *F*. When overlap exists between *CLK* and $\overline{CLK}$, the next input is already being applied to *F*, and its effect might propagate to C_2 before $\overline{CLK}$ goes low (assuming that the *contamination delay* of *F* is small). In other words, a *race* develops between the previous input and the current one. Which value wins depends upon the logic function *F*, the overlap time, and the value of the inputs since the *propagation delay* is often a function of the applied inputs. The latter factor makes the detection and elimination of race conditions non-trivial.

7.8.2 NORA-CMOS—A Logic Style for Pipelined Structures

The latch-based pipeline circuit can also be implemented using C^2MOS latches, as shown in Figure 7.42. The operation is similar to the one discussed above. This topology has one additional, important property:

> A C^2MOS-based pipelined circuit is race-free as long as all the logic functions *F* (implemented using static logic) between the latches are noninverting.

The reasoning for the above argument is similar to the argument made in the construction of a C^2MOS register. During a (0-0) overlap between *CLK* and $\overline{CLK}$, all C^2MOS latches, simplify to pure pull-up networks (see Figure 7.27). The only way a signal can race from stage to stage under this condition is when the logic function *F* is inverting, as illustrated in Figure 7.43, where *F* is replaced by a single, static CMOS inverter. Similar considerations are valid for the (1-1) overlap.

Based on this concept, a logic circuit style called *NORA-CMOS* was conceived [Goncalves83]. It combines C^2MOS pipeline registers and *NORA* dynamic logic function blocks. Each module consists of a block of combinational logic that can be a mixture of static and dynamic logic, followed by a C^2MOS latch. Logic and latch are clocked in such a way that both are simultaneously in either evaluation, or hold (precharge) mode. A block

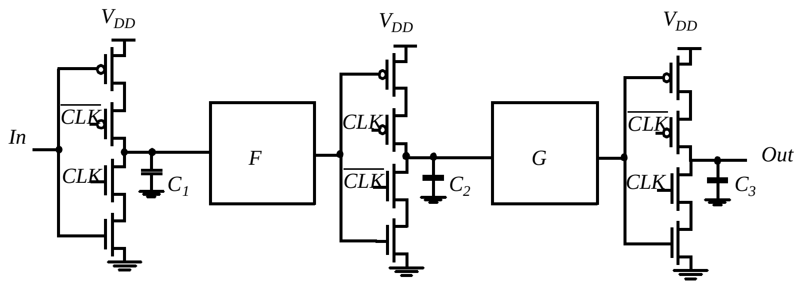


Figure 7.42 Pipelined datapath using C^2MOS latches.

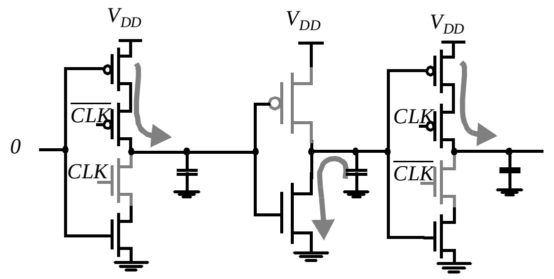


Figure 7.43 Potential race condition during (0-0) overlap in C^2MOS-based design.

that is in evaluation during CLK = 1 is called a *CLK-module*, while the inverse is called a *$\overline{CLK}$-module*. Examples of both classes are shown in Figure 7.44 a and b, respectively. The operation modes of the modules are summarized in Table 7.2.

Table 7.2 Operation modes for NORA logic modules.

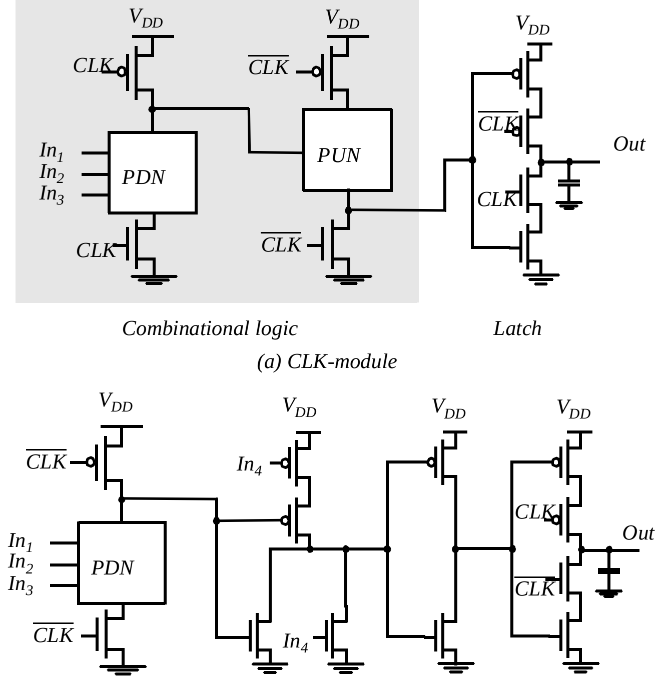
	CLK **block**		$\overline{CLK}$ **block**	
	Logic	**Latch**	**Logic**	**Latch**
CLK = 0	Precharge	Hold	Evaluate	Evaluate
CLK = 1	Evaluate	Evaluate	Precharge	Hold

A NORA datapath consists of a chain of alternating CLK and $\overline{CLK}$ modules. While one class of modules is precharging with its output latch in hold mode, preserving the previous output value, the other class is evaluating. Data is passed in a pipelined fashion from module to module.

NORA offers designers a wide range of design choices. Dynamic and static logic can be mixed freely, and both CLK_p and CLK_n dynamic blocks can be used in cascaded or in pipelined form. With this freedom of design, extra inverter stages, as required in DOMINO-CMOS, are most often avoided.

Design Rules

In order to ensure correct operation, two important rules should always be followed:

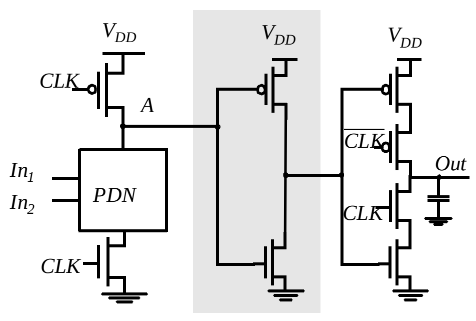


(b) $\overline{CLK}$-module

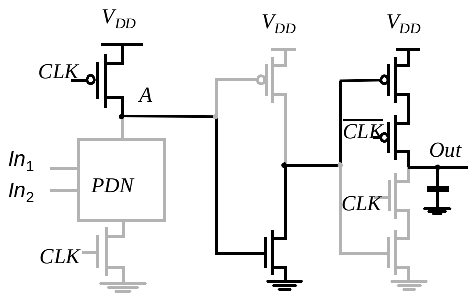
Figure 7.44 Examples of NORA CMOS Modules.

- **The dynamic-logic rule:** Inputs to a dynamic CLK_n (CLK_p) block are only allowed to make a single $0 \rightarrow 1$ ($1 \rightarrow 0$) transition during the evaluation period (Chapter 6).
- **The C^{2}MOS rule:** In order to avoid races, the number of static inversions between C^{2}MOS latches should be even.

The presence of dynamic logic circuits requires the introduction of some extensions to the latter rule. Consider the situation pictured in Figure 7.45a. During precharge (CLK = 0), the output register of the module has to be in hold mode, isolating the output node from the internal events in the module. Assume now that a (0-0) overlap occurs. Node A gets precharged to V_{DD}, while the latch simplifies to a pull-up network (Figure 7.45b). It can be observed that under those circumstances the output node charges to V_{DD}, and the stored value is erased! This malfunctioning is caused by the fact that the number of static inversions between the last dynamic node in the module and the latch is odd, which creates an active path between the precharged node and the output. This translates into the following rule: The number of static inversions between the last dynamic block in a logic function and the C^{2}MOS latch should be *even*. This and similar considerations lead to a reformulated C^{2}MOS rule [Goncalvez83].



(a) Circuit with odd number of static inversions between dynamic logic stage and register



(b) Same circuit during (0-0) clock overlap.

Figure 7.45 Extended C^2MOS rules.

Revised C^2MOS Rule

- The number of static inversions between C^2MOS latches should be even (in the absence of dynamic nodes); if dynamic nodes are present, the number of static inverters between a latch and a dynamic gate in the logic block should be even. The number of static inversions between the last dynamic gate in a logic block and the latch should be even as well.

Adhering to the above rules is not always trivial and requires a careful analysis of the logic equations to be implemented. This often makes the design of an operational NORA-CMOS structure cumbersome. Its use should only be considered when maximum circuit performance is a must.

7.9 Non-Bistable Sequential Circuits

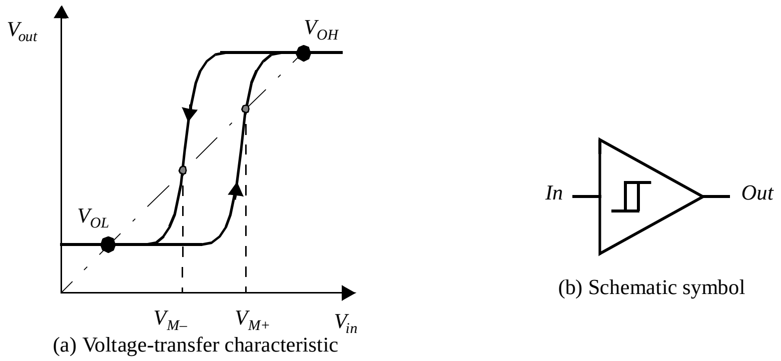
In the preceding sections, we have focused on one single type of sequential element, this is the latch (and its sibling the register). The most important property of such a circuit is that it has two stable states, and is hence called *bistable*. The *bistable* element is not the only sequential circuit of interest. Other regenerative circuits can be catalogued as *astable* and *monostable*. The former act as oscillators and can, for instance, be used for on-chip clock generation. The latter serve as pulse generators, also called *one-shot circuits*. Another interesting regenerative circuit is the Schmitt trigger. This component has the useful property of showing hysteresis in its dc characteristics—its switching threshold is variable and depends upon the direction of the transition (low-to-high or high-to-low). This peculiar feature can come in handy in noisy environments.

7.9.1 The Schmitt Trigger

Definition

A Schmitt trigger [Schmitt38] is a device with two important properties:

1. It responds to a slowly changing input waveform with a *fast transition time at the output*.
2. The voltage-transfer characteristic of the device displays *different switching thresholds for positive- and negative-going input signals*. This is demonstrated in Figure 7.46, where a typical voltage-transfer characteristic of the Schmitt trigger is shown (and its schematics symbol). The switching thresholds for the low-to-high and high-to-low transitions are called V_{M+} and V_{M-}, respectively. The *hysteresis voltage* is defined as the difference between the two.



(a) Voltage-transfer characteristic
(b) Schematic symbol

Figure 7.46 Non-inverting Schmitt trigger.

One of the main uses of the Schmitt trigger is to turn a noisy or slowly varying input signal into a clean digital output signal. This is illustrated in Figure 7.47. Notice how the hysteresis suppresses the ringing on the signal. At the same time, the fast low-to-high (and high-to-low) transitions of the output signal should be observed. For instance, steep signal slopes are beneficial in reducing power consumption by suppressing direct-path currents. The “secret” behind the Schmitt trigger concept is the use of positive feedback.

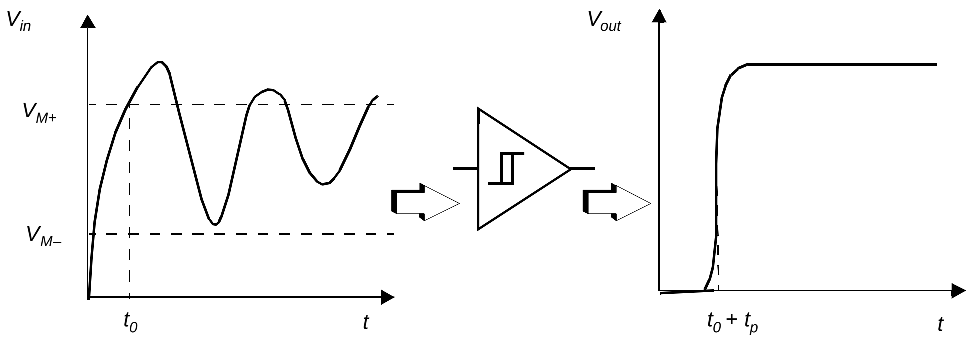


Figure 7.47 Noise suppression using a Schmitt trigger.

CMOS Implementation

One possible CMOS implementation of the Schmitt trigger is shown in Figure 7.48. The idea behind this circuit is that the switching threshold of a CMOS inverter is determined by the (k_n/k_p) ratio between the NMOS and PMOS transistors. Increasing the ratio results in a reduction of the threshold, while decreasing it results in an increase in $V_{M.}$ Adapting the ratio depending upon the direction of the transition results in a shift in the switching threshold and a hysteresis effect. This adaptation is achieved with the aid of feedback.

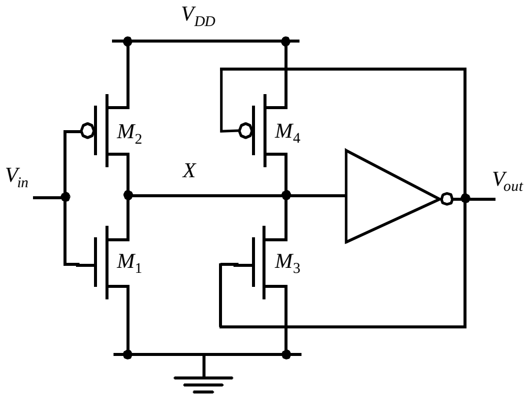


Figure 7.48 CMOS Schmitt trigger.

Suppose that V_{in} is initially equal to 0, so that V_{out} = 0 as well. The feedback loop biases the PMOS transistor M_4 in the conductive mode while M_3 is off. The input signal effectively connects to an inverter consisting of two PMOS transistors in parallel (M_2 and M_4) as a pull-up network, and a single NMOS transistor (M_1) in the pull-down chain. This modifies the effective transistor ratio of the inverter to $k_{M1}/(k_{M2}+k_{M4})$, which moves the switching threshold upwards.

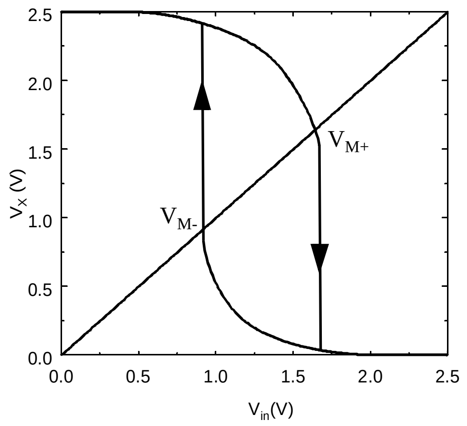
Once the inverter switches, the feedback loop turns off M_4, and the NMOS device M_3 is activated. This extra pull-down device speeds up the transition and produces a clean output signal with steep slopes.

A similar behavior can be observed for the high-to-low transition. In this case, the pull-down network originally consists of M_1 and M_3 in parallel, while the pull-up network is formed by M_2. This reduces the value of the switching threshold to V_{M-}.

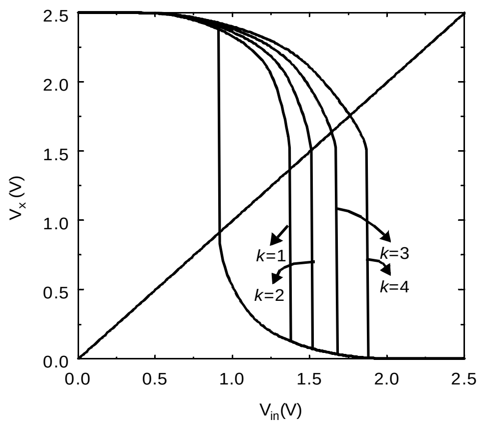
Example 7.7 CMOS Schmitt Trigger

Consider the schmitt trigger with the following device sizes. Devices M_1 and M_2 are 1µm/0.25µm, and 3µm/0.25µm, respectively. The inverter is sized such that the switching threshold is around $V_{DD}/2$ (= 1.25 V). Figure 7.49a shows the simulation of the Schmitt trigger assuming that devices M_3 and M_4 are 0.5µm/0.25µm and 1.5µm/0.25µm, respectively. As apparent from the plot, the circuit exhibits hysteresis. The high-to-low switching point (V_{M-} = 0.9 V) is lower than $V_{DD}/2$, while the low-to-high switching threshold (V_{M+} = 1.6 V) is larger than $V_{DD}/2$.

It is possible to shift the switching point by changing the sizes of M_3 and M_4. For example, to modify the low-to-high transition, we need to vary the PMOS device. The high-to-low threshold is kept constant by keeping the device width of M_3 at 0.5 µm. The device width of M_4 is varied as k * 0.5µm. Figure 7.49b demonstrates how the switching threshold increases with raising values of k.



(a) Voltage-transfer characteristics with hysteresis.



(b) The effect of varying the ratio of the PMOS device M_4. The width is k * 0.5µm.

Figure 7.49 Schmitt trigger simulations.

Problem 7.8 An Alternative CMOS Schmitt Trigger

Another CMOS Schmitt trigger is shown in Figure 7.50. Discuss the operation of the gate, and derive expressions for V_{M-} and V_{M+}.

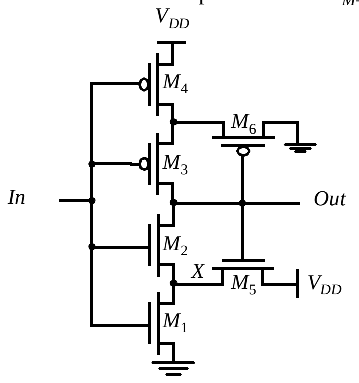


Figure 7.50 Alternate CMOS Schmitt trigger.

7.9.2 Monostable Sequential Circuits

A monostable element is a circuit that generates *a pulse of a predetermined width* every time the quiescent circuit is triggered by a pulse or transition event. It is called *monostable* because it has only one stable state (the quiescent one). A trigger event, which is either a signal transition or a pulse, causes the circuit to go temporarily into another quasi-stable state. This means that it eventually returns to its original state after a time period determined by the circuit parameters. This circuit, also called a *one-shot*, is useful in generating pulses of a known length. This functionality is required in a wide range of applications. We have already seen the use of a one-shot in the construction of glitch registers. Another notorious example is the *address transition detection* (ATD) circuit, used for the timing generation in static memories. This circuit detects a change in a signal, or group of signals, such as the address or data bus, and produces a pulse to initialize the subsequent circuitry.

The most common approach to the implementation of one-shots is the use of a simple delay element to control the duration of the pulse. The concept is illustrated in Figure 7.51. In the quiescent state, both inputs to the XOR are identical, and the output is low. A transition on the input causes the XOR inputs to differ temporarily and the output to go high. After a delay t_d (of the delay element), this disruption is removed, and the output goes low again. A pulse of length t_d is created. The delay circuit can be realized in many different ways, such as an *RC*-network or a chain of basic gates.

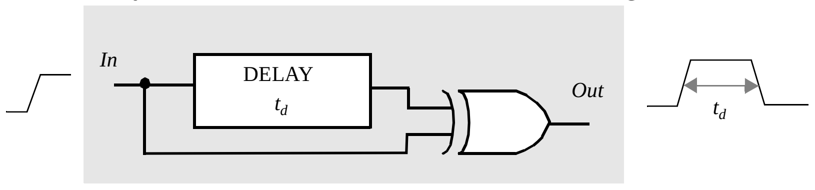


Figure 7.51 Transition-triggered one-shot.

7.9.3 Astable Circuits

An astable circuit has no stable states. The output oscillates back and forth between two quasi-stable states with a period determined by the circuit topology and parameters (delay, power supply, etc.). One of the main applications of oscillators is the on-chip generation of clock signals. This application is discussed in detail in a later chapter (on timing).

The ring oscillator is a simple, example of an astable circuit. It consists of an odd number of inverters connected in a circular chain. Due to the odd number of inversions, no stable operation point exists, and the circuit oscillates with a period equal to $2 \times t_p \times N$, with N the number of inverters in the chain and t_p the *propagation delay* of each inverter.

Example 7.8 Ring oscillator

The simulated response of a ring oscillator with five stages is shown in Figure 7.52 (all gates use minimum-size devices). The observed oscillation period approximately equals 0.5 nsec, which corresponds to a gate *propagation delay* of 50 psec. By tapping the chain at various points, different phases of the oscillating waveform are obtained (phases 1, 3, and 5 are dis-

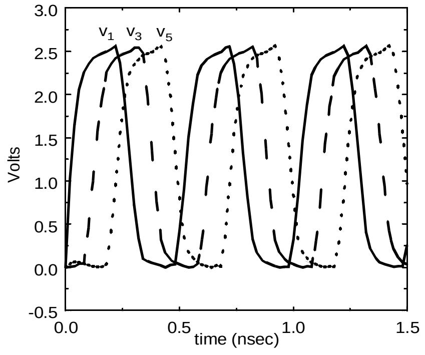


Figure 7.52 Simulated waveforms of five-stage ring oscillator. The outputs of stages 1, 3, and 5 are shown.

played in the plot). A wide range of clock signals with different duty-cycles and phases can be derived from those elementary signals using simple logic operations.

The ring oscillator composed of cascaded inverters produces a waveform with a fixed oscillating frequency determined by the delay of an inverter in the CMOS process. In many applications, it is necessary to control the frequency of the oscillator. An example of such a circuit is the *voltage-controlled oscillator (VCO)*, whose oscillation frequency is a function (typically non-linear) of a control voltage. The standard ring oscillator can be modified into a *VCO* by replacing the standard inverter with a *current-starved* inverter as shown in Figure 7.53 [Jeong87]. The mechanism for controlling the delay of each inverter is to limit the current available to discharge the load capacitance of the gate.

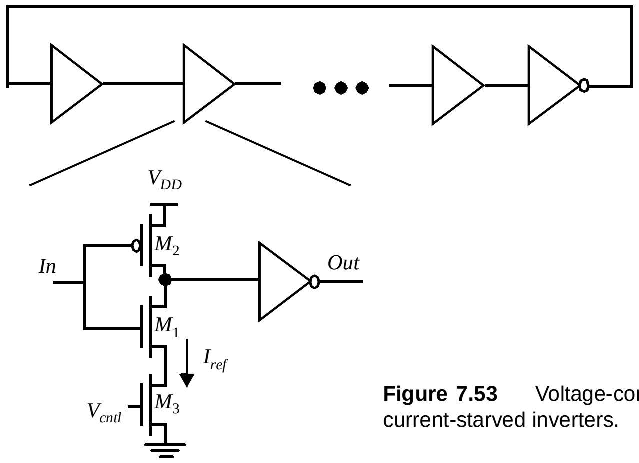


Figure 7.53 Voltage-controlled oscillator based on current-starved inverters.

In this modified inverter circuit, the maximal discharge current of the inverter is limited by adding an extra series device. Note that the low-to-high transition on the inverter can also be controlled by adding a PMOS device in series with M_2. The added NMOS transistor M_3, is controlled by an analog control voltage V_{cntl}, which determines the available discharge current. Lowering V_{cntl} reduces the discharge current and, hence, increases t_{pHL}. The ability to alter the *propagation delay* per stage allows us to control the frequency of the ring structure. The control voltage is generally set using feedback techniques. Under low operating current levels, the current-starved inverter suffers from slow fall times at its output. This can result in significant short-circuit current. This is resolved by feeding its output into a CMOS inverter or better yet a Schmitt trigger. An extra inverter is needed at the end to ensure that the structure oscillates.

Example 7.9 Current-Starved Inverter Simulation

Figure 7.54 show the simulated delay of the current-starved inverter as a function of the control voltage V_{cntl}. The delay of the inverter can be varied over a large range. When the control voltage is smaller than the threshold, the device enters the sub-threshold region. This results in large variations of the *propagation delay*, as the drive current is exponentially dependent on the drive voltage. When operating in this region, the delay is very sensitive to variations in the control voltage, and, hence, to noise.

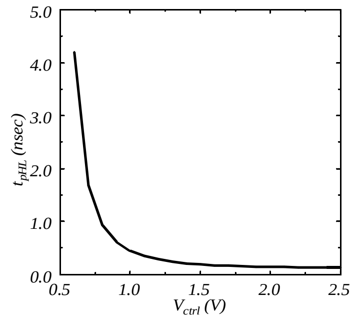


Figure 7.54 t_{pHL} of current-starved inverter as a function of the control voltage.

Another approach to implement the delay element is to use a differential element as shown in Figure 7.55a. Since the delay cell provides both inverting and non-inverting outputs, an oscillator with an even number of stages can be implemented. Figure 7.55b shows a two-stage differential *VCO*, where the feedback loop provides 180^o phase shift through two gate delays, one non-inverting and the other inverting, therefore forming an oscillation. The simulated waveforms of this two stage *VCO* are shown in Figure 7.55c. The in-phase and quadrature phase outputs are available simultaneously. The differential type *VCO* has better immunity to common mode noise (for example, supply noise) compared to the common ring oscillator. However, it consumes more power due to the increased complexity, and the static current.

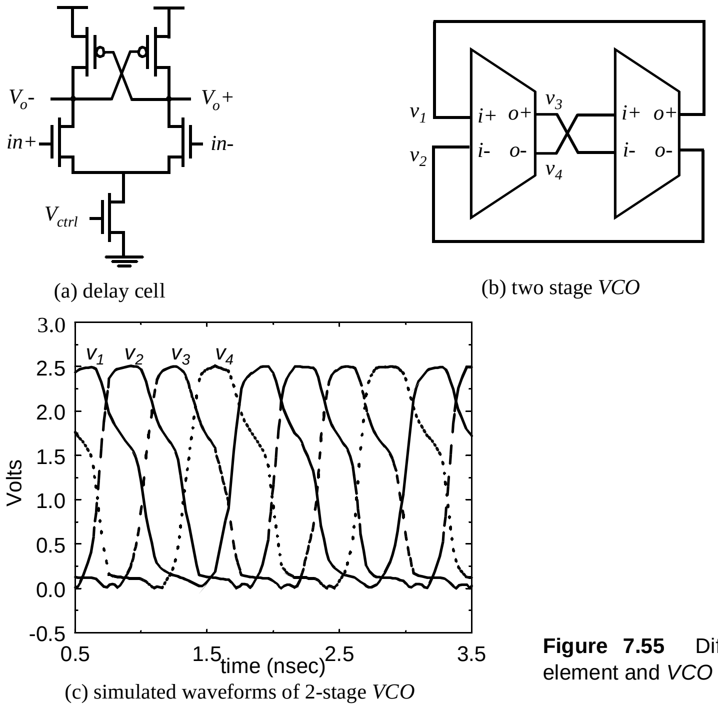


Figure 7.55 Differential delay element and *VCO* topology.

7.10 Perspective: Choosing a Clocking Strategy

A crucial decision that must be made in the earliest phases of a chip design is to select the appropriate clocking methodology. The reliable synchronization of the various operations occurring in a complex circuit is one of the most intriguing challenges facing the digital designer of the next decade. Choosing the right clocking scheme affects the functionality, speed and power of a circuit.

A number of widely-used clocking schemes were introduced in this chapter. The most robust and conceptually simple scheme is the two-phase *master-slave* design. The predominant approach is use the multiplexer-based register, and to generate the two clock phases locally by simply inverting the clock. More exotic schemes such as the glitch register are also used in practice. However, these schemes require significant hand tuning and must only be used in specific situations. An example of such is the need for a negative *set-up time* to cope with clock skew.

The general trend in high-performance CMOS VLSI design is therefore to *use simple clocking schemes*, even at the expense of performance. Most automated design methodologies such as standard cell employ a single-phase, *edge-triggered* approach, based on static flip-flops. But the tendency towards simpler clocking approaches is also apparent in high-performance designs such as microprocessors. The use of latches between logic is also very common to improve circuit performance.

7.11 Summary

This chapter has explored the subject of sequential digital circuits. The following topics were discussed:

- The cross-coupling of two inverters creates a *bistable* circuit, called a *flip-flop*. A third potential operation point turns out to be metastable; that is, any diversion from this bias point causes the flip-flop to converge to one of the stable states.
- A latch is a *level-sensitive* memory element that samples data on one phase and holds data on the other phase. A register (sometime also called a *flip-flop*) on the other hand samples the data on the rising or falling edge. A register has three important parameter *the set-up time, the hold time, and the propagation delay.* These parameters must be carefully optimized since they may account for a significant portion of the clock period.
- Registers can be *static* or *dynamic*. A static register holds state as long as the power supply is turned on. It is ideal for memory that is accessed infrequently (e.g., reconfiguration registers or control information). Dynamic memory is based on temporary charge store on capacitors. The primary advantage is the reduced complexity and higher performance/lower power. However, charge on a dynamic node leaks away with time, and hence dynamic circuits have a minimum clock frequency.
- There are several fundamentally different approaches towards building a register. The most common and widely used approach is the *master-slave configuration* which involves cascading a positive latch and negative latch (or vice-versa).

- Registers can also be constructed using the *pulse or glitch concept*. An intentional pulse (using a one shot circuit) is used to sample the input around an edge. Generally, the design of such circuits requires careful timing analysis across all process corners. Sense-amplifier based schemes are also used to construct registers and are to be used when high performance or low signal swing signalling is required.
- Choice of clocking style is an important consideration. Two phase design can result in race problems. Circuit techniques such as C^2MOS can be used to eliminate race conditions in two-phase clocking. Another option is to use true single phase clocking. However, the rise time of clocks must be carefully optimized to eliminate races.
- The combination of dynamic logic with dynamic latches can produce extremely fast computational structures. An example of such an approach, the NORA logic style, is very effective in pipelined datapaths.
- Monostable structures have only one stable state. They are useful as pulse generators.
- Astable multivibrators, or oscillators, possess no stable state. The ring oscillator is the best-known example of a circuit of this class.
- Schmitt triggers display hysteresis in their dc characteristic and fast transitions in their transient response. They are mainly used to suppress noise.

7.12 To Probe Further

The basic concepts of sequential gates can be found in many logic design textbooks (e.g., [Mano82] and [Hill74]). The design of sequential circuits is amply documented in most of the traditional digital circuit handbooks.

References

[Dopperpuhl92] D. Dopperpuhl et al., “A 200 MHz 64-b Dual Issue CMOS Microprocessor,” *IEEE JSSC*, vol. 27, no. 11, Nov. 1992, pp. 1555–1567.

[Gieseke97] B. Bieseke et al., “A 600MHz Superscalar RISC Microprocessor with Out-Of-Order Execution,” *IEEE ISSCC*, pp. 176-177, Feb. 1997.

[Goncalves83] N. Goncalves and H. De Man, “NORA: a racefree dynamic CMOS technique for pipelined logic structures,” *IEEE JSSC*, vol. SC-18, no. 3, June 1983, pp. 261–266.

[Haznedar91] H. Haznedar, *Digital Microelectronics*, Benjamin/Cummings, 1991.

[Hill74] F. Hill and G. Peterson, *Introduction to Switching Theory and Logical Design*, Wiley, 1974.

[Hodges88] D. Hodges and H. Jackson, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill, 1988.

[Jeong87] D. Jeong et al., “Design of PLL-based clock generation circuits,” *IEEE JSSC*, vol. SC-22, no. 2, April 1987, pp. 255–261.

[Kuzo96] S. Kuzo et al., “A 100MHz 0.4W RISC Processor with 200MHz Multiply-Adder, using Pulse-Register Technique,” *IEEE ISSCC*, pp. 140-141, February 1996.

[Mano82] M. Mano, *Computer System Architecture*, Prentice Hall, 1982.

[Montanaro96] J. Montanaro et al., “A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor,” *IEEE JSSC*, pp. 1703-1714, November 1996.

[Mutoh95] S. Mutoh et al., “1-V Power Supply High-Speed Digital Circuit Technology with Multi-threshold-Voltage CMOS,” *IEEE JSSC* , pp. 847-854, August 1995.

[Partovi96] H. Partovi, “Flow-Through Latch and *Edge-Triggered* Flip Flop Hybrid Elements,” *IEEE ISSCC*, pp. 138-139, February 1996.

[Schmitt38] O. H. Schmitt, “A Thermionic Trigger,” *Journal of Scientific Instruments*, vol. 15, January 1938, pp. 24–26.

[Shoji88] M. Shoji, *CMOS Digital Circuit Technology*, Prentice Hall, 1988.

[Suzuki73] Y. Suzuki, K. Odagawa, and T. Abe, “Clocked CMOS calculator circuitry,” *IEEE Journal of Solid State Circuits*, vol. SC-8, December 1973, pp. 462–469.

[Veendrick92] H. Veendrick, *MOS ICs: From Basics to ASICs*, VCH, Weinheim, 1992.

[Yuan89] J. Yuan and Svensson C., “High-Speed CMOS Circuit Technique,” *IEEE JSSC*, vol. 24, no. 1, February 1989, pp. 62–70.